22nd International Conference on Database Theory

ICDT 2019, March 26–28, 2019, Lisbon, Portugal

Edited by

Pablo Barcelo

Marco Calautti

Editors

Pablo Barcelo
Department of Computer Science, Universidad de Chile, CL
pbarcelo@dcc.uchile.cl

Marco Calautti
School of Informatics, University of Edinburgh, UK
mcalautt@inf.ed.ac.uk

ACM Classification 2012
Computing Methodologies → Knowledge Representation and Reasoning; Theory of computation → Data modeling; Theory of computation → Incomplete, inconsistent, and uncertain databases; Information systems → Data management systems; Information systems → Data streams; Information systems → Database query processing; Information systems → Incomplete data; Information systems → Inconsistent data; Information systems → Relational database model; Information systems → Relational database query languages; Mathematics of computing → Graph theory; Theory of computation → Complexity theory and logic; Theory of computation → Data structures and algorithms for data management; Theory of computation → Semantics and reasoning

ISBN 978-3-95977-101-6

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-101-6.

Publication date
March, 2019

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

Digital Object Identifier: 10.4230/LIPIcs.ICDT.2019.0

ISBN 978-3-95977-101-6 **ISSN 1868-8969** **https://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

LIPIcs is a series of high-quality conference proceedings across all fields in informatics. LIPIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Invited Talks

Regular Papers

22nd International Conference on Database Theory (ICDT 2019).
Editors: Pablo Barcelo and Marco Calautti
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

The 22nd International Conference on Database Theory (ICDT 2019) was held in Lisbon, Portugal, March 26–28, 2019. The proceedings of ICDT 2019 include an extended abstract of a keynote by Dan Olteanu (University of Oxford), an extended abstract of a keynote by Lise Getoor (University of California, Santa Cruz), a paper by Markus Kroetzsch (Technical University of Dresden) and coauthors based on his invited tutorial, a laudation concerning the ICDT 2019 Test of Time Award, and 21 research papers that were selected by the Program Committee from 67 submissions.

The Program Committee selected the paper *Counting Triangles under Updates in Worst-Case Optimal Time* by Ahmet Kara, Hung Q. Ngo, Milos Nikolic, Dan Olteanu and Haozhe Zhang for the ICDT 2019 Best Paper Award. Furthermore, a specially designated committee formed by Wenfei Fan, Magdalena Ortiz, and Ke Yi decided to give the Test of Time Award for ICDT 2019 to the paper *Automatic verification of data-centric business processes* by Alin Deutsch, Richard Hull, Fabio Patrizi, and Victor Vianu (originally published in ICDT 2009). We sincerely congratulate to the authors of these award winning papers!

We would like to thank all people who contributed to the success of ICDT 2019, including authors who submitted papers, keynote and tutorial speakers, and, of course, all members of the Program Committee, and the external reviewers, for the very dedicated work they have done over the two submission cycles of ICDT 2019. Their help and assistance were crucial to ensure that the final program of the conference keeps satisfying the highest standards.

We would also like to thank the ICDT Council members for their support on a wide variety of matters concerning ICDT, and the local organizers of the EDBT/ICDT 2019 conference, leaded by General Chair Helena Galhardas (Universidade de Lisboa, Portugal), for their constant help in the organization of the conference and some co-located events. Finally, we wish to acknowledge Dagstuhl Publishing for their support with the publication of the proceedings in the LIPIcs (Leibniz International Proceedings in Informatics) Series.

Pablo Barcelo and Marco Calautti
March 2019

22nd International Conference on Database Theory (ICDT 2019).
Editors: Pablo Barcelo and Marco Calautti

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Organization

Conference Chair

Wim Martens (University of Bayreuth)

Program Chair

Pablo Barcelo (University of Chile)

Proceedings and Publicity Chair

Marco Calautti (University of Edinburgh)

Program Committee Members

Isolde Adler (University of Leeds)

Antoine Amarilli (Telecom ParisTech)

Paul Beame (University of Washington)

Elena Botoeva (Imperial College London)

Victor Dalmau (Universitat Pompeu Fabra)

Diego Figueira (Labri, CNRS)

Dominik D. Freydenberger (Loughborough University)

Paolo Guagliardo (University of Edinburgh)

Zengfeng Huang (The University of New South Wales)

Benny Kimelfeld (Technion, Israel)

Paris Koutris (University of Wisconsin)

Maurizio Lenzerini (Sapienza University of Rome)

Yakov Nekrich (University of Waterloo)

Matthias Niewerth (University of Bayreuth)

Cristian Riveros (PUC, Chile)

Sebastian Skritek (TU Vienna)

Julia Stoyanovich (Drexel University)

Szymon Torunczyk (University of Warsaw)

Ke Yi (HKUST)

Thomas Zeume (University of Dortmund)

22nd International Conference on Database Theory (ICDT 2019).
Editors: Pablo Barcelo and Marco Calautti
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

External Reviewers

Giovanni Amendola
Christoph Berkholz
Leopoldo Bertossi
Marco Console
Shaleen Deep
Johannes Doleschal
Nadime Francis
Travis Gagie
Floris Geerts
Boris Glavic
Tomasz Gogacz
Alejandro Grez
Aidan Hogan
Jean Christoph Jung
Egor V. Kostylev
Sławomir Lasota
Leonid Libkin
Ester Livshits
Brendan Lucier
Mikaël Monet
Frank Neven
Miguel Romero Orth
Paweł Parys
Thomas Pellissier Tanon
Liat Peterfreund
Andreas Pieris
Juan L. Reutter
Vladislav Ryzhikov
Emanuel Sallinger
Luc Segoufin
Mantas Simkus
Dan Suciu
Stijn Vansummeren
Nils Vortmeier
Domagoj Vrgoc
Zhewei Wei
Xuan Wu

22nd International Conference on Database Theory (ICDT 2019).
Editors: Pablo Barcelo and Marco Calautti
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Contributing Authors

Antoine Amarilli
Nuno Bandeira
Gerald Berger
Leopoldo Bertossi
Pierre Bourhis
Marco Calautti
Balder ten Cate
Marco Console
Christopher De Sa
Ronald Fagin
Arnab Ganguly
Floris Geerts
Lise Getoor
Grzegorz Gluch
Georg Gottlob
Alejandro Grez
Emilie Grienenberger
Paolo Guagliardo
Ihab F. Ilyas
Suraj Jog
Ahmet Kara
Mahmoud Abo Khamis
Benny Kimelfeld
Paraschos Koutris
Markus Krötzsch
Nirman Kumar
Michel Leclère
Yuliang Li
Leonid Libkin
Silviu Maniu
Jerzy Marcinkowski
Maximilian Marx
Stefan Mengel
Marie-Laure Mugnier
Ian Munro
Yakov Nekrich
Frank Neven
Hung Q. Ngo
Matthias Niewerth
Milos Nikolic
Dan Olteanu
Piotr Ostropolski-Nalewaja
Martin Otto
Yannis Papakonstantinou
Liat Peterfreund
Reinhard Pichler
Andreas Pieris
Benjamin Pullman
Benjamin Raichel
Christopher Ré
Theodoros Rekatsinas
Martin Ritzert
Cristian Riveros
Sebastian Rudolph
Thomas Schwentick
Pierre Senellart
Rahul Shah
Stavros Sintos
Sebastian Skritek
Christopher Spinrath
Dan Suciu
Dimitri Surinx
Sharma Thankachan
Michaël Thomazo
Martín Ugarte
Federico Ulliana
Gregory Van Buskirk
Jan Van den Bussche
Brecht Vandevoort
Jianguo Wang
Jef Wijsen
Haozhe Zhang

22nd International Conference on Database Theory (ICDT 2019).
Editors: Pablo Barcelo and Marco Calautti
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

ICDT 2019 Test of Time Award

In 2013, the International Conference on Database Theory (ICDT) began awarding the ICDT test-of-time (ToT) award, with the goal of recognizing one paper, or a small number of papers, presented at ICDT a decade earlier that have best met the "test of time". In 2019, the award recognizes a paper from the ICDT 2009 proceedings that has had the most impact in terms of research, methodology, conceptual contribution, or transfer to practice over the past decade. The award was presented during the EDBT/ICDT 2019 Joint Conference, March 26-29, 2019, in Lisbon, Portugal.

The 2019 ToT Committee consists of Wenfei Fan (chair), Magdalena Ortiz, and Ke Yi. After careful consideration and soliciting external assessments, the committee has chosen the following recipient of the 2019 ICDT Test of Time Award:

Automatic verification of data-centric business processes
Alin Deutsch, Richard Hull, Fabio Patrizi, Victor Vianu

The paper has been a cornerstone in the research on artifact-centric and data-aware processes. It opens up the possibility of verifying data-aware processes not only in niche cases but also in quite broad classes. It provides a formalization of IBM's artifact-based approach to business processes, and models arbitrary inputs from external users with infinite domains. This gives rise to processes that have infinite states and hence are impossible to verify through standard model checking techniques. In addition to this formalization, it investigates verification of properties in an extension of LTL over artifacts, which is challenging since it deals with infinite alphabets.

The paper has generated impact not only on database and business process communities, but also influenced artificial intelligence, verification and Web services. It has received over 250 citations coming from these diverse communities. In addition, its techniques have influenced theoretical work on fundamental aspects of languages over infinite alphabets.

Wenfei Fan	Magdalena Ortiz	Ke Yi
University of Edinburgh	Vienna University of Technology (TU Wien)	Hong Kong University of Science and Technology

The ICDT Test-of-Time Award Committee for 2019

22nd International Conference on Database Theory (ICDT 2019).
Editors: Pablo Barcelo and Marco Calautti

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Learning Models over Relational Databases

Dan Olteanu
Department of Computer Science, University of Oxford, Oxford, UK
dan.olteanu@cs.ox.ac.uk

Abstract

In this talk, I will make the case for a first-principles approach to machine learning over relational databases that exploits recent development in database systems and theory.

The input to learning classification and regression models is defined by feature extraction queries over relational databases. The mainstream approach to learning over relational data is to materialize the training dataset, export it out of the database, and then learn over it using statistical software packages. These three steps are expensive and unnecessary. Instead, one can cast the machine learning problem as a database problem by decomposing the learning task into a batch of aggregates over the feature extraction query and by computing this batch over the input database.

The performance of this database-centric approach benefits tremendously from structural properties of the relational data and of the feature extraction query; such properties may be algebraic (semi-ring), combinatorial (hypertree width), or statistical (sampling). It also benefits from database systems techniques such as factorized query evaluation and query compilation. For a variety of models, including factorization machines, decision trees, and support vector machines, this approach may come with lower computational complexity than the materialization of the training dataset used by the mainstream approach. Recent results show that this translates to several orders-of-magnitude speed-up over state-of-the-art systems such as TensorFlow, R, Scikit-learn, and mlpack.

While these initial results are promising, there is much more awaiting to be discovered.

2012 ACM Subject Classification Theory of computation → Database query processing and optimization (theory); Information systems → Database query processing; Computing methodologies → Supervised learning; Computing methodologies → Machine learning approaches

Keywords and phrases In-database analytics, Data complexity, Feature extraction queries, Database dependencies, Model reparameterization

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.1

Category Invited Talk

Funding This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 682588.

Acknowledgements This work is part of the FDB project (`https://fdbresearch.github.io`) and based on collaboration with Maximilian Schleich (Oxford), Mahmoud Abo-Khamis, Ryan Curtin, Hung Q. Ngo (RelationalAI), Ben Moseley (CMU), and XuanLong Nguyen (Michigan).

The Power of Relational Learning

Lise Getoor
Computer Science Department, University of California, Santa Cruz, US
getoor@soe.ucsc.edu

Abstract

We live in a richly interconnected world and, not surprisingly, we generate richly interconnected data. From smart cities to social media to financial networks to biological networks, data is relational. While database theory is built on strong relational foundations, the same is not true for machine learning. The majority of machine learning methods flatten data into a single table before performing any processing. Further, database theory is also built on a bedrock of declarative representations. The same is not true for machine learning, in particular deep learning, which results in black-box, uninterpretable and unexplainable models. In this talk, I will introduce the field of statistical relational learning, an alternative machine learning approach based on declarative relational representations paired with probabilistic models. I'll describe our work on probabilistic soft logic, a probabilistic programming language that is ideally suited to richly connected, noisy data. Our recent results show that by building on state-of-the-art optimization methods in a distributed implementation, we can solve very large relational learning problems orders of magnitude faster than existing approaches.

2012 ACM Subject Classification Computing methodologies → Machine learning; Information systems → Data management systems

Keywords and phrases Machine learning, Probabilistic soft logic, Relational model

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.2

Category Invited Talk

22nd International Conference on Database Theory (ICDT 2019).
Editors: Pablo Barcelo and Marco Calautti; Article No. 2; pp. 2:1–2:1

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The Power of the Terminating Chase

Markus Krötzsch
TU Dresden, Germany
markus.kroetzsch@tu-dresden.de

Maximilian Marx
TU Dresden, Germany
maximilian.marx@tu-dresden.de

Sebastian Rudolph
TU Dresden, Germany
sebastian.rudolph@tu-dresden.de

Abstract

The chase has become a staple of modern database theory with applications in data integration, query optimisation, data exchange, ontology-based query answering, and many other areas. Most application scenarios and implementations require the chase to terminate and produce a finite universal model, and a large arsenal of sufficient termination criteria is available to guarantee this (generally undecidable) condition. In this invited tutorial, we therefore ask about the expressive power of logical theories for which the chase terminates. Specifically, which database properties can be recognised by such theories, i.e., which Boolean queries can they realise? For the skolem (semi-oblivious) chase, and almost any known termination criterion, this expressivity is just that of plain Datalog. Surprisingly, this limitation of most prior research does not apply to the chase in general. Indeed, we show that standard–chase terminating theories can realise queries with data complexities ranging from PTime to non-elementary that are out of reach for the terminating skolem chase. A "Datalog-first" standard chase that prioritises applications of rules without existential quantifiers makes modelling simpler – and we conjecture: computationally more efficient. This is one of the many open questions raised by our insights, and we conclude with an outlook on the research opportunities in this area.

2012 ACM Subject Classification Theory of computation → Database query languages (principles); Theory of computation → Complexity theory and logic; Theory of computation → Database constraints theory; Theory of computation → Logic and databases

Keywords and phrases Existential rules, Tuple-generating dependencies, all-instances chase termination, expressive power, data complexity

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.3

Category Invited Talk

Funding This work is partly supported by the German Research Foundation (DFG) in CRC 248 (Perspicuous Systems), CRC 912 (HAEC), and Emmy Noether grant KR 4381/1-1; and by the European Research Council (ERC) Consolidator Grant 771779 (DeciGUT).

Acknowledgements We thank David Carral for his comments on an earlier version of this paper.

1 Introduction

Forty years ago, the first versions of the *chase* algorithm were developed, initially as a way of deciding implication problems of specific classes of dependencies [26, 2], and soon thereafter as a proof system for the more general (and undecidable) case of *tuple-generating* and *equality-generating dependencies* (TGDs and EGDs) [6, 7]. As of today, the chase is understood as an iterative method for constructing *universal models* of Horn logic theories [19], which corresponds to the computation of most general solutions to logical entailment problems. Such

universal solutions have many natural applications in data integration, query optimisation, data exchange, and query answering under constraints. The re-discovery of TGDs as *existential rules*, and their application to ontological modelling and knowledge representation has motivated many further studies [3, 11, 15].

Of course, our practical interest is usually in cases where the chase terminates and the resulting model is therefore finite, corresponding to a database that has been "repaired" to satisfy the given dependencies. Unfortunately, chase termination is undecidable, no matter if we ask for the termination on a particular database [7] or for "universal" termination on all databases [22, 23]. Nonetheless, practical implementations of chase procedures have been successfully applied in many contexts [21, 4, 9, 8, 35]. It is then up to the user to provide terminating inputs, and this is facilitated by many decidable sufficient criteria for ensuring chase termination, which have been developed for this purpose [20, 27, 3, 25, 15, 12].

This brings up a natural question: given an efficient practical implementation of some variant of the chase, what computational problems can we solve with it? That is: what is the expressive power of logical theories for which the chase terminates? To answer this, we first need to clarify what the *input* is: the theory and database, or merely the database while the theory is fixed? These two views can be associated with traditional perspectives from formerly separated and now converging fields of computer science [31]. The first perspective is that of knowledge representation, where logical theories ("ontologies") are exchanged and combined, and reasoning over such theories is necessary to access the encoded knowledge.

The second perspective is common in database theory, where theories are comparatively small and static, whereas databases are large and dynamic. This is also reflected in many current chase implementations, which treat logical theories as "programs," usually in a tool-specific format that prevents any kind of interchange, while supporting data-level interoperability with standard sources (such as relational DBMS or RDF stores). The data-centric view is also taken by most studies on termination criteria, which aim to identify theories for which the chase will terminate with *every* database instance.

We can therefore specify our question as follows: what is the expressivity of logical theories for which the chase terminates on all database instances? Surprisingly little is known about this, and what we can conclude from previous results is thoroughly disappointing: if our chase variant is the skolem (a.k.a. semi-oblivious) chase, expressivity is essentially limited to traditional Datalog [27, 25, 36]. The vast majority of known termination criteria applies to the skolem chase [15], and is therefore facing the same limitations. In spite of the significant challenges that existential quantifiers introduce to rule-based reasoning, it seems that our current approaches to ensuring decidability cannot offer substantial advantages regarding the problems that one can solve in practice.

Surprisingly, this apparent weakness of the chase is specific to the skolem chase and the termination criteria that are tailored to this variant. We show this by demonstrating

1. that there are PTime decision problems that are not expressible in Datalog, and hence neither via terminating skolem chase, but that can be solved in the terminating standard (a.k.a. restricted) chase, and
2. that the terminating standard chase can solve decision problems of non-elementary (data) complexity, whereas Datalog and the skolem chase have PTime data complexity.

Therefore, the standard chase is superior to the skolem chase regarding tractable as well as highly complex computations. Each implementation of this algorithm inherits this power – as long as one is willing to give up the restriction to skolem-chase terminating theories.

As another contribution, we investigate a particularly natural class of chase strategies, where Datalog rules (containing no existential quantification) are always applied before any

true existential rule. This *Datalog-first* strategy simplifies the construction of universally terminating theories, and in fact is the only variant of the chase for which we could solve decision problems as in (1) in polynomial time. Although we give an encoding of (1) that terminates in the standard chase, this might require exponentially many steps (in spite of the query describing a property of databases recognizable in PTime).

A major consequence of our study is that focussing research activities related to chase termination on the skolem chase is arguably too restrictive, since one relinquishes significant expressive power when doing so. This insight is complemented by recent empirical studies proving the standard chase almost always more efficient in practical implementations [8, 35]. This might be surprising, given that a standard chase step is computationally more demanding [23]; but the reduction in redundant computation seems to compensate for this cost even in cases where termination does not rely on it. Shifting the focus away from the skolem chase also reveals how little we know about more complex chase variants, and raises a number of open questions for future research.

After a quick overview of preliminary definitions (Section 2), we review several important chase variants (Section 3), and the state of the art regarding their termination (Section 4). Thereafter, we make the previously established expressive limits of the skolem chase explicit (Section 5). We then continue to demonstrate that these limits can be overcome both on queries in PTime (Section 6) and on queries of considerably higher complexity (Section 7). We conclude with an overview of open questions and conjectures (Section 8).

2 Preliminaries

We briefly introduce the necessary concepts and terminology. For a more thorough introduction, we refer to [1, 14]. We construct expressions from countably infinite, mutually disjoint sets $\mathbf{V}$ of *variables*, $\mathbf{F}$ of *(skolem) function symbols*, $\mathbf{N}$ of *labelled nulls*, and $\mathbf{R}$ of *relation names*. Each function or relation name $s \in \mathbf{F} \cup \mathbf{R}$ has an *arity* $\mathsf{ar}(s) \geq 0$. Function symbols of arity 0 are *constants*, and we set $\mathbf{C} \coloneqq \{c \in \mathbf{F} \mid \mathsf{ar}(c) = 0\}$. *Terms* are either elements of $\mathbf{V} \cup \mathbf{N} \cup \mathbf{C}$, or, recursively, expressions $f(t_1, \ldots, t_n)$ with $f \in \mathbf{F}$, $\mathsf{ar}(f) = n$, and $t_1, \ldots, t_n$ terms. We generally use $\boldsymbol{t}$ to denote a list $t_1, \ldots, t_{|\boldsymbol{t}|}$ of terms, and similar for special types of terms. An *atom* is an expression $r(\boldsymbol{t})$ with $r \in \mathbf{R}$, $\boldsymbol{t}$ a list of terms, and $\mathsf{ar}(r) = |\boldsymbol{t}|$. *Ground* terms or atoms contain neither variables nor nulls.

Rules and queries. An *existential rule* (or just *rule*) ρ is a formula

$$\rho = \forall \boldsymbol{x}, \boldsymbol{y}.\ \varphi[\boldsymbol{x}, \boldsymbol{y}] \rightarrow \exists \boldsymbol{z}.\ \psi[\boldsymbol{y}, \boldsymbol{z}], \tag{1}$$

where φ and ψ are conjunctions of atoms using only terms from $\mathbf{C}$ or from the mutually disjoint lists of variables $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \subseteq \mathbf{V}$. We call φ the *body* (denoted $\mathsf{body}(\rho)$), ψ the *head* (denoted $\mathsf{head}(\rho)$), and $\boldsymbol{y}$ the *frontier* of ρ. We often treat conjunctions of atoms like sets, and omit the universal quantifiers in rules. A rule is *Datalog* if it has no existential quantifiers. A *conjunctive query* (CQ) $q[\boldsymbol{x}]$ with free variables $\boldsymbol{x}$ is a formula $\exists \boldsymbol{y}.\varphi[\boldsymbol{x}, \boldsymbol{y}]$, where φ is a conjunction of atoms using only constants and variables from $\boldsymbol{x} \cup \boldsymbol{y}$. A *boolean conjunctive query* (BCQ) is a CQ without free variables. Since CQ answering and BCQ answering are polynomially reducible to one another [3], we restrict our attention to BCQs.

Databases and morphism. A *database* $\mathcal{I}$ is a set of atoms without variables. A *concrete database* $\mathcal{D}$ is a finite database without function symbols, except for constants. Given a

set of atoms $\mathcal{A}$ and database $\mathcal{I}$, a *homomorphism* $h : \mathcal{A} \to \mathcal{I}$ is a function that maps the terms occurring in $\mathcal{A}$ to (the variable-free) terms occurring in $\mathcal{I}$, such that: (i) for all $f \in \mathbf{F}$: $h(f(\boldsymbol{t})) = f(h(\boldsymbol{t}))$; (ii) for all $r \in \mathbf{R}$: if $r(\boldsymbol{t}) \in \mathcal{A}$, then $r(h(\boldsymbol{t})) \in \mathcal{I}$, where $h(\boldsymbol{t})$ is the list of h-images of the terms $\boldsymbol{t}$. A homomorphism h is *strong* if $r(\boldsymbol{t}) \in \mathcal{A} \iff r(h(\boldsymbol{t})) \in \mathcal{I}$ for all $r \in \mathbf{R}$, and an *embedding* if it is strong and injective.

Semantics of rules and queries. A match of a rule ρ in a database $\mathcal{I}$ is a function h from the universally quantified variables of ρ to $\mathcal{I}$ that restricts to a homomorphism $\mathsf{body}(\rho) \to \mathcal{I}$.[1] A match h of ρ in $\mathcal{I}$ is *satisfied* if there is a homomorphism $h' : \mathsf{head}(\rho) \to \mathcal{I}$ that agrees with h on all frontier variables. Rule ρ is *satisfied* by $\mathcal{I}$, written $\mathcal{I} \models \rho$, if every match of ρ in $\mathcal{I}$ is satisfied. A set of rules Σ is satisfied by $\mathcal{I}$, written $\mathcal{I} \models \Sigma$, if $\mathcal{I} \models \rho$ for all $\rho \in \Sigma$. We may treat a concrete database $\mathcal{D}$ as sets of rules with empty bodies (also called *facts*), and write, e.g., $\mathcal{I} \models \mathcal{D}, \Sigma$ to express that $\mathcal{I} \models \Sigma$ and $\mathcal{D} \subseteq \mathcal{I}$. In this case, $\mathcal{I}$ is a *model* of Σ and $\mathcal{D}$. A BCQ $q = \exists \boldsymbol{x}.\varphi[\boldsymbol{x}]$ is *satisfied* by a database $\mathcal{I}$, written $\mathcal{I} \models q$, if there is a homomorphism $\varphi[\boldsymbol{x}] \to \mathcal{I}$; it is *entailed* by a concrete database $\mathcal{D}$ and rule set Σ, written $\mathcal{D}, \Sigma \models q$, if $\mathcal{I} \models q$ for every $\mathcal{I}$ with $\mathcal{I} \models \mathcal{D}, \Sigma$ (note that for empty rule sets, satisfaction coincides with entailment). Since homomorphisms are closed under composition, the existence of homomorphisms $q \to \mathcal{I}$ and $\mathcal{I} \to \mathcal{J}$ implies that q is satisfied by $\mathcal{J}$: the set of databases that satisfy a BCQ is *closed under homomorphisms*.

Expressivity. When discussing expressivity of query languages, it is convenient to have an abstract notion of (boolean) queries. An *abstract query* over a given finite signature $\mathbf{R}^{\mathrm{EDB}} \subseteq \mathbf{R}$ of so-called *extensional database relations* is a set $\mathfrak{D}$ of concrete databases over $\mathbf{R}^{\mathrm{EDB}}$. We say that a given set of rules Σ and BCQ q *realise* $\mathfrak{D}$ if for every database $\mathcal{D}$ over $\mathbf{R}^{\mathrm{EDB}}$ holds $\mathcal{D}, \Sigma \models q$ exactly if $\mathcal{D} \in \mathfrak{D}$. Note that Σ and q can make use of other (so-called *intensional database*) relations not in $\mathbf{R}^{\mathrm{EDB}}$. This notion of query realisation allows us to focus on input databases which are "well-formed" in terms of the relation names used.

For an arbitrary set of rules Σ, checking $\mathcal{D}, \Sigma \models q$ is undecidable, however, the set $\{\mathcal{D} \mid \mathcal{D}, \Sigma \models q\}$ is recursively enumerable and closed under homomorphisms. In fact, for every homomorphism-closed, recursively enumerable abstract query $\mathfrak{D}$, there exist Σ and q that realise $\mathfrak{D}$ [32].

3 Universal Models and the Chase

BCQ entailment (and CQ answering) can be solved by computing *universal models* [19]. A model $\mathcal{I}$ of a set of rules Σ is universal if it admits a homomorphism $h : \mathcal{I} \to \mathcal{J}$ to every model $\mathcal{J}$ of Σ. Due to closure under homomorphisms, BCQs that are entailed by a universal model are entailed by every model. Since the converse is also true, universal models characterise BCQ entailment.[2] For this reason (among others), many algorithms for computing universal models have been developed. Their basic approach, known as the *chase*, is to construct such models bottom-up by applying rules to facts. We consider several variants of this approach: the *standard* (a.k.a. *restricted*) *chase* [20], the *skolem* (a.k.a. *semi-oblivious*) *chase* [27], and the *core chase* [19].

[1] If all frontier variables occur in body atoms (i.e., the rule is *safe*), h is a homomorphism $\mathsf{body}(\rho) \to \mathcal{I}$ itself; the chosen definition accommodates the possibility of unsafe rules.

[2] We remark that universal models are generally neither unique, nor the only models that characterise BCQ entailment (see [13] for some discussion).

We first define the standard and skolem chases. For a rule ρ of form (1), the *skolemised rule* $\mathrm{sk}(\rho)$ is $\varphi[\boldsymbol{x},\boldsymbol{y}] \to \psi[\boldsymbol{y},\boldsymbol{t_z}]$, where $\psi[\boldsymbol{y},\boldsymbol{t_z}]$ is obtained from $\psi[\boldsymbol{y},\boldsymbol{z}]$ by replacing each variable $z \in \boldsymbol{z}$ with a skolem term $f(\boldsymbol{y})$ using a fresh skolem function symbol f. Notions that were defined for existential rules naturally extend to skolemised rules. For uniformity, the next definition also treats skolemised rules as formulas of the form (1), where $\boldsymbol{z}$ is empty and ψ may contain functions instead.

▶ **Definition 1.** *A* chase sequence *for a concrete database $\mathcal{D}$ and a set of existential or skolemised rules Σ is a potentially infinite sequence $\mathcal{D}^0, \mathcal{D}^1, \ldots$ such that*

(1) *$\mathcal{D}^0 = \mathcal{D}$;*

(2) *for every $\mathcal{D}^i$ with $i \geq 0$, there is a match h for some rule $\rho = \varphi[\boldsymbol{x},\boldsymbol{y}] \to \exists \boldsymbol{z}.\psi[\boldsymbol{y},\boldsymbol{z}] \in \Sigma$ in $\mathcal{D}^i$ such that*

- **a.** *h is an unsatisfied match in $\mathcal{D}^i$ (i.e., h cannot be extended to a homomorphism $\psi \to \mathcal{D}^i$), and*
- **b.** *$\mathcal{D}^{i+1} = \mathcal{D}^i \cup \psi[h'(\boldsymbol{y}), h'(\boldsymbol{z})]$, where $h' : \psi \to \mathcal{D}^{i+1}$ is such that $h'(y) = h(y)$ for all $y \in \boldsymbol{y}$, and for all $z \in \boldsymbol{z}$, $h'(z) \in \mathbf{N}$ is a distinct labelled null that does not occur in $\mathcal{D}^i$ (in particular, h is a satisfied match in $\mathcal{D}^{i+1}$).*

(3) *if h is a match for a rule $\rho \in \Sigma$ and $\mathcal{D}^i$ ($i \geq 0$), then there is $j > i$ such that h is satisfied in $\mathcal{D}^j$* (fairness).

The chase *for such a chase sequence is the database $\bigcup_{i \geq 0} \mathcal{D}^i$.*

Note that this definition allows individual rule applications to occur in any (fair) order, and in particular does not require that all matches for a rule are processed together. This generality makes sense to cover all current implementations, some of which deploy parallelised, streaming rule applications that might result in such an interleaved derivation order [30].

Given a set Σ of existential rules, a chase for Σ is called *standard chase*, and a chase for $\mathrm{sk}(\Sigma)$ is called *skolem chase*. A *Datalog-first chase* is a standard chase where non-Datalog rules are applied in step (2) only if all matches for Datalog rules in $\mathcal{D}^{i-1}$ are satisfied. Standard and Datalog-first chases might not be unique, due to the dependence of condition (2.a) on the order of rule applications, i.e., on the chase *strategy*.

▶ **Example 2.** Consider the concrete database $r(a,b)$ and the rules

$$r(x,y) \to \exists v.r(y,v) \tag{2}$$

$$r(x,y) \to r(y,y) \tag{3}$$

In the standard chase, we can apply rule (2) with the match $\{x \mapsto a, y \mapsto b\}$ to derive $r(b,n_1)$, where n_1 is a new null. Applying (3) to $\{x \mapsto a, y \mapsto b\}$ yields $r(b,b)$, which would have blocked the first rule application if it had been computed earlier. Depending on the order of further rule applications in the given strategy, the standard chase might terminate after arbitrarily many steps, or fail to terminate altogether. In contrast, the Datalog-first chase prioritises (3), and therefore terminates with the model $\{r(a,b), r(b,b)\}$. Regarding the entailed BCQs, all of these results (including the infinite r-chain with loops) are equivalent.

The Datalog-first chase restricts to a sensible class of strategies, but it does not free the chase from its dependence on a selected strategy. For example, standard and Datalog-first chases coincide whenever there are no Datalog rules, which is easy to achieve by extending rule heads with redundant existential statements, e.g., we could replace (3) by $r(x,y) \to \exists v.r(y,y) \wedge r(x,v)$. In contrast, the skolem chase is always unique, since (2.a) is equivalent to $\psi[h(\boldsymbol{y})] \not\subseteq \mathcal{D}^{i-1}$ in this case.

▶ **Example 3.** Skolemising Example 2, rule (2) turns into $r(x, y) \to r(y, f(y))$ for some skolem function f. The skolem chase is the infinite database $\{r(a, b), r(b, b), r(b, f(b)), r(f(b), f(b)), r(f(b), f(f(b))), \ldots\}$.

Further variations of these chases have been defined in the literature [8]. For example, the *parallel chase* computes $\mathcal{D}^{i+1}$ from $\mathcal{D}^i$ by considering all matches (w.r.t. $\mathcal{D}^i$) in step (2). $\mathcal{D}^{i+1}$ then contains new derivations from many rule applications, even if some of them could have prevented the application of others. The *1-parallel chase* is similar, but only considers the matches of one rule in each step. These modifications lead to variants of the standard and Datalog-first chase, but do not affect the skolem chase.

An important extension of the parallel standard chase is the *core chase*, where the database is reduced to a core after each derivation step [19]. A database $\mathcal{I}$ is a *core* if every homomorphism $h : \mathcal{I} \to \mathcal{I}$ is an embedding. Given a database $\mathcal{J}$, a *core of* $\mathcal{J}$ is a core obtained by restricting $\mathcal{J}$ to its image under some appropriate homomorphism $h : \mathcal{J} \to \mathcal{J}$ [24, 5]. Every finite database $\mathcal{D}$ has a unique core (up to isomorphism),[3] which we denote by $\mathsf{core}(\mathcal{D})$.

▶ **Definition 4.** *A* core chase sequence *for a concrete database $\mathcal{D}$ and a set of existential rules Σ is a maximal sequence $\mathcal{D} = \mathcal{D}^0, \mathcal{D}^1, \ldots,$ such that D^i is not isomorphic to D^{i+1} and $\mathcal{D}^{i+1} = \mathsf{core}(\Sigma(\mathcal{D}^i))$, where $\Sigma(\mathcal{D}^i)$ is the result of applying all rules $\rho \in \Sigma$ that have a match for $\mathcal{D}^i$ as in step (2) of Definition 1. If this sequence is finite, then its final element D^ℓ is the* core chase*, which is then unique up to isomorphism.*

The key advantage of this more complex algorithm is that it characterises when a set of rules admits a finite universal model over a given database:

▶ **Theorem 5** ([19])**.** *A concrete database $\mathcal{D}$ and a set of existential rules Σ admit a finite universal model if and only if the core chase terminates (producing such a model).*

4 Chase Termination

In practice, we are particularly interested in cases where the chase produces a finite universal model. In this section, we discuss this situation and review several criteria for recognising it effectively. We start by defining the most important types of termination.

Let Σ be a set of existential rules and a concrete database $\mathcal{D}$. Σ has an *all-strategies terminating* standard chase on $\mathcal{D}$ if all chase sequences for Σ and $\mathcal{D}$ are finite. Adopting notation of Grahne and Onet [23], we write $\mathsf{CT}^{\mathsf{std}}_{\mathcal{D}\forall}$ for the class of all such rule sets. Analogously, $\mathsf{CT}^{\mathsf{dlf}}_{\mathcal{D}\forall}$ denotes the class of all *Datalog-first terminating* rule sets. These notions can be generalised to require *all-instances* termination:[4] $\mathsf{CT}^{\mathsf{std}}_{\forall\forall} = \bigcap_{\mathcal{D}} \mathsf{CT}^{\mathsf{std}}_{\mathcal{D}\forall}$ and $\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall} = \bigcap_{\mathcal{D}} \mathsf{CT}^{\mathsf{dlf}}_{\mathcal{D}\forall}$. For the skolem chase and for the core chase, the chosen strategy does not affect termination, hence we simplify notation and write $\mathsf{CT}^{\mathsf{sk}}_{\mathcal{D}}$ and $\mathsf{CT}^{\mathsf{sk}}_{\forall}$, respectively, and similarly for the core chase. It is known that $\mathsf{CT}^{\mathsf{sk}}_{\forall} \subset \mathsf{CT}^{\mathsf{std}}_{\forall\forall} \subset \mathsf{CT}^{\mathsf{core}}_{\forall}$ [23], and it is similarly easy to see that we also have $\mathsf{CT}^{\mathsf{std}}_{\forall\forall} \subset \mathsf{CT}^{\mathsf{dlf}}_{\forall\forall} \subset \mathsf{CT}^{\mathsf{core}}_{\forall}$.

Termination is generally undecidable, often not even recursively enumerable [22, 23],[5] but many sufficient criteria have been proposed. Most existing works give criteria for inclusion in

[3] Infinite databases can have several non-isomorphic cores, or none at all [13].

[4] Note that this includes instances $\mathcal{D}$ that do not only contain extensional database relations, i.e., termination is also required if the input database is not "well-formed."

[5] For the Datalog-first chase, one can reduce from the undecidable termination problems for the standard chase by extending Datalog rules with irrelevant existential quantifiers.

$\mathsf{CT}^{\mathsf{sk}}_{\forall}$ [20, 27, 28, 25, 3, 15], and some can also be used for data-dependent classes $\mathsf{CT}^{\mathsf{sk}}_{\mathcal{D}}$ [28, 15]. Comparatively few works propose criteria that exploit the better termination behaviour of the standard chase [19, 28, 12]. We give a more detailed overview below.

Recent works established the decidability of termination for syntactically restricted classes of rules: $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ is decidable on *sticky rules* [10], while $\mathsf{CT}^{\mathsf{std}}_{\forall\forall}$ is decidable on linear rules [29] and also on guarded rules [34]. An even stronger notion than all-instances termination is *k-boundedness*, which requires that any fact in the chase can be derived by at most k rule applications. This criterion is known to be decidable for skolem and standard chase [18].

4.1 Termination of the Skolem Chase

In general, the question "$\Sigma \in \mathsf{CT}^{\mathsf{sk}}_{\mathcal{D}}$?" is clearly semi-decidable by simply running the chase, i.e., $\mathsf{CT}^{\mathsf{sk}}_{\mathcal{D}}$ is recursively enumerable. Marnette observed that all-instance termination can be reduced to this special case by considering what he called the *critical instance* [27].

▶ **Definition 6.** *Let Σ be a rule set, and denote by* $\mathrm{const}(\Sigma)$ *the sets of constants occurring in Σ. The* critical instance *for Σ is the concrete database $\mathcal{D}^*$ consisting of all atoms of form $r(\boldsymbol{t})$, where r is a relation name occurring in Σ and $\boldsymbol{t}$ is a list of constants from $\{*\} \cup \mathrm{const}(\Sigma)$, with $*$ a fresh constant. By a slight abuse of notation, we write $\mathsf{CT}^{\mathsf{sk}}_{\mathcal{D}^*}$ for the class of all rule sets for which the skolem chase on their respective critical instance terminates.*

▶ **Theorem 7** ([27])**.** $\mathsf{CT}^{\mathsf{sk}}_{\mathcal{D}^*} = \mathsf{CT}^{\mathsf{sk}}_{\forall}$.

Therefore, $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ is recursively enumerable, and, moreover, any technique for establishing skolem-chase termination on a specific instance can be used to establish all-instance termination. Many such techniques have been proposed. We can readily turn the chase itself into a decidable termination criterion if we ensure that the computation will halt even if the chase does not terminate. For example, we can stop the skolem chase when a *cyclic skolem term* occurs, i.e., a term of the form $f(t_1, \ldots, t_n)$ where f also occurs in some t_i. This termination criterion was called *MFA* (*model-faithful acyclicity*) and yields one of the largest decidable classes of skolem-chase terminating rules known today [15].

MFA is decidable but 2ExpTime-complete, and the chase might become double exponential before a cyclic term occurs. Many easier-to-check criteria are obtained by abstracting from the exact chase sequence and adapting the notion of *cycle* accordingly. If we identify terms that use the same outermost function symbol (equivalently: replace skolem terms by constants), we obtain *MSA* (*model-summarising acyclicity*), which is ExpTime-complete [15]. *Super-weak acyclicity* (SWA) [27] and *joint acyclicity* (JA) [25] replace terms by *places* and *positions*, respectively, both of which describe terms based on the general shape of inferred facts they might occur in. These possible facts can be (over-)estimated in polynomial time, and both criteria are PTime-complete. Even simpler – namely NL-complete – is *weak-acyclicity* (WA), which over-estimates term propagation by solving a reachability problem [20]. In terms of generality, the notions form a total order:

$$\mathrm{WA} \subset \mathrm{JA} \subset \mathrm{SWA} \subset \mathrm{MSA} \subset \mathrm{MFA} \subset \mathsf{CT}^{\mathsf{sk}}_{\forall}$$

Another way of abstracting MFA is to consider dependencies among rules instead of propagation of terms, i.e., we ask whether the conclusion produced by one rule might make another rule applicable. Termination is guaranteed if we thus obtain an *acyclic graph of rule dependencies* [3]. While subsumed by MFA, this NP-complete criterion is incomparable to other term-based notions [15]. Various kinds of dependencies of rules can be used to decompose rule sets into *strata*, which can then be analysed independently for termination [19, 28, 3].

Other sufficient criteria for skolem-chase termination were proposed. However, to the best of our knowledge, our listing includes all of the best-performing basic proposals in terms of complexity/expressivity trade-off.[6]

4.2 Termination of the Standard and Core Chase

Much less is known for the standard chase. Skolem chase termination yields a sufficient condition, since $\mathsf{CT}^{\mathsf{sk}}_{\forall} \subset \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$, but very few criteria take advantage of the standard chase's stronger inclination to terminate. Deutsch et al. define rule dependencies that are specific to the standard chase [19], and obtain a criterion for all-instances termination of the standard chase under *some* strategies. A corrected version that works for all strategies was proposed by Meier et al. [28].

These early dependency-based termination conditions were combined with weak acyclicity, i.e., term-level cycle detection remained confined to conditions for $\mathsf{CT}^{\mathsf{sk}}_{\forall}$. The difficulty is that the critical instance can no longer be used to detect universal termination, since the standard chase always terminates on this instance. More generally, the set of databases on which the standard chase with a given rule set terminates is no longer closed under homomorphisms – adding more facts may lead to termination.

Carral et al. recently proposed a way of approaching this problem [12]. Their notions of *restricted JA* (RJA) and *restricted MFA* (RMFA) show membership in $\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}$, i.e., for arbitrary instances and Datalog-first strategies, while also covering rule sets that are not in $\mathsf{CT}^{\mathsf{std}}_{\forall\forall}$ (or $\mathsf{CT}^{\mathsf{sk}}_{\forall}$). The essential idea for RMFA is to perform a modified skolem chase that, like MFA, starts from the critical instance and checks for cyclic terms. However, a match of a skolem rule is only applied after verifying that the match is not necessarily satisfied in all Datalog-first chase sequences. To this end, rule applications are retraced to find the most general set of facts from which the considered rule match can follow – a "universal premise" of the match that has a homomorphism into any chase where this match occurs. The Datalog-first strategy is honoured by further closing these facts under the given Datalog rules. If the match is satisfied by this Datalog-closed, universal premise, then the match is *blocked* and will not be used. The approach for RJA uses a similar block check to restrict the estimated propagation of terms along positions in JA.

It is easy to generalise RMFA and RJA to arbitrary (not necessarily Datalog-first) strategies by simply omitting the closure under Datalog rules. However, the wider practical applicability shown empirically by Carral et al. often hinges on the presence of Datalog rules that make the use of other existential rules obsolete.

For the core chase, due to Theorem 5, termination criteria correspond to general criteria for showing that a rule set has a finite universal model. We are not aware of any specific criteria that were proposed for this case while covering cases that are not in $\mathsf{CT}^{\mathsf{std}}_{\forall\forall}$.

5 The Weakness of the Terminating Skolem Chase

Already when Marnette first proposed the skolem chase, he observed an important feature that is tied to its all-instances termination: the size of the chase of any rule set in $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ is polynomial in the size of the underlying concrete database [27]. It follows that the data complexity of BCQ answering over $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ is in PTime, but also that $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ is inherently limited

[6] There can be no "most general termination criterion" for any complexity class that is closed under finite variations, so "best-performing" can only refer to empirical coverage of rule sets found in practice [15].

in terms of its expressive power. In this section, we elaborate on this limitation and relate it to more recent insights on the expressivity limits of Datalog. Let us start by explaining Marnette's result:

▶ **Theorem 8** ([27]). *For every $\Sigma \in \mathsf{CT}^{\mathsf{sk}}_{\forall}$ and concrete database $\mathcal{D}$, the skolem chase over Σ and $\mathcal{D}$ is polynomial in the size of $\mathcal{D}$. The data complexity of BCQ entailment over $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ is* PTime*-complete.*

Proof. We expand on the idea underlying the use of the critical instance in Theorem 7. Let $\mathsf{chase}_{\mathsf{sk}}(\Sigma, \mathcal{D})$ denote the skolem chase over Σ and $\mathcal{D}$. For any concrete database $\mathcal{D}$, let $h : \mathcal{D} \to \mathcal{D}^*$ be the unique mapping that (i) is identical on $\mathsf{const}(\Sigma)$, (ii) maps all other constants to $*$, and (iii) satisfies $h(f(\boldsymbol{t})) = f(h(\boldsymbol{t}))$ for all $f \in \mathbf{F}$ with $\mathsf{ar}(f) \geq 1$. Then h extends to a mapping $h' : \mathsf{chase}_{\mathsf{sk}}(\Sigma, \mathcal{D}) \to \mathsf{chase}_{\mathsf{sk}}(\Sigma, \mathcal{D}^*)$ satisfying (i)–(iii), as is easy to show by induction along the length of the chase sequence. In particular, $\mathsf{chase}_{\mathsf{sk}}(\Sigma, \mathcal{D})$ contains only terms for which a term of the same nesting structure occurs in $\mathsf{chase}_{\mathsf{sk}}(\Sigma, \mathcal{D}^*)$. The number k of distinct terms in $\mathsf{chase}_{\mathsf{sk}}(\Sigma, \mathcal{D}^*)$ is finite, and there is a largest number ℓ of occurrences of constants in these terms. Therefore, if $\mathcal{D}$ contains n distinct nulls and constants, and relation names of arity $\leq a$, then $\mathsf{chase}_{\mathsf{sk}}(\Sigma, \mathcal{D})$ contains $\leq kn^{\ell}$ terms and $\leq (kn^{\ell})^a$ facts, which is polynomial in n. The number of possible matches of any rule body likewise is polynomial in n, and $\mathsf{chase}_{\mathsf{sk}}(\Sigma, \mathcal{D})$ can therefore be computed in polynomial time. PTime-hardness follows from the known data complexity of Datalog. ◀

Theorem 8 already establishes that rule sets in $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ cannot express properties that are not in PTime, such as defining an ExpTime-hard set of concrete databases. However, we can obtain even tighter limits. Krötzsch and Rudolph define a transformation from jointly acyclic rule sets (a subset of $\mathsf{CT}^{\mathsf{sk}}_{\forall}$) to Datalog that preserves entailment of (similarly rewritten) BCQs [25]. Zhang et al. observed that this idea generalises to all of $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ and that BCQ rewriting is not necessary if we allow for a set of weakly acyclic "output" rules on top of Datalog [36]. In summary, we find that $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ has the same expressivity as Datalog:

▶ **Theorem 9.** *For every $\Sigma \in \mathsf{CT}^{\mathsf{sk}}_{\forall}$ and BCQ q, there is a set of Datalog rules Σ' and BCQ q' such that $\{\mathcal{D} \mid \mathcal{D}, \Sigma \models q\} = \{\mathcal{D} \mid \mathcal{D}, \Sigma' \models q'\}$.*

Proof sketch. As observed in the proof of Theorem 8, the size of terms occurring in any chase of Σ is bounded. We can therefore encode terms as bounded-length lists of elements (that occur as leaves in the term tree), new auxiliary constants (to encode function symbols of arity > 1), and the special constant $\square$ (to fill unused positions in a list). If the maximal arity of skolem functions is k and the maximal nesting depth of functions in the critical-instance chase is ℓ, then each term can be represented as a list of $k^{\ell} + k^{\ell-1} + 1$ elements (corresponding to the total number of leaves and inner nodes of a tree of depth ℓ and branching factor k). For example, if $k = 2$ and $\ell = 2$, then the fact $p(f(a, g(b)))$ can be "flattened" to $\hat{p}(f, a, \square, \square, g, b, \square)$. It is not hard to modify all (skolemised) rules accordingly. ◀

It is clear that $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ does not capture PTime (since existential rules can only express properties that are preserved under homomorphisms), but Theorem 9 further confines $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ to the expressiveness boundaries of Datalog, which cannot even express every homomorphism-closed Boolean query in PTime. Dawar and Kreutzer show that the following query cannot be realised by Datalog [17]:

▶ **Definition 10.** *Consider the following decision problem:*

Input: A directed graph G with two distinguished vertices s and t *Question: Is G cyclic, or is there a simple path from s to t of length $2^{2^{n^2}}$ for some $n \in \mathbb{N}$?*

The DK query $\mathfrak{D}_{\text{DK}}$ *is the abstract Boolean query containing exactly those concrete databases that encode an instance of this decision problem using a binary relation* `edge` *and constant symbols* `s` *and* `t`*.*

$\mathfrak{D}_{\text{DK}}$ is closed under homomorphisms: homomorphisms preserve cycles, and simple paths are either preserved or mapped to sub-structures with cycles. Moreover, the DK query can be checked in polynomial time, where we note that the length of any simple path is at most linear in the size of the input. Therefore, since the DK query is not expressible in Datalog [17], Theorem 9 implies that there are homomorphism-closed PTime queries that cannot be expressed in $\mathsf{CT}^{\mathsf{sk}}_{\forall}$:

▶ **Theorem 11.** *The DK query* $\mathfrak{D}_{\text{DK}}$ *is homomorphism-closed and in* PTime*, yet it is not realised by any* $\Sigma \in \mathsf{CT}^{\mathsf{sk}}_{\forall}$ *and BCQ* q*.*

6 Beyond the Skolem Chase

Surprisingly, the mechanisms that limit the expressive power of $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ are specific to the skolem chase. Other chase variants are not confined in such ways, and can break both the PTime-barrier of Theorem 8 and surpass the expressive power of Datalog within PTime. In this section, we focus on the latter aspect by discussing several techniques of expressing the Dawar-Kreutzer query of Definition 10. We start with a result for the Datalog-first chase:

▶ **Theorem 12.** *There is a rule set* $\Sigma \in \mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}$ *and BCQ* q *that realise the DK query. Moreover, the Datalog-first chase on* Σ *is polynomial in the size of the input database.*

Note that Theorem 12 establishes two independent results: (1) that the terminating Datalog-first chase is more expressive than the terminating skolem chase even on polynomial-time queries; and (2) that it will terminate in polynomial time in spite of this increased expressivity. For the standard chase, we will also establish property (1), but we conjecture that (2) cannot be attained.

Proof of Theorem 12. Inspired by a technique from Rudolph and Thomazo [33], we provide a rule set $\Sigma \in \mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}$ with the required properties in Figure 1, where the corresponding BCQ is `goal` (a query with a nullary relation name). The first group of rules (4)–(6) computes the transitive closure `path` of `edge`, and derives `goal` if there is a cycle.

Rules (7)–(16) check for the other condition of the DK query by measuring the length of paths from `s` to `t`. We define an initial `zero` element (7) and assign it to measure the distance of `s` from itself (8). Representatives of numbers larger than zero are created by adding successors (9), used in rule (10) to measure the `s`-distance of further vertices reached via `edge`. Note that a vertex might be assigned more than one distance if it is reachable through paths of different lengths. Two details are significant: (i) successors are only created for elements that are already used as distances (9), which ensures that the creation of successors terminates naturally in acyclic graphs; and (ii) establishing the relation of vertices to distances in rule (10) is independent of the creation of new successor elements in (9), which ensures that `succ` forms a unique chain, used globally to measure distances. Together, (i) and (ii) imply that only a linear number of new elements are created in acyclic graphs, even though such graphs might contain an exponential number of distinct paths.

Using the linear order of `succ`, rules (11)–(15) axiomatise arithmetic relationships in the usual way. Relation names have the expected intended meaning: $\texttt{add}(x, y, z)$ means "$x + y = z$", $\texttt{mul}(x, y, z)$ means "$x * y = z$", and $\texttt{exp}(x, y)$ means "$2^x = y$." Finally, rule (16) derives `goal` if an `s`–`t` path of the required length is discovered.

$$\texttt{edge}(v_1, v_2) \to \texttt{path}(v_1, v_2) \tag{4}$$
$$\texttt{edge}(v_1, v_2) \wedge \texttt{path}(v_2, v_3) \to \texttt{path}(v_1, v_3) \tag{5}$$
$$\texttt{path}(v, v) \to \texttt{goal} \tag{6}$$
$$\to \exists x.\texttt{zero}(x) \tag{7}$$
$$\texttt{zero}(x) \to \texttt{dist}(\texttt{s}, x) \tag{8}$$
$$\texttt{dist}(v, x) \to \exists x'.\texttt{succ}(x, x') \tag{9}$$
$$\texttt{dist}(v_1, x_1) \wedge \texttt{edge}(v_1, v_2) \wedge \texttt{succ}(x_1, x_2) \to \texttt{dist}(v_2, x_2) \tag{10}$$
$$\texttt{zero}(x) \wedge \texttt{dist}(v, y) \to \texttt{add}(x, y, y) \wedge \texttt{mul}(x, y, x) \tag{11}$$
$$\texttt{add}(x, y, z) \wedge \texttt{succ}(x, x') \wedge \texttt{succ}(z, z') \to \texttt{add}(x', y, z') \tag{12}$$
$$\texttt{mul}(x, y, z) \wedge \texttt{succ}(x, x') \wedge \texttt{add}(z, y, z') \to \texttt{mul}(x', y, z') \tag{13}$$
$$\texttt{zero}(x) \wedge \texttt{succ}(x, x') \to \texttt{exp}(x, x') \tag{14}$$
$$\texttt{exp}(x, y) \wedge \texttt{succ}(x, x') \wedge \texttt{add}(y, y, y') \to \texttt{exp}(x', y') \tag{15}$$
$$\texttt{mul}(x, x, y) \wedge \texttt{exp}(y, y') \wedge \texttt{exp}(y', z) \wedge \texttt{dist}(\texttt{t}, z) \to \texttt{goal} \tag{16}$$
$$\texttt{dist}(v, x) \wedge \texttt{goal} \to \texttt{succ}(x, x) \tag{17}$$

Figure 1 Rule set in $\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}$ that expresses the DK query, with termination guaranteed after polynomially many chase steps.

As argued above, the rules terminate polynomially on acyclic graphs (even when using the skolem chase). If there are cycles, however, paths can be of unbounded length, leading to a non-terminating creation of successor elements in (9). This is prevented by rule (17), which entails `succ`-loops on all distances once `goal` was derived, thus blocking further applications of (9) in the standard chase. In the Datalog-first chase, cycles will be detected before creating any elements, and `succ`-loops are established before considering (9). Therefore, this chase terminates after polynomially many steps on cyclic graphs as well. ◀

Although the rules of Figure 1 are in $\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}$, they do not fall into the RMFA fragment of Carral et al. [12]. Indeed, we obtain termination by a case distinction: either the graph is cyclic and existential rules will be blocked, or the graph is acyclic and existential rules will apply at most once for each vertex. It is open how this type of reasoning by cases can be integrated into practical termination criteria.

Also note that the rules of Figure 1 are neither in $\mathsf{CT}^{\mathsf{sk}}_{\forall}$ nor in $\mathsf{CT}^{\mathsf{std}}_{\forall\forall}$. The skolem chase produces an infinite `succ`-chain if an `edge`-cycle is reachable from `s`, and rule (17) cannot prevent the application of rule (9) in this chase. The standard chase does have the ability to block rule (9), but it fails to do so if its strategy is to apply rule (9) before rule (17).

Indeed, our rule (17) effectively acts as an "emergency break" for the chase. Similar devices were considered before. Grahne and Onet introduce so-called *denial constraints*, which are rules that, when applied, stop the chase immediately [23]. Gogacz and Marcinkowski observe that this is also achievable with regular rules in the style of (17), which they call *flooding rules* [22]. This approach is strategy-dependent since it requires that the "break" is triggered eagerly before creating further unnecessary elements.

Can we modify Figure 1 to work for the standard chase? Even with arbitrary strategies, we require fairness, so all rule matches must eventually be satisfied. This is not enough, however, since rule (17) has an unbounded number of matches that could all be satisfied

$$\rightarrow \exists x.\texttt{zero}(x) \tag{21}$$

$$\texttt{zero}(x) \rightarrow \texttt{dist}(\texttt{s}, x) \wedge \texttt{ins}(\texttt{s}, x, x) \wedge \texttt{done}(x) \tag{22}$$

$$\texttt{dist}(v_1, x_1) \wedge \texttt{edge}(v_1, v_2) \wedge \texttt{done}(x_1) \rightarrow \exists x_2.\texttt{ins}(v_2, x_1, x_2) \wedge \texttt{subset}(x_2, x_2) \tag{23}$$

$$\texttt{subset}(x_1, x_2) \wedge \texttt{ins}(v, x_0, x_1) \rightarrow \texttt{ins}(v, x_2, x_2) \wedge \texttt{subset}(x_0, x_2) \tag{24}$$

$$\texttt{subset}(x_1, x_2) \wedge \texttt{zero}(x_1) \rightarrow \texttt{ins}(\texttt{s}, x_2, x_2) \wedge \texttt{done}(x_2) \tag{25}$$

$$\texttt{dist}(v_1, x_1) \wedge \texttt{edge}(v_1, v_2) \wedge \texttt{ins}(v_2, x_1, x_2) \rightarrow \texttt{dist}(v_2, x_2) \wedge \texttt{succ}(x_1, x_2) \tag{26}$$

Figure 2 Rules for creating sets of vertices to represent simple paths.

too late. An all-strategies terminating version can be obtained by modifying (17) to require only a single satisfied match to halt all computation globally. To this end, we introduce two new unary relation names `block` and `real`, remove the rules (9) and (17), add new rules:

$$\rightarrow \exists x.\texttt{block}(x) \wedge \texttt{succ}(x, x) \tag{18}$$

$$\texttt{dist}(v, x) \wedge \texttt{block}(y) \rightarrow \exists x'.\texttt{succ}(x, x') \wedge \texttt{real}(x') \wedge \texttt{succ}(x', y) \tag{19}$$

$$\texttt{block}(x) \wedge \texttt{goal} \rightarrow \texttt{real}(x) \tag{20}$$

and replace every body atom of the form $\texttt{succ}(e, f)$ with a conjunction $\texttt{succ}(e, f) \wedge \texttt{real}(f)$ in any of the other rules.

Intuitively, `block` defines a single element that acts as a universal blocker for all potential applications of rule (19). However, rules are restricted to work with successors that are marked as `real`, which is initially not the case for the blocking element. The block becomes "real" and therefore effective when rule (20) is applied. In contrast to the earlier version, (20) only needs to be applied for a single match. By fairness, any cyclic graph will eventually lead to the derivation of `goal` and hence to the application of (20). Therefore, the modified rule set is in $\mathsf{CT}^{\mathsf{std}}_{\forall\forall}$, but there termination can take arbitrarily long depending on the strategy.

This result connects to recent observations of Gogacz et al., who show that fairness is irrelevant for standard chase termination if all rules have only one atom per rule head [34]. They noted, however, that this no longer holds if rules may have multiple head atoms. Our construction requires such larger heads in all existential rules that should be affected by the global blocker, i.e., rule (19) in our example.

We have therefore learned that the standard chase, too, is strictly more expressive than the skolem chase. We can strengthen this result to obtain an upper bound for the size of the chase, albeit not a polynomial one:

▶ **Theorem 13.** *There is a rule set $\Sigma \in \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$ and BCQ q that realise the DK query. Moreover, the standard chase on Σ is at most exponential in the size of the input database.*

It remains open whether this can be strengthened to obtain a polynomial runtime guarantee – we conjecture that the exponential increase in effort is unavoidable. Indeed, the rule set used to show Theorem 13 may result in a chase of exponential size, even when using the Datalog-first strategy.

Proof of Theorem 13. A bigger change in our original approach is now needed. Instead of constructing a unique `succ`-chain to measure distances, we build a tree-like `succ`-structure that grows one branch for every `s`-path, and which stops to grow when encountering cycles. In other words, every element of the `succ`-structure represents a simple path. To accomplish

$$\texttt{first}(v) \rightarrow \exists x.\texttt{start}(x,x,v) \wedge \texttt{end}(x) \tag{27}$$

$$\texttt{start}(x,u,v) \wedge \texttt{end}(u) \wedge \texttt{next}(v,v') \rightarrow \exists y_1,y_2.\texttt{start}(y_1,x,v') \wedge \texttt{succ}(y_1,y_2) \wedge \texttt{end}(y_2) \tag{28}$$

$$\texttt{start}(x,u,v) \wedge \texttt{succ}(u,u') \rightarrow \exists y.\texttt{left}(x,y) \wedge \texttt{start}(y,u',v) \tag{29}$$

$$\texttt{left}(x,y) \rightarrow \exists y'.\texttt{right}(x,y') \wedge \texttt{succ}(y,y') \tag{30}$$

$$\texttt{right}(x,y) \wedge \texttt{succ}(x,x') \rightarrow \exists y'.\texttt{left}(x',y') \wedge \texttt{succ}(y,y') \tag{31}$$

$$\texttt{end}(x) \wedge \texttt{right}(x,y) \rightarrow \texttt{end}(y) \tag{32}$$

$$\texttt{start}(x,u,v) \wedge \texttt{end}(u) \wedge \texttt{last}(v) \wedge \texttt{succ}(x,x') \rightarrow \texttt{chain}(x,x') \tag{33}$$

$$\texttt{chain}(x,x') \wedge \texttt{succ}(x',x'') \rightarrow \texttt{chain}(x',x'') \tag{34}$$

Figure 3 Rule set to generate a k-exponentially long chain for the proof of Theorem 14.

this, we associate each element with the set of all vertices that have previously been visited along this path. The rules (7)–(10) and (17) in Figure 1 are replaced by the rules in Figure 2. We use facts of the form $\texttt{ins}(v,x,y)$ to express that y is the result of inserting v into the set x, i.e., "$\{v\} \cup x = y$." In particular, $\texttt{ins}(v,x,x)$ can be read as "$v \in x$." Rules (21) and (22) initialise a zero element to encode $\{\texttt{s}\}$. Rule (23) creates a new vertex set when required for extending a path, where $\texttt{done}(x_1)$ asserts that set x_1 was fully initialised and $\texttt{subset}(x_1,x_2)$ states that x_2 contains all elements of x_1. Rules (24) and (25) propagate `subset` to farther ancestors while establishing that all previously added vertices are also in the newly created set. A set is only considered `done` when when reaching the zero element $\{\texttt{s}\}$ (25). Finally, rule (26) uses the vertex sets to measure distances. The resulting `succ` branches are used as a basis for arithmetic operations as before, using rules (11)–(16).

At any given intermediate stage of the chase, we can associate any element c with a set $[c]$ that contains all elements e for which there is a fact $\texttt{ins}(e,c,c)$. Clearly, the unique zero element c_0 with $\texttt{zero}(c_0)$ satisfies $[c_0] = \{\texttt{s}\}$. By an easy induction, one can show that elements c_n for which $\texttt{done}(c_n)$ was derived satisfy the following: there is a unique chain of facts $\texttt{ins}(e_1,c_0,c_1),\ldots,\texttt{ins}(e_n,c_{n-1},c_n)$ with $c_i \neq c_{i+1}$, and we have $[c_n] = \{\texttt{s}, e_1, \ldots e_n\}$ and $\texttt{subset}(c_i,c_n)$ for all $0 \leq i \leq n$. The existence of these facts implies that a match h of rule (23) is always satisfied if $h(v) \in [h(x_1)]$, i.e., the rule can only be used to create strictly larger sets. Since the size of any set of vertices is bounded by the size of the input, there are at most exponentially many such elements, even if the graph has cycles.

If the graph is acyclic, then the `succ` relation forms a tree structure that corresponds to an unravelling of the directed acyclic graph rooted in `s`. The distance-related checks work as in Theorem 12. If the graph is cyclic, then `succ` might also contain cycles, but is still finite. Rules (4)–(6) will detect the cycle and lead to acceptance, as required. ◀

7 Beyond Polynomial Time

After observing the superior expressive power of the standard and Datalog-first chase on polynomial time problems, we turn to the question of whether one can also express queries of higher complexity in rule sets that are guaranteed to terminate in these chases. The answer is a resounding *yes*:

▶ **Theorem 14.** *There is a rule set* $\Sigma \in \mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}$ *and a BCQ* q *that express a non-elementary Boolean query.*

Proof. We reduce from the following non-elementary decision problem:

Input: A Turing machine $\mathcal{M}$ and a number k Question: When started on the empty tape, does $\mathcal{M}$ halt in at most $\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{k \text{ times}}$ steps?

The number k is encoded by input facts $\texttt{first}(e_0), \texttt{next}(e_0, e_1), \dots, \texttt{next}(e_{k-1}, e_k), \texttt{last}(e_k)$. We use the chase to construct a chain of the required k-exponential length. A simulator for k-exponential Turing machine computations can then be implemented with Datalog rules using a standard construction [16]. Note that the query result on inputs that do not use the required encoding (of k or the Turing machine) is irrelevant for hardness; yet we must ensure that the chase terminates on such inputs.

We first describe the basic construction of the k-exponential chain and discuss termination later. The rules in Figure 3 produce a series of k full binary trees of depth $1, 2, 2^2, 2^{2^2}, \dots$ Each tree starts on the second level (containing two elements) and uses relation names `left` and `right` to define a node's children. Nodes on the same level form a chain $\texttt{succ}(n_1, n_2), \texttt{succ}(n_2, n_3), \dots, \texttt{succ}(n_{\ell-1}, n_\ell)$; the first element n_1 is marked by a fact $\texttt{start}(n_1, u, v)$ where u defines the level of the tree, and v defines the number of the tree; the last element n_ℓ is marked by $\texttt{end}(n_\ell)$.

Rule (27) creates a one-element "tree" as a start (not in the above list of k rootless trees). Rule (28) initialises the first level of the next tree, using the `succ`-chain of the last level of the previous tree to count up the levels. Subsequent levels of the tree are initialised by (29) and completed by rules (30) and (31). The next level's last element is marked by (32). Rules (33) and (34) define the last level of the last tree as the required chain.

It is not hard to see that the rules in Figure 3 terminate, even in the skolem chase, if the `next`-graph does not have cycles and the database only mentions the relation names `first`, `next`, and `last`. To ensure termination in case of `next`-cycles, we can add cycle-detection rules similar to (4)–(6). To ensure termination on all database instances (i.e., such that already contain facts using other relation names such as `succ` or `start`), we need to detect similar cyclic arrangements involving `succ` or `start`. Termination of the Datalog-first chase can then be ensured by adding "flooding rules" akin to (17), in this case creating ubiquitous `start`, `left`, `right`, and `succ` relations, so as to block rules (28)–(31). ◀

As discussed in Section 6, the flooding rules used to ensure termination of the Datalog-first chase can be replaced by introducing a global blocking element that can be activated in a single rule application when a cycle is detected. Fairness then also ensures the termination of the standard chase, and we obtain:

▶ **Theorem 15.** *There is a rule set* $\Sigma \in \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$ *and a BCQ* q *that express a non-elementary Boolean query.*

It is not difficult to define a fixed `next`-chain in rules, so as to use a constant tower of k exponentials instead of a data-dependent one. This yields rule sets that realise k-ExpTime-complete queries, which by the Time Hierarchy theorems cannot be realised in $(k-1)$-exponential time. This yields the following:

▶ **Theorem 16.** *The classes of rule sets in* $\mathsf{CT}^{\mathsf{std}}_{\forall\forall}$ *that terminate after at most* k*-exponentially many steps form a hierarchy of strictly increasing expressivity. The same applies to* k*-exponentially terminating rule sets in* $\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}$.

8 Conclusion

We have studied classes $\mathsf{CT}^{\mathsf{x}}_{\forall(\forall)}$ of existential rules for which a certain chase variant $\mathsf{x} \in \{\mathsf{sk}, \mathsf{std}, \mathsf{dlf}, \mathsf{core}\}$ terminates on all database instances, and (where applicable) under all strategies. To review our results, it is meaningful to further distinguish chase termination classes by upper bounds in terms of the size of the input database. For example, $\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}(\text{poly})$ would denote the subset of $\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}$ where chase termination is guaranteed after polynomial time. Now given such a class CT, we investigated the set $[\![\mathsf{CT}]\!]$ of all abstract queries (sets of concrete databases over an input signature) that can be expressed by some theory from CT. Using this notation, our main results are as follows:

$$\begin{array}{rcccccccccc}
[\![\mathsf{Datalog}]\!] & \overset{\text{(Thm 9)}}{=} & [\![\mathsf{CT}^{\mathsf{sk}}_{\forall}]\!] & \overset{\text{(Thm 12)}}{\subset} & [\![\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}(\text{poly})]\!] & \overset{\text{(Thm 16)}}{\subset} & [\![\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}(\text{exp})]\!] & \overset{\text{(Thm 16)}}{\subset} \ldots \subset & [\![\bigcup_k \mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}(k\text{-exp})]\!] & \overset{\text{(Thm 14)}}{\subset} & [\![\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}]\!] \\
 & & \overset{\text{(Thm 8)}}{\shortparallel} & & \cup\!| & & \cup\!| & & \cup\!| & & \cup\!| \\
[\![\mathsf{CT}^{\mathsf{sk}}_{\forall}(\text{poly})]\!] & \subseteq & [\![\mathsf{CT}^{\mathsf{std}}_{\forall\forall}(\text{poly})]\!] & \overset{\text{(Thm 16)}}{\subset} & [\![\mathsf{CT}^{\mathsf{std}}_{\forall\forall}(\text{exp})]\!] & \overset{\text{(Thm 16)}}{\subset} & \ldots \subset & [\![\bigcup_k \mathsf{CT}^{\mathsf{std}}_{\forall\forall}(k\text{-exp})]\!] & \overset{\text{(Thm 15)}}{\subset} & [\![\mathsf{CT}^{\mathsf{std}}_{\forall\forall}]\!]
\end{array}$$

Many further questions remain open, and indicate promising directions for future research:

Absolute expressibility. A terminating chase can only express queries that are decidable and closed under homomorphism, but we saw that the skolem chase can express much less. Some of the other chase variants might actually capture this class of queries. Even if not, it would be interesting to characterise their expressivity semantically.

Relative expressibility. We know that $[\![\mathsf{CT}^{\mathsf{std}}_{\forall\forall}]\!] \subseteq [\![\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}]\!] \subseteq [\![\mathsf{CT}^{\mathsf{core}}_{\forall}]\!]$, but it remains open if any of these inclusions are strict. If some are equalities, it would be interesting to find computable rewritings that produce rule sets for which a weaker chase terminates.

Complexity relationships. Theorem 13 achieved termination for the standard chase at the cost of an exponential runtime increase. We conjecture that this is unavoidable, and that $[\![\mathsf{CT}^{\mathsf{std}}_{\forall\forall}(\text{poly})]\!] \subset [\![\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}(\text{poly})]\!]$. Can all queries in $[\![\mathsf{CT}^{\mathsf{dlf}}_{\forall\forall}]\!]$ be implemented in worst-case optimal time? And can the standard chase express the same queries at an exponential penalty? Do we even have $[\![\mathsf{CT}^{\mathsf{sk}}_{\forall}]\!] \neq [\![\mathsf{CT}^{\mathsf{std}}_{\forall\forall}(\text{poly})]\!]$?

Decidable termination criteria. None of our beyond-skolem queries satisfy any of the known termination criteria. New approaches are needed to encompass our examples.

Termination on restricted database classes. We required termination on all databases, using databases with restricted EDB signatures only to define query realisation. Requiring termination only on databases over EDB relations might lead to larger classes of rule sets, possibly with higher expressivity. This certainly occurs when restricting to specific "well-formed" instance databases: if we would exclude cyclic databases, even the skolem chase could express non-elementary queries. However, such restrictions are also enough to capture all of PTime (assuming negation and order to be axiomatised), and even unrealistically powerful classes, such as non-uniform $\mathrm{P}_{/\mathrm{poly}}$.

Practical applications. Practical implementations for standard chase and also for Datalog-first chase exist, so it is promising to explore the use of beyond-skolem expressive power in applications. Specific uses could help guide the theoretical research.

References

1. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison Wesley, 1994.
2. Alfred V. Aho, Catriel Beeri, and Jeffrey D. Ullman. The Theory of Joins in Relational Databases. *ACM Trans. Database Syst.*, 4(3):297–314, 1979.
3. Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. On rules with existential variables: Walking the decidability line. *Artificial Intelligence*, 175(9–10):1620–1654, 2011.
4. Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, Swan Rocher, and Clément Sipieter. Graal: A Toolkit for Query Answering with Existential Rules. In Nick Bassiliades, Georg Gottlob, Fariba Sadri, Adrian Paschke, and Dumitru Roman, editors, *Proc. 9th Int. Web Rule Symposium (RuleML'15)*, volume 9202 of *LNCS*, pages 328–344. Springer, 2015.
5. Bruce L. Bauslaugh. Core-like properties of infinite graphs and structures. *Discrete Math.*, 138(1):101–111, 1995.
6. Catriel Beeri and Moshe Y. Vardi. The Implication Problem for Data Dependencies. In Shimon Even and Oded Kariv, editors, *Proc. 8th Colloquium on Automata, Languages and Programming (ICALP'81)*, volume 115 of *LNCS*, pages 73–85. Springer, 1981.
7. Catriel Beeri and Moshe Y. Vardi. A Proof Procedure for Data Dependencies. *J. ACM*, 31(4):718–741, 1984.
8. Michael Benedikt, George Konstantinidis, Giansalvatore Mecca, Boris Motik, Paolo Papotti, Donatello Santoro, and Efthymia Tsamoura. Benchmarking the Chase. In *Proc. 36th Symposium on Principles of Database Systems (PODS'17)*, pages 37–52. ACM, 2017.
9. Angela Bonifati, Ioana Ileana, and Michele Linardi. Functional Dependencies Unleashed for Scalable Data Exchange. In Peter Baumann, Ioana Manolescu-Goujot, Luca Trani, Yannis E. Ioannidis, Gergely Gábor Barnaföldi, László Dobos, and Evelin Bányai, editors, *Proc. 28th Int. Conf. on Scientific and Statistical Database Management (SSDBM'16)*, pages 2:1–2:12. ACM, 2016.
10. Marco Calautti and Andreas Pieris. Oblivious Chase Termination: The Sticky Case. In *Proc. 22nd Int. Conf. on Database Theory (ICDT'19)*. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2019.
11. Andrea Calì, Georg Gottlob, and Andreas Pieris. Query Answering under Non-guarded Rules in Datalog+/-. In Pascal Hitzler and Thomas Lukasiewicz, editors, *Proc. 4th Int. Conf. on Web Reasoning and Rule Systems (RR 2010)*, volume 6333 of *LNCS*, pages 1–17. Springer, 2010.
12. David Carral, Irina Dragoste, and Markus Krötzsch. Restricted Chase (Non)Termination for Existential Rules with Disjunctions. In Carles Sierra, editor, *Proc. 26th Int. Joint Conf. on Artificial Intelligence (IJCAI'17)*, pages 922–928. ijcai.org, 2017.
13. David Carral, Markus Krötzsch, Maximilian Marx, Ana Ozaki, and Sebastian Rudolph. Preserving Constraints with the Stable Chase. In Benny Kimelfeld and Yael Amsterdamer, editors, *Proc. 21st Int. Conf. on Database Theory (ICDT'18)*, volume 98 of *LIPIcs*, pages 12:1–12:19. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2018.
14. Stefano Ceri, Georg Gottlob, and Letizia Tanca. *Logic Programming and Databases*. Springer, 1990.
15. Bernardo Cuenca Grau, Ian Horrocks, Markus Krötzsch, Clemens Kupke, Despoina Magka, Boris Motik, and Zhe Wang. Acyclicity Notions for Existential Rules and Their Application to Query Answering in Ontologies. *J. of Artificial Intelligence Research*, 47:741–808, 2013.
16. Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys*, 33(3):374–425, 2001.
17. Anuj Dawar and Stephan Kreutzer. On Datalog vs. LFP. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Proc. 35th Int. Colloquium on Automata, Languages, and Programming (ICALP'08); Part II*, volume 5126 of *LNCS*, pages 160–171. Springer, 2008.

18 Stathis Delivorias, Michel Leclère, Marie-Laure Mugnier, and Federico Ulliana. On the k-Boundedness for Existential Rules. In Christoph Benzmüller, Francesco Ricca, Xavier Parent, and Dumitru Roman, editors, *Proc. 2nd Int. Joint Conf. on Rules and Reasoning (RuleML+RR'18)*, volume 11092 of *LNCS*, pages 48–64. Springer, 2018.

19 Alin Deutsch, Alan Nash, and Jeffrey B. Remmel. The Chase Revisited. In Maurizio Lenzerini and Domenico Lembo, editors, *Proc. 27th Symposium on Principles of Database Systems (PODS'08)*, pages 149–158. ACM, 2008.

20 Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89–124, 2005.

21 Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. That's All Folks! LLUNATIC Goes Open Source. *PVLDB*, 7(13):1565–1568, 2014.

22 Tomasz Gogacz and Jerzy Marcinkowski. All-Instances Termination of Chase is Undecidable. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Proc. 41st Int. Colloquium on Automata, Languages, and Programming (ICALP'14); Part II*, volume 8573 of *LNCS*, pages 293–304. Springer, 2014.

23 Gösta Grahne and Adrian Onet. Anatomy of the Chase. *Fundam. Inform.*, 157(3):221–270, 2018.

24 Pavol Hell and Jaroslav Nešetřil. The core of a graph. *Discrete Math.*, 109:117–126, 1992.

25 Markus Krötzsch and Sebastian Rudolph. Extending Decidable Existential Rules by Joining Acyclicity and Guardedness. In Toby Walsh, editor, *Proc. 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI'11)*, pages 963–968. AAAI Press/IJCAI, 2011.

26 David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. Testing implications of data dependencies. *ACM Transactions on Database Systems*, 4:455–469, 1979.

27 Bruno Marnette. Generalized Schema-Mappings: from Termination to Tractability. In Jan Paredaens and Jianwen Su, editors, *Proc. 28th Symposium on Principles of Database Systems (PODS'09)*, pages 13–22. ACM, 2009.

28 Michael Meier, Michael Schmidt, and Georg Lausen. On Chase Termination Beyond Stratification. *PVLDB*, 2(1):970–981, 2009.

29 Michaël Thomazo Michel Leclére, Marie-Laure Mugnier and Federico Ulliana. A Single Approach to Decide Chase Termination on Linear Existential Rules. *CoRR*, abs/1810.02132, 2018. `arXiv:1810.02132`.

30 Boris Motik, Yavor Nenov, Robert Piro, Ian Horrocks, and Dan Olteanu. Parallel Materialisation of Datalog Programs in Centralised, Main-Memory RDF Systems. In *Proc. 28th AAAI Conf. on Artif. Intell. (AAAI'14)*, pages 129–137. AAAI Press, 2014.

31 Sebastian Rudolph. The Two Views on Ontological Query Answering. In Georg Gottlob and Jorge Pérez, editors, *Proc. 8th Alberto Mendelzon Workshop on Foundations of Data Management (AMW'14)*, volume 1189 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2014.

32 Sebastian Rudolph and Michaël Thomazo. Characterization of the Expressivity of Existential Rule Queries. In Qiang Yang and Michael Wooldridge, editors, *Proc. 24th Int. Joint Conf. on Artificial Intelligence (IJCAI'15)*, pages 3193–3199. AAAI Press, 2015.

33 Sebastian Rudolph and Michaël Thomazo. Expressivity of Datalog Variants - Completing the Picture. In Subbarao Kambhampati, editor, *Proc. 25th Int. Joint Conf. on Artificial Intelligence (IJCAI'15)*, pages 1230–1236. AAAI Press, 2016.

34 Andreas Pieris Tomasz Gogacz, Jerzy Marcinkowski. All-Instances Restricted Chase Termination: The Guarded Case. *CoRR*, abs/1901.03897, 2019. `arXiv:1901.03897`.

35 Jacopo Urbani, Markus Krötzsch, Ceriel J. H. Jacobs, Irina Dragoste, and David Carral. Efficient Model Construction for Horn Logic with VLog: System description. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *Proc. 9th Int. Joint Conf. on Automated Reasoning (IJCAR'18)*, volume 10900 of *LNCS*, pages 680–688. Springer, 2018.

36 Heng Zhang, Yan Zhang, and Jia-Huai You. Existential Rule Languages with Finite Chase: Complexity and Expressiveness. In Blai Bonet and Sven Koenig, editors, *Proc. 29th AAAI Conf. on Artificial Intelligence (AAAI'15)*. AAAI Press, 2015.

Counting Triangles under Updates in Worst-Case Optimal Time

Ahmet Kara
Department of Computer Science, University of Oxford, Oxford, UK
ahmet.kara@cs.ox.ac.uk

Hung Q. Ngo
RelationalAI, Inc., Berkeley, CA, USA
hung.ngo@relational.ai

Milos Nikolic[1]
School of Informatics, University of Edinburgh, Edinburgh, UK
milos.nikolic@ed.ac.uk

Dan Olteanu
Department of Computer Science, University of Oxford, Oxford, UK
dan.olteanu@cs.ox.ac.uk

Haozhe Zhang
Department of Computer Science, University of Oxford, Oxford, UK
haozhe.zhang@cs.ox.ac.uk

Abstract

We consider the problem of incrementally maintaining the triangle count query under single-tuple updates to the input relations. We introduce an approach that exhibits a space-time tradeoff such that the space-time product is quadratic in the size of the input database and the update time can be as low as the square root of this size. This lowest update time is worst-case optimal conditioned on the Online Matrix-Vector Multiplication conjecture.

The classical and factorized incremental view maintenance approaches are recovered as special cases of our approach within the space-time tradeoff. In particular, they require linear-time maintenance under updates, which is suboptimal. Our approach can also count all triangles in a static database in the worst-case optimal time needed for enumerating them.

2012 ACM Subject Classification Theory of computation → Database query processing and optimization (theory); Information systems → Database views; Information systems → Data streams

Keywords and phrases incremental view maintenance, amortized analysis, data skew

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.4

Related Version An extended version of this work is available online at [13], `https://arxiv.org/abs/1804.02780`.

Funding This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 682588. Kara acknowledges funding from Fondation Wiener Anspach.

Acknowledgements Olteanu would like to thank Nicole Schweikardt for the connection between the Online Matrix-Vector Multiplication conjecture and the triangle query.

[1] Work performed while being at the University of Oxford.

1 Introduction

We consider the problem of incrementally maintaining the result of the triangle count query

$$Q() = \sum_{a \in \mathsf{Dom}(A)} \sum_{b \in \mathsf{Dom}(B)} \sum_{c \in \mathsf{Dom}(C)} R(a,b) \cdot S(b,c) \cdot T(c,a) \tag{1}$$

under single-tuple updates to the relations R, S, and T with schemas (A,B), (B,C), and (C,A), respectively. The relations are given as functions mapping tuples over relation schemas to tuple multiplicities. A single-tuple update $\delta R = \{ (\alpha,\beta) \mapsto m \}$ to relation R maps the tuple (α,β) to a nonzero multiplicity m, which is positive for inserts and negative for deletes.

The triangle query and its counting variant have served as a milestone for worst-case optimality of join algorithms in the centralized and parallel settings and for randomized approximation schemes for data processing. They serve as the workhorse showcasing suboptimality of mainstream join algorithms used currently by virtually all commercial database systems. For a database $\mathbf{D}$ consisting of R, S, and T, standard binary join plans implementing these queries may take $O(|\mathbf{D}|^2)$ time, yet these queries can be solved in $\mathcal{O}(|\mathbf{D}|^{\frac{3}{2}})$ time [2]. This observation motivated a new line of work on worst-case optimal algorithms for arbitrary join queries [18]. The triangle query has also served as a yardstick for understanding the optimal communication cost for parallel query evaluation in the Massively Parallel Communication model [15]. The triangle count query has witnessed the development of randomized approximation schemes with increasingly lower time and space requirements, e.g., [9].

A worst-case optimal result for incrementally maintaining the exact triangle count query has so far not been established. Incremental maintenance algorithms may benefit from a good range of processing techniques whose flexible combinations may make it harder to reason about optimality. Such techniques include algorithms for aggregate-join queries with low complexity developed for the non-incremental case [17]; pre-materialization of views that reduces maintenance of the query to that of simpler subqueries [14]; and delta processing that allows to only compute the change in the result instead of the entire result [7].

1.1 Existing Incremental View Maintenance (IVM) Approaches

The problem of incrementally maintaining the triangle count has received a fair amount of attention. Existing exact approaches require at least linear time in worst case. After each update to a database $\mathbf{D}$, the naïve approach joins the relations R, S, and T in time $\mathcal{O}(|\mathbf{D}|^{\frac{3}{2}})$ using a worst-case optimal algorithm [2, 18] and counts the result tuples. The number of distinct tuples in the result is at most $|\mathbf{D}|^{\frac{3}{2}}$, which is a well-known result by Loomis and Whitney from 1949 (see recent notes on the history of this result [17]). The classical first-order IVM [7] computes on the fly a delta query δQ per single-tuple update δR to relation R (or any other relation) and updates the query result:

$$\delta Q() = \delta R(\alpha,\beta) \cdot \sum_{c \in \mathsf{Dom}(C)} S(\beta,c) \cdot T(c,\alpha), \qquad Q() = Q() + \delta Q().$$

The delta computation takes $\mathcal{O}(|\mathbf{D}|)$ time since it needs to intersect two lists of possibly linearly many C-values that are paired with β in S and with α in T (i.e., the multiplicity of such pairs in S and T is nonzero). The recursive IVM [14] speeds up the delta computation by precomputing three auxiliary views representing the update-independent parts of the delta queries for updates to R, S, and T:

$$V_{ST}(b,a) = \sum_{c \in \mathsf{Dom}(C)} S(b,c) \cdot T(c,a)$$
$$V_{TR}(c,b) = \sum_{a \in \mathsf{Dom}(A)} T(c,a) \cdot R(a,b)$$
$$V_{RS}(a,c) = \sum_{b \in \mathsf{Dom}(B)} R(a,b) \cdot S(b,c).$$

These three views take $\mathcal{O}(|\mathbf{D}|^2)$ space but allow to compute the delta query for single-tuple updates to the input relations in $\mathcal{O}(1)$ time. Computing the delta $\delta Q() = \delta R(\alpha,\beta) \cdot V_{ST}(\beta,\alpha)$ requires just a constant-time lookup in V_{ST}; however, maintaining the views V_{RS} and V_{TR}, which refer to R, still requires $\mathcal{O}(|\mathbf{D}|)$ time. The factorized IVM [19] materializes only one of the three views, for instance, V_{ST}. In this case, the maintenance under updates to R takes $\mathcal{O}(1)$ time, but the maintenance under updates to S and T still takes $\mathcal{O}(|\mathbf{D}|)$ time.

Further exact IVM approaches focus on acyclic conjunctive queries. For free-connex acyclic conjunctive queries, the dynamic Yannakakis approach allows for enumeration of result tuples with constant delay under single-tuple updates [11]. For databases with or without integrity constraints, it is known that a strict, small subset of the class of acyclic conjunctive queries admit constant-time update, while all other conjunctive queries have update times dependent on the size of the input database [4, 5].

Further away from our line of work is the development of dynamic descriptive complexity, starting with the DynFO complexity class and the much-acclaimed result on FO expressibility of the maintenance for graph reachability under edge inserts and deletes, cf. a recent survey [20]. The k-clique query can be maintained under edge inserts by a quantifier-free update program of arity $k-1$ but not of arity $k-2$ [22].

A distinct line of work investigates randomized approximation schemes with an arbitrary relative error for counting triangles in a graph given as a stream of edges, e.g., [3, 12, 6, 16, 8]. Each edge in the data stream corresponds to a tuple insert, and tuple deletes are not considered. The emphasis of these approaches is on space efficiency, and they express the space utilization as a function of the number of nodes and edges in the input graph and of the number of triangles. The space utilization is generally sublinear but may become superlinear if, for instance, the number of edges is greater than the square root of the number of triangles. The update time is polylogarithmic in the number of nodes in the graph.

A complementary line of work unveils structure in the PTIME complexity class by giving lower bounds on the complexity of problems under various conjectures [10, 21].

▶ **Definition 1** (Online Matrix-Vector Multiplication (OMv) [10]). *We are given an $n \times n$ Boolean matrix $\mathbf{M}$ and receive n column vectors of size n, denoted by $\mathbf{v}_1, \ldots, \mathbf{v}_n$, one by one; after seeing each vector $\mathbf{v}_i$, we output the product $\mathbf{M}\mathbf{v}_i$ before we see the next vector.*

▶ **Conjecture 2** (OMv Conjecture, Theorem 2.4 in [10]). *For any $\gamma > 0$, there is no algorithm that solves OMv in time $\mathcal{O}(n^{3-\gamma})$.*

The OMv conjecture has been used to exhibit conditional lower bounds for many dynamic problems, including those previously based on other popular problems and conjectures, such as 3SUM and combinatorial Boolean matrix multiplication [10]. This also applies to our triangle count query: For any $\gamma > 0$ and database of domain size n, there is no algorithm that incrementally maintains the triangle count under single-tuple updates with arbitrary preprocessing time, $\mathcal{O}(n^{1-\gamma})$ update time, and $\mathcal{O}(n^{2-\gamma})$ answer time, unless the OMv conjecture fails [4].

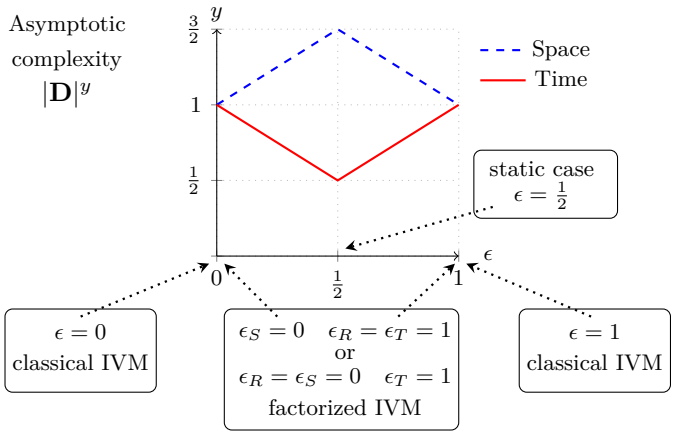


Figure 1 IVM$^\epsilon$'s space and amortized update time parameterized by ϵ. The classical IVM is recovered by setting $\epsilon \in \{0, 1\}$. The factorized IVM is recovered by setting $\epsilon_R \in \{0, 1\}$, $\epsilon_S = 0$, and $\epsilon_T = 1$ when V_{ST} is materialized (similar treatment when V_{RS} or V_{TR} is materialized). For $\epsilon = \frac{1}{2}$, IVM$^\epsilon$ counts all triangles in a static database in the worst-case optimal time for enumerating them.

1.2 Our Contribution

This paper introduces IVM$^\epsilon$, an incremental view maintenance approach that maintains the triangle count in amortized sublinear time. Our main result is as follows:

▶ **Theorem 3.** *Given a database* $\mathbf{D}$ *and* $\epsilon \in [0, 1]$*, IVM*$^\epsilon$ *incrementally maintains the result of Query* (1) *under single-tuple updates to* $\mathbf{D}$ *with* $\mathcal{O}(|\mathbf{D}|^{\frac{3}{2}})$ *preprocessing time,* $\mathcal{O}(|\mathbf{D}|^{\max\{\epsilon, 1-\epsilon\}})$ *amortized update time, constant answer time, and* $\mathcal{O}(|\mathbf{D}|^{1+\min\{\epsilon, 1-\epsilon\}})$ *space.*

The preprocessing time is for computing the triangle count on the initial database before the updates; if we start with the empty database, then this time is $\mathcal{O}(1)$. The IVM$^\epsilon$ approach exhibits a tradeoff between space and amortized update time, cf. Figure 1.

IVM$^\epsilon$ uses a data structure that partitions each input relation into a heavy part and a light part based on the degrees of data values. The degree of an A-value a in relation R is the number of B-values paired with a in R. The light part of R consists of all tuples (a, b) from R such that the degree of a in R is below a certain threshold that depends on the database size and ϵ. All other tuples are included in the heavy part of R. Similarly, the relations S and T are partitioned based on the degrees of B-values in S and C-values in T, respectively. The maintenance is adaptive in that it uses different evaluation strategies for different heavy-light combinations of parts of the input relations that overall keep the update time sublinear. Section 3 introduces this adaptive maintenance strategy.

As the database evolves under updates, IVM$^\epsilon$ needs to rebalance the heavy-light partitions to account for a new database size and updated degrees of data values. While this rebalancing may take superlinear time, it remains sublinear per single-tuple update. The update time is therefore amortized. Section 4 discusses the rebalancing strategy of IVM$^\epsilon$.

For $\epsilon = \frac{1}{2}$, IVM$^\epsilon$ achieves the lowest update time $\mathcal{O}(|\mathbf{D}|^{\frac{1}{2}})$ while requiring $\mathcal{O}(|\mathbf{D}|^{\frac{3}{2}})$ space. This update time is optimal conditioned on the OMv conjecture. For this, we specialize the lower bound result in [4] to refer to the size $|\mathbf{D}|$ of the database:

▶ **Proposition 4.** *For any* $\gamma > 0$ *and database* $\mathbf{D}$*, there is no algorithm that incrementally maintains the result of Query* (1) *under single-tuple updates to* $\mathbf{D}$ *with arbitrary preprocessing time,* $\mathcal{O}(|\mathbf{D}|^{\frac{1}{2}-\gamma})$ *amortized update time, and* $\mathcal{O}(|\mathbf{D}|^{1-\gamma})$ *answer time, unless the OMv conjecture fails.*

This lower bound is shown in the extended paper [13]. Theorem 3 and Proposition 4 imply that IVM$^\epsilon$ incrementally maintains the triangle count with optimal update time:

▶ **Corollary 5** (Theorem 3 and Proposition 4). *Given a database* $\mathbf{D}$*, IVM*$^\epsilon$ *incrementally maintains the result of Query* (1) *under single-tuple updates to* $\mathbf{D}$ *with worst-case optimal amortized update time* $\mathcal{O}(|\mathbf{D}|^{\frac{1}{2}})$ *and constant answer time, unless the* OMv *conjecture fails.*

IVM$^\epsilon$ also applies to triangle count queries with self-joins, such as when maintaining the count of triangles in a graph given by the edge relation. The space and time complexities are the same as in Theorem 3 [13].

IVM$^\epsilon$ defines a continuum of maintenance approaches that exhibit a space-time tradeoff based on ϵ. As depicted in Figure 1, the classical first-order IVM and the factorized IVM are specific extreme points in this continuum. To recover the former, we set $\epsilon \in \{0, 1\}$ for $\mathcal{O}(|\mathbf{D}|)$ update time and $\mathcal{O}(|\mathbf{D}|)$ space for the input relations. To recover the latter, we use a distinct parameter ϵ per relation: for example, using $\epsilon_R \in \{0, 1\}$, $\epsilon_S = 0$, and $\epsilon_T = 1$, we support updates to R in $\mathcal{O}(1)$ time and updates to S and T in $\mathcal{O}(|\mathbf{D}|)$ time; the view V_{ST} takes $\mathcal{O}(|\mathbf{D}|^2)$ space [13].

We observe that at optimality, IVM$^\epsilon$ recovers the worst-case optimal time $\mathcal{O}(|\mathbf{D}|^{\frac{3}{2}})$ of non-incremental algorithms for enumerating all triangles [18]. Whereas these algorithms are monolithic and require processing the input data in bulk and all joins at the same time, IVM$^\epsilon$ achieves the same complexity by inserting $|\mathbf{D}|$ tuples one at a time in initially empty relations R, S, and T, and by using standard join plans [13].

2 Preliminaries

Data Model. A schema $\mathbf{X}$ is a tuple of variables. Each variable X has a discrete domain $\mathsf{Dom}(X)$ of data values. A tuple $\mathbf{x}$ of data values over schema $\mathbf{X}$ is an element from $\mathsf{Dom}(\mathbf{X}) = \prod_{X \in \mathbf{X}} \mathsf{Dom}(X)$. We use uppercase letters for variables and lowercase letters for data values. Likewise, we use bold uppercase letters for schemas and bold lowercase letters for tuples of data values.

A relation K over schema $\mathbf{X}$ is a function $K : \mathsf{Dom}(\mathbf{X}) \to \mathbb{Z}$ mapping tuples over $\mathbf{X}$ to integers such that $K(\mathbf{x}) \neq 0$ for finitely many tuples $\mathbf{x}$. We say that a tuple $\mathbf{x}$ is in K, denoted by $\mathbf{x} \in K$, if $K(\mathbf{x}) \neq 0$. The value $K(\mathbf{x})$ represents the multiplicity of $\mathbf{x}$ in K. The size $|K|$ of K is the size of the set $\{\mathbf{x} \mid \mathbf{x} \in K\}$. A database $\mathbf{D}$ is a set of relations, and its size $|\mathbf{D}|$ is the sum of the sizes of the relations in $\mathbf{D}$.

Given a tuple $\mathbf{x}$ over schema $\mathbf{X}$ and a variable X in $\mathbf{X}$, we write $\mathbf{x}[X]$ to denote the value of X in $\mathbf{x}$. For a relation K over $\mathbf{X}$, a variable X in $\mathbf{X}$, and a data value $x \in \mathsf{Dom}(X)$, we use $\sigma_{X=x}K$ to denote the set of tuples in K whose X-value is x, that is, $\sigma_{X=x}K = \{\mathbf{x} \mid \mathbf{x} \in K \wedge \mathbf{x}[X] = x\}$. We write $\pi_X K$ to denote the set of X-values in K, that is, $\pi_X K = \{\mathbf{x}[X] \mid \mathbf{x} \in K\}$.

Query Language. We express queries and view definitions in the language of functional aggregate queries (FAQ) [1]. Compared to the original FAQ definition that uses several commutative semirings, we define our queries using the single commutative ring $(\mathbb{Z}, +, \cdot, 0, 1)$ of integers with the usual addition and multiplication. A query Q has one of the two forms:

1. Given a set $\{X_i\}_{i\in[n]}$ of variables and an index set $S \subseteq [n]$, let $\mathbf{X}_S$ denote a tuple $(X_i)_{i\in S}$ of variables and $\mathbf{x}_S$ denote a tuple of data values over the schema $\mathbf{X}_S$. Then,

$$Q(\mathbf{x}_{[f]}) = \sum_{x_{f+1} \in \mathsf{Dom}(X_{f+1})} \cdots \sum_{x_n \in \mathsf{Dom}(X_n)} \prod_{S \in \mathcal{M}} K_S(\mathbf{x}_S), \text{ where:}$$

- $\mathcal{M}$ is a multiset of index sets.
- For every index set $S \in \mathcal{M}$, $K_S : \mathsf{Dom}(\mathbf{X}_S) \to \mathbb{Z}$ is a relation over the schema $\mathbf{X}_S$.
- $\mathbf{X}_{[f]}$ is the tuple of free variables of Q. The variables $X_{f+1}, \ldots, X_n$ are called bound.

2. $Q(\mathbf{x}) = Q_1(\mathbf{x}) + Q_2(\mathbf{x})$, where Q_1 and Q_2 are queries over the same tuple of free variables.

In the following, we use $\sum_{x_i}$ as a shorthand for $\sum_{x_i \in \mathsf{Dom}(X_i)}$.

Updates and Delta Queries. An update δK to a relation K is a relation over the schema of K. A single-tuple update, written as $\delta K = \{\mathbf{x} \mapsto m\}$, maps the tuple $\mathbf{x}$ to the nonzero multiplicity $m \in \mathbb{Z}$ and any other tuple to 0; that is, $|\delta K| = 1$. The data model and query language make no distinction between inserts and deletes – these are updates represented as relations in which tuples have positive and negative multiplicities.

Given a query Q and an update δK, the delta query δQ defines the change in the query result after applying δK to the database. The rules for deriving delta queries follow from the associativity, commutativity, and distributivity of the ring operations.

Query $Q(\mathbf{x})$	Delta query $\delta Q(\mathbf{x})$
$Q_1(\mathbf{x}_1) \cdot Q_2(\mathbf{x}_2)$	$\delta Q_1(\mathbf{x}_1) \cdot Q_2(\mathbf{x}_2) + Q_1(\mathbf{x}_1) \cdot \delta Q_2(\mathbf{x}_2) + \delta Q_1(\mathbf{x}_1) \cdot \delta Q_2(\mathbf{x}_2)$
$\sum_x Q_1(\mathbf{x}_1)$	$\sum_x \delta Q_1(\mathbf{x}_1)$
$Q_1(\mathbf{x}) + Q_2(\mathbf{x})$	$\delta Q_1(\mathbf{x}) + \delta Q_2(\mathbf{x})$
$K'(\mathbf{x})$	$\delta K(\mathbf{x})$ when $K = K'$ and 0 otherwise

Computation Time. Our maintenance algorithm takes as input the triangle count query Q and a database $\mathbf{D}$ and maintains the result of Q under a sequence of single-tuple updates. We distinguish the following computation times: (1) *preprocessing time* is spent on initializing the algorithm using $\mathbf{D}$ before any update is received, (2) *update time* is spent on processing one single-tuple update, and (3) *answer time* is spent on obtaining the result of Q. We consider two types of bounds on the update time: *worst-case bounds*, which limit the time each individual update takes in the worst case, and *amortized worst-case bounds*, which limit the average worst-case time taken by a sequence of updates. Enumerating a set of tuples with constant delay means that the time until reporting the first tuple, the time between reporting two consecutive tuples, and the time between reporting the last tuple and the end of enumeration is constant. When referring to sublinear time, we mean $\mathcal{O}(|\mathbf{D}|^{1-\gamma})$ for some $\gamma > 0$, where $|\mathbf{D}|$ is the database size.

Computational Model. We consider the RAM model of computation. Each relation (view) K over schema $\mathbf{X}$ is implemented by a data structure that stores key-value entries $(\mathbf{x}, K(\mathbf{x}))$ for each tuple $\mathbf{x}$ over $\mathbf{X}$ with $K(\mathbf{x}) \neq 0$ and needs space linear in the number of such tuples. We assume that this data structure supports (1) looking up, inserting, and deleting entries in constant time, (2) enumerating all stored entries in K with constant delay, and (3) returning $|K|$ in constant time. For instance, a hash table with chaining, where entries are doubly linked for efficient enumeration, can support these operations in constant time on average, under the assumption of simple uniform hashing.

For each variable X in the schema $\mathbf{X}$ of relation K, we further assume there is an index structure on X that allows: (4) enumerating all entries in K matching $\sigma_{X=x}K$ with constant delay, (5) checking $x \in \pi_X K$ in constant time, and (6) returning $|\sigma_{X=x}K|$ in constant time, for any $x \in \mathsf{Dom}(X)$, and (7) inserting and deleting index entries in constant time. Such an index structure can be realized, for instance, as a hash table with chaining where each key-value entry stores an X-value x and a doubly-linked list of pointers to the entries in K

having the X-value x. Looking up an index entry given x takes constant time on average, and its doubly-linked list enables enumeration of the matching entries in K with constant delay. Inserting an index entry into the hash table additionally prepends a new pointer to the doubly-linked list for a given x; overall, this operation takes constant time on average. For efficient deletion of index entries, each entry in K also stores back-pointers to its index entries (as many back-pointers as there are index structures for K). When an entry is deleted from K, locating and deleting its index entries takes constant time per index.

Data Partitioning. We partition each input relation into two parts based on the degrees of its values. Similar to common techniques used in databases to deal with data skew, our IVM approach employs different maintenance strategies for values of high and low frequency.

▶ **Definition 6** (Relation Partition). *Given a relation K over schema $\mathbf{X}$, a variable X from the schema $\mathbf{X}$, and a threshold θ, a partition of K on X with threshold θ is a set $\{K_h, K_l\}$ satisfying the following conditions:*

$$\begin{aligned}
&\textit{(union)} && K(\mathbf{x}) = K_h(\mathbf{x}) + K_l(\mathbf{x}) \textit{ for } \mathbf{x} \in \mathsf{Dom}(\mathbf{X}) \\
&\textit{(domain partition)} && (\pi_X K_h) \cap (\pi_X K_l) = \emptyset \\
&\textit{(heavy part)} && \textit{for all } x \in \pi_X K_h : \ |\sigma_{X=x} K_h| \geq \tfrac{1}{2}\,\theta \\
&\textit{(light part)} && \textit{for all } x \in \pi_X K_l : \ |\sigma_{X=x} K_l| < \tfrac{3}{2}\,\theta
\end{aligned}$$

The set $\{K_h, K_l\}$ is called a strict partition of K on X with threshold θ if it satisfies the union and domain partition conditions and the following strict versions of the heavy part and light part conditions:

$$\begin{aligned}
&\textit{(strict heavy part)} && \textit{for all } x \in \pi_X K_h : \ |\sigma_{X=x} K_h| \geq \theta \\
&\textit{(strict light part)} && \textit{for all } x \in \pi_X K_l : \ |\sigma_{X=x} K_l| < \theta
\end{aligned}$$

The relations K_h and K_l are called the heavy and light parts of K.

Definition 6 admits multiple ways to (non-strictly) partition a relation K on variable X with threshold θ. For instance, assume that $|\sigma_{X=x} K| = \theta$ for some X-value x in K. Then, all tuples in K with X-value x can be in either the heavy or light part of K; but they cannot be in both parts because of the domain partition condition. If the partition is strict, then all such tuples are in the heavy part of K.

The strict partition of a relation K is unique for a given threshold and can be computed in time linear in the size of K.

3 IVM$^\epsilon$: Adaptive Maintenance of the Triangle Count

We present IVM$^\epsilon$, our algorithm for the incremental maintenance of the result of Query (1). We start with a high-level overview. Consider a database $\mathbf{D}$ consisting of three relations R, S, and T with schemas (A, B), (B, C), and (C, A), respectively. We partition R, S, and T on variables A, B, and C, respectively, for a given threshold. We then decompose Query (1) into eight skew-aware views expressed over these relation parts:

$$Q_{rst}() = \sum_{a,b,c} R_r(a, b) \cdot S_s(b, c) \cdot T_t(c, a), \qquad \text{for } r, s, t \in \{h, l\}.$$

Query (1) is then the sum of these skew-aware views: $Q() = \sum_{r,s,t \in \{h,l\}} Q_{rst}()$.

IVM$^\epsilon$ adapts its maintenance strategy to each skew-aware view Q_{rst} to ensure amortized update time that is sublinear in the database size. While most of these views admit sublinear

Materialized View Definition	Space Complexity
$Q() = \sum_{r,s,t \in \{h,l\}} \sum_{a,b,c} R_r(a,b) \cdot S_s(b,c) \cdot T_t(c,a)$	$\mathcal{O}(1)$
$V_{RS}(a,c) = \sum_b R_h(a,b) \cdot S_l(b,c)$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{1+\min\{\epsilon,1-\epsilon\}})$
$V_{ST}(b,a) = \sum_c S_h(b,c) \cdot T_l(c,a)$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{1+\min\{\epsilon,1-\epsilon\}})$
$V_{TR}(c,b) = \sum_a T_h(c,a) \cdot R_l(a,b)$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{1+\min\{\epsilon,1-\epsilon\}})$

Figure 2 The definition and space complexity of the materialized views in $\mathbf{V} = \{Q, V_{RS}, V_{ST}, V_{TR}\}$ as part of an IVM$^\epsilon$ state of a database $\mathbf{D}$ partitioned for $\epsilon \in [0,1]$.

delta computation over the relation parts, few exceptions require linear-time maintenance. For these exceptions, IVM$^\epsilon$ precomputes the update-independent parts of the delta queries as *auxiliary materialized views* and then exploits these views to speed up the delta evaluation.

One such exception is the view Q_{hhl}. Consider a single-tuple update $\delta R_h = \{(\alpha,\beta) \mapsto m\}$ to the heavy part R_h of relation R, where α and β are fixed data values. Computing the delta view $\delta Q_{hhl}() = \delta R_h(\alpha,\beta) \cdot \sum_c S_h(\beta,c) \cdot T_l(c,\alpha)$ requires iterating over all the C-values c paired with β in S_h and with α in T_l; the number of such C-values can be linear in the size of the database. To avoid this iteration, IVM$^\epsilon$ precomputes the view $V_{ST}(b,a) = \sum_c S_h(b,c) \cdot T_l(c,a)$ and uses this view to evaluate $\delta Q_{hhl}() = \delta R_h(\alpha,\beta) \cdot V_{ST}(\beta,\alpha)$ in constant time.

Such auxiliary views, however, also require maintenance. All such views created by IVM$^\epsilon$ can be maintained in sublinear time under single-tuple updates to the input relations. Figure 2 summarizes these views used by IVM$^\epsilon$ to maintain Query (1): V_{RS}, V_{ST} and V_{TR}. They serve to avoid linear-time delta computation for updates to T, R, and S, respectively. IVM$^\epsilon$ also materializes the result of Query (1), which ensures constant answer time.

We now describe our strategy in detail. We start by defining the state that IVM$^\epsilon$ initially creates and maintains upon each update. Then, we specify the procedure for processing a single-tuple update to any input relation, followed by the space complexity analysis of IVM$^\epsilon$. Section 4 gives the procedure for processing a sequence of such updates.

▶ **Definition 7** (IVM$^\epsilon$ State). *Given a database $\mathbf{D} = \{R,S,T\}$ and $\epsilon \in [0,1]$, an IVM$^\epsilon$ state of $\mathbf{D}$ is a tuple $\mathcal{Z} = (\epsilon, N, \mathbf{P}, \mathbf{V})$, where:*

- *N is a natural number such that the size invariant $\lfloor \frac{1}{4}N \rfloor \leq |\mathbf{D}| < N$ holds. N is called the threshold base.*
- *$\mathbf{P} = \{R_h, R_l, S_h, S_l, T_h, T_l\}$ consists of the partitions of R, S, and T on variables A, B, and C, respectively, with threshold $\theta = N^\epsilon$.*
- *$\mathbf{V}$ is the set of materialized views $\{Q, V_{RS}, V_{ST}, V_{TR}\}$ as defined in Figure 2.*

The initial state $\mathcal{Z}$ of $\mathbf{D}$ has $N = 2 \cdot |\mathbf{D}| + 1$ and the three partitions in $\mathbf{P}$ are strict.

By construction, $|\mathbf{P}| = |\mathbf{D}|$. The size invariant implies $|\mathbf{D}| = \Theta(N)$ and, together with the heavy and light part conditions, facilitates the amortized analysis of IVM$^\epsilon$ in Section 4. Definition 6 provides two essential upper bounds for each relation partition in an IVM$^\epsilon$ state: The number of distinct A-values in R_h is at most $\frac{N}{\frac{1}{2}N^\epsilon} = 2N^{1-\epsilon}$, i.e., $|\pi_A R_h| \leq 2N^{1-\epsilon}$, and the number of tuples in R_l with an A-value a is less than $\frac{3}{2}N^\epsilon$, i.e., $|\sigma_{A=a} R_l| < \frac{3}{2}N^\epsilon$, for any $a \in \mathsf{Dom}(A)$. The same bounds hold for B-values in $\{S_h, S_l\}$ and C-values in $\{T_h, T_l\}$.

3.1 Preprocessing Stage

The preprocessing stage constructs the initial IVM$^\epsilon$ state given a database $\mathbf{D}$ and $\epsilon \in [0,1]$.

▶ **Proposition 8.** *Given a database* $\mathbf{D}$ *and* $\epsilon \in [0,1]$*, constructing the initial IVM$^\epsilon$ state of* $\mathbf{D}$ *takes* $\mathcal{O}(|\mathbf{D}|^{\frac{3}{2}})$ *time.*

Proof. We analyze the time to construct the initial state $\mathcal{Z} = (\epsilon, N, \mathbf{P}, \mathbf{V})$ of $\mathbf{D}$. Retrieving the size $|\mathbf{D}|$ and computing $N = 2 \cdot |\mathbf{D}| + 1$ take constant time. Strictly partitioning the input relations from $\mathbf{D}$ using the threshold N^ϵ, as described in Definition 6, takes $\mathcal{O}(|\mathbf{D}|)$ time. Computing the result of the triangle count query on $\mathbf{D}$ (or $\mathbf{P}$) using a worst-case optimal join algorithm [18] takes $\mathcal{O}(|\mathbf{D}|^{\frac{3}{2}})$ time. Computing the auxiliary views V_{RS}, V_{ST}, and V_{TR} takes $\mathcal{O}(|\mathbf{D}|^{1+\min\{\epsilon,1-\epsilon\}})$ time, as shown next. Consider the view $V_{RS}(a,c) = \sum_b R_h(a,b) \cdot S_l(b,c)$. To compute V_{RS}, one can iterate over all (a,b) pairs in R_h and then find the C-values in S_l for each b. The light part S_l contains at most N^ϵ distinct C-values for any B-value, which gives an upper bound of $|R_h| \cdot N^\epsilon$ on the size of V_{RS}. Alternatively, one can iterate over all (b,c) pairs in S_l and then find the A-values in R_h for each b. The heavy part R_h contains at most $N^{1-\epsilon}$ distinct A-values, which gives an upper bound of $|S_l| \cdot N^{1-\epsilon}$ on the size of V_{RS}. The number of steps needed to compute this result is upper-bounded by $\min\{\, |R_h| \cdot N^\epsilon,\, |S_l| \cdot N^{1-\epsilon} \,\} < \min\{\, N \cdot N^\epsilon,\, N \cdot N^{1-\epsilon} \,\} = N^{1+\min\{\epsilon,1-\epsilon\}}$. From $|\mathbf{D}| = \Theta(N)$ follows that computing V_{RS} on the database partition $\mathbf{P}$ takes $\mathcal{O}(|\mathbf{D}|^{1+\min\{\epsilon,1-\epsilon\}})$ time; the analysis for V_{ST} and V_{TR} is analogous. Note that $\max_{\epsilon\in[0,1]}\{1+\min\{\epsilon,1-\epsilon\}\} = \frac{3}{2}$. Overall, the initial state $\mathcal{Z}$ of $\mathbf{D}$ can be constructed in $\mathcal{O}(|\mathbf{D}|^{\frac{3}{2}})$ time. ◀

The preprocessing stage of IVM$^\epsilon$ happens *before* any update is received. In case we start from an empty database, the preprocessing cost of IVM$^\epsilon$ is $\mathcal{O}(1)$.

3.2 Processing a Single-Tuple Update

We describe the IVM$^\epsilon$ strategy for maintaining the result of Query (1) under a single-tuple update to the relation R. This update can affect either the heavy or light part of R, hence we write δR_r, where r stands for h or l. We assume that checking whether the update affects the heavy or light part of R takes constant time. The update is represented as a relation $\delta R_r = \{\, (\alpha,\beta) \mapsto m \,\}$, where α and β are data values and $m \in \mathbb{Z}$. Due to the symmetry of the triangle query and auxiliary views, updates to S and T are handled similarly.

Figure 3 shows the procedure ApplyUpdate that takes as input a current IVM$^\epsilon$ state $\mathcal{Z}$ and the update δR_r, and returns a new state that results from applying δR_r to $\mathcal{Z}$. The procedure computes the deltas of the skew-aware views referencing R_r, which are δQ_{rhh} (Line 3), δQ_{rhl} (Line 4), δQ_{rlh} (Line 5), and δQ_{rll} (Line 6), and uses these deltas to maintain the triangle count (Line 7). These skew-aware views are not materialized, but their deltas facilitate the maintenance of the triangle count. If the update affects the heavy part R_h of R, the procedure maintains V_{RS} (Line 9) and R_h (Line 12); otherwise, it maintains V_{TR} (Line 11) and R_l (Line 12). The view V_{ST} remains unchanged as it has no reference to R_h or R_l.

Figure 3 also gives the time complexity of computing these deltas and applying them to $\mathcal{Z}$. This complexity is either constant or dependent on the number of C-values for which matching tuples in the parts of S and T have nonzero multiplicities.

▶ **Proposition 9.** *Given a state* $\mathcal{Z}$ *constructed from a database* $\mathbf{D}$ *for* $\epsilon \in [0,1]$*, IVM$^\epsilon$ maintains* $\mathcal{Z}$ *under a single-tuple update to any input relation in* $\mathcal{O}(|\mathbf{D}|^{\max\{\epsilon,1-\epsilon\}})$ *time.*

ApplyUpdate($\delta R_r, \mathcal{Z}$)		Time
1	**let** $\delta R_r = \{(\alpha, \beta) \mapsto m\}$	
2	**let** $\mathcal{Z} = (\epsilon, N, \{R_h, R_l, S_h, S_l, T_h, T_l\}, \{Q, V_{RS}, V_{ST}, V_{TR}\})$	
3	$\delta Q_{rhh}() = \delta R_r(\alpha, \beta) \cdot \sum_c S_h(\beta, c) \cdot T_h(c, \alpha)$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{1-\epsilon})$
4	$\delta Q_{rhl}() = \delta R_r(\alpha, \beta) \cdot V_{ST}(\beta, \alpha)$	$\mathcal{O}(1)$
5	$\delta Q_{rlh}() = \delta R_r(\alpha, \beta) \cdot \sum_c S_l(\beta, c) \cdot T_h(c, \alpha)$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{\min\{\epsilon, 1-\epsilon\}})$
6	$\delta Q_{rll}() = \delta R_r(\alpha, \beta) \cdot \sum_c S_l(\beta, c) \cdot T_l(c, \alpha)$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{\epsilon})$
7	$Q() = Q() + \delta Q_{rhh}() + \delta Q_{rhl}() + \delta Q_{rlh}() + \delta Q_{rll}()$	$\mathcal{O}(1)$
8	**if** (r is h)	
9	$\quad V_{RS}(\alpha, c) = V_{RS}(\alpha, c) + \delta R_h(\alpha, \beta) \cdot S_l(\beta, c)$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{\epsilon})$
10	**else**	
11	$\quad V_{TR}(c, \beta) = V_{TR}(c, \beta) + T_h(c, \alpha) \cdot \delta R_l(\alpha, \beta)$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{1-\epsilon})$
12	$R_r(\alpha, \beta) = R_r(\alpha, \beta) + \delta R_r(\alpha, \beta)$	$\mathcal{O}(1)$
13	**return** $\mathcal{Z}$	
	Total update time:	$\mathcal{O}(\lvert\mathbf{D}\rvert^{\max\{\epsilon, 1-\epsilon\}})$

Figure 3 (left) Counting triangles under a single-tuple update. ApplyUpdate takes as input an update δR_r to the heavy or light part of R, hence $r \in \{h, l\}$, and the current IVM$^\epsilon$ state $\mathcal{Z}$ of a database $\mathbf{D}$ partitioned using $\epsilon \in [0, 1]$. It returns a new state that results from applying δR_r to $\mathcal{Z}$. Lines 3-6 compute the deltas of the affected skew-aware views, and Line 7 maintains Q. Lines 9 and 11 maintain the auxiliary views V_{RS} and V_{TR}, respectively. Line 12 maintains the affected part R_r. (right) The time complexity of computing and applying deltas. The evaluation strategy for computing δQ_{rlh} in Line 5 may choose either S_l or T_h to bound C-values, depending on ϵ. The total time is the maximum of all individual times. The maintenance procedures for S and T are similar.

Proof. We analyze the running time of the procedure from Figure 3 given a single-tuple update $\delta R_r = \{(\alpha, \beta) \mapsto m\}$ and a state $\mathcal{Z} = (\epsilon, N, \mathbf{P}, \mathbf{V})$ of $\mathbf{D}$. Since the query and auxiliary views are symmetric, the analysis for updates to S and T is similar.

We first analyze the evaluation strategies for the deltas of the skew-aware views Q_{rst}:

- (Line 3) Computing δQ_{rhh} requires summing over C-values (α and β are fixed). The minimum degree of each C-value in T_h is $\frac{1}{2}N^\epsilon$, which means the number of distinct C-values in T_h is at most $\frac{N}{\frac{1}{2}N^\epsilon} = 2N^{1-\epsilon}$. Thus, this delta evaluation takes $\mathcal{O}(N^{1-\epsilon})$ time.
- (Line 4) Computing δQ_{rhl} requires constant-time lookups in δR_r and V_{ST}.
- (Line 5) Computing δQ_{rlh} can be done in two ways, depending on ϵ: either sum over at most $2N^{1-\epsilon}$ C-values in T_h for the given α or sum over at most $\frac{3}{2}N^\epsilon$ C-values in S_l for the given β. This delta computation takes at most $\min\{2N^{1-\epsilon}, \frac{3}{2}N^\epsilon\}$ constant-time operations, thus $\mathcal{O}(N^{\min\{\epsilon, 1-\epsilon\}})$ time.
- (Line 6) Computing δQ_{rll} requires summing over at most $\frac{3}{2}N^\epsilon$ C-values in S_l for the given β. This delta computation takes $\mathcal{O}(N^\epsilon)$ time.

Maintaining the result of Query (1) using these deltas takes constant time (Line 7). The views V_{RS} and V_{TR} are maintained for updates to distinct parts of R. Maintaining V_{RS} requires iterating over at most $\frac{3}{2}N^\epsilon$ C-values in S_l for the given β (Line 9); similarly, maintaining V_{TR} requires iterating over at most $2N^{1-\epsilon}$ C-values in T_h for the given α (Line 11). Finally,

maintaining the (heavy or light) part of R affected by δR_r takes constant time (Line 12). The total update time is $\mathcal{O}(\max\{1, N^\epsilon, N^{1-\epsilon}, N^{\min\{\epsilon,1-\epsilon\}}\}) = \mathcal{O}(N^{\max\{\epsilon,1-\epsilon\}})$. From the invariant $|\mathbf{D}| = \Theta(N)$ follows the claimed time complexity $\mathcal{O}(|\mathbf{D}|^{\max\{\epsilon,1-\epsilon\}})$. ◀

3.3 Space Complexity

We next analyze the space complexity of the IVM$^\epsilon$ maintenance strategy.

▶ **Proposition 10.** *Given a database* $\mathbf{D}$ *and* $\epsilon \in [0,1]$*, the IVM*$^\epsilon$ *state constructed from* $\mathbf{D}$ *to support the maintenance of the result of Query* (1) *takes* $\mathcal{O}(|\mathbf{D}|^{1+\min\{\epsilon,1-\epsilon\}})$ *space.*

Proof. We consider a state $\mathcal{Z} = (\epsilon, N, \mathbf{P}, \mathbf{V})$ of database $\mathbf{D}$. N and ϵ take constant space and $|\mathbf{P}| = |\mathbf{D}|$. Figure 2 summarizes the space complexity of the materialized views Q, V_{RS}, V_{ST}, and V_{TR} from $\mathbf{V}$. The result of Q takes constant space. As discussed in the proof of Proposition 8, to compute the auxiliary view $V_{RS}(a,c) = \sum_b R_h(a,b) \cdot S_l(b,c)$, we can use either R_h or S_l as the outer relation:

$$|V_{RS}| \le \min\{\, |R_h| \cdot \max_{b \in \pi_B S_l} |\sigma_{B=b} S_l|,\ |S_l| \cdot \max_{b \in \pi_B R_h} |\sigma_{B=b} R_h| \,\} < \min\{\, N \cdot \frac{3}{2} N^\epsilon, N \cdot 2N^{1-\epsilon} \,\}$$

The size of V_{RS} is thus $\mathcal{O}(N^{1+\min\{\epsilon,1-\epsilon\}})$. From $|\mathbf{D}| = \Theta(N)$ follows that V_{RS} takes $\mathcal{O}(|\mathbf{D}|^{1+\min\{\epsilon,1-\epsilon\}})$ space; the space analysis for V_{ST} and V_{TR} is analogous. Overall, the state $\mathcal{Z}$ of $\mathbf{D}$ takes $\mathcal{O}(|\mathbf{D}|^{1+\min\{\epsilon,1-\epsilon\}})$ space. ◀

4 Rebalancing Partitions

The partition of a relation may change after updates. For instance, an insert $\delta R_l = \{(\alpha, \beta) \mapsto 1\}$ may violate the size invariant $\lfloor \frac{1}{4} N \rfloor \le |\mathbf{D}| < N$ or may violate the light part condition $|\sigma_{A=\alpha} R_l| < \frac{3}{2} N^\epsilon$ and require moving all tuples with the A-value α from R_l to R_h. As the database evolves under updates, IVM$^\epsilon$ performs *major* and *minor* rebalancing steps to ensure the size invariant and the conditions for heavy and light parts of each partition always hold. This rebalancing also ensures that the upper bounds on the number of data values, such as the number of B-values paired with α in R_l and the number of distinct A-values in R_h, are valid. The rebalancing cost is amortized over multiple updates.

Major Rebalancing

If an update causes the database size to fall below $\lfloor \frac{1}{4} N \rfloor$ or reach N, IVM$^\epsilon$ halves or, respectively, doubles N, followed by strictly repartitioning the database with the new threshold N^ϵ and recomputing the materialized views, as shown in Figure 4.

▶ **Proposition 11.** *Given* $\epsilon \in [0,1]$*, major rebalancing of an IVM*$^\epsilon$ *state constructed from a database* $\mathbf{D}$ *takes* $\mathcal{O}(|\mathbf{D}|^{1+\min\{\epsilon,1-\epsilon\}})$ *time.*

Proof. We consider the major rebalancing procedure from Figure 4. Strictly partitioning the input relations takes $\mathcal{O}(|\mathbf{D}|)$ time. From the proof of Proposition 8 and $|\mathbf{D}| = \Theta(N)$ follow that recomputing V_{RS}, V_{ST}, and V_{TR} takes $\mathcal{O}(|\mathbf{D}|^{1+\min\{\epsilon,1-\epsilon\}})$ time. ◀

The (super)linear time of major rebalancing is amortized over $\Omega(N)$ updates. After a major rebalancing step, it holds that $|\mathbf{D}| = \frac{1}{2} N$ (after doubling), or $|\mathbf{D}| = \frac{1}{2} N - \frac{1}{2}$ or $|\mathbf{D}| = \frac{1}{2} N - 1$ (after halving, i.e., setting N to $\lfloor \frac{1}{2} N \rfloor - 1$; the two options are due to the floor functions in the size invariant and halving expression). To violate the size invariant $\lfloor \frac{1}{4} N \rfloor \le |\mathbf{D}| < N$ and trigger another major rebalancing, the number of required updates is at least $\frac{1}{4} N$. Section 4.1 proves the amortized $\mathcal{O}(|\mathbf{D}|^{\min\{\epsilon,1-\epsilon\}})$ time of major rebalancing.

OnUpdate($\delta R, \mathcal{Z}$)

let $\delta R = \{(\alpha, \beta) \mapsto m\}$
let $\mathcal{Z} = (\epsilon, N, \{R_h, R_l\} \cup \mathbf{P}, \mathbf{V})$
if ($\alpha \in \pi_A R_h$ **or** $\epsilon = 0$)
  $\mathcal{Z} =$ ApplyUpdate($\delta R_h = \{(\alpha, \beta) \mapsto m\}, \mathcal{Z}$)
else
  $\mathcal{Z} =$ ApplyUpdate($\delta R_l = \{(\alpha, \beta) \mapsto m\}, \mathcal{Z}$)
if ($|\mathbf{D}| = N$)
  $N = 2N$
  $\mathcal{Z} =$ MajorRebalancing($\mathcal{Z}$)
else if ($|\mathbf{D}| < \lfloor \frac{1}{4} N \rfloor$)
  $N = \lfloor \frac{1}{2} N \rfloor - 1$
  $\mathcal{Z} =$ MajorRebalancing($\mathcal{Z}$)
else if ($\alpha \in \pi_A R_l$ **and** $|\sigma_{A=\alpha} R_l| \geq \frac{3}{2} N^\epsilon$)
  $\mathcal{Z} =$ MinorRebalancing($R_l, R_h, A, \alpha, \mathcal{Z}$)
else if ($\alpha \in \pi_A R_h$ **and** $|\sigma_{A=\alpha} R_h| < \frac{1}{2} N^\epsilon$)
  $\mathcal{Z} =$ MinorRebalancing($R_h, R_l, A, \alpha, \mathcal{Z}$)
return $\mathcal{Z}$

MajorRebalancing($\mathcal{Z}$)

let $\mathcal{Z} = (\epsilon, N, \{R_h, R_l, S_h, S_l, T_h, T_l\},$
  $\{Q, V_{RS}, V_{ST}, V_{TR}\})$
$\{R_h, R_l\} =$ StrictPartition(R_h, R_l, A, N^ϵ)
$\{S_h, S_l\} =$ StrictPartition(S_h, S_l, B, N^ϵ)
$\{T_h, T_l\} =$ StrictPartition(T_h, T_l, C, N^ϵ)
$V_{RS}(a, c) = \sum_b R_h(a, b) \cdot S_l(b, c)$
$V_{ST}(b, a) = \sum_c S_h(b, c) \cdot T_l(c, a)$
$V_{TR}(c, b) = \sum_a T_h(c, a) \cdot R_l(a, b)$
return $\mathcal{Z}$

MinorRebalancing($K_{src}, K_{dst}, X, x, \mathcal{Z}$)

foreach $\mathbf{t} \in \sigma_{X=x} K_{src}$ **do**
  $m = K_{src}(\mathbf{t})$
  $\mathcal{Z} =$ ApplyUpdate($\delta K_{src} = \{\, \mathbf{t} \mapsto -m \,\}, \mathcal{Z}$)
  $\mathcal{Z} =$ ApplyUpdate($\delta K_{dst} = \{\, \mathbf{t} \mapsto m \,\}, \mathcal{Z}$)
return $\mathcal{Z}$

Figure 4 Counting triangles under a single-tuple update with rebalancing. OnUpdate takes as input an update δR and the current IVM$^\epsilon$ state $\mathcal{Z}$ of a database $\mathbf{D}$. It returns a new state that results from applying δR to $\mathcal{Z}$ and, if necessary, rebalancing partitions. The condition $\epsilon = 0$ in the third line ensures that all tuples are in R_h when $\epsilon = 0$. ApplyUpdate is given in Figure 3. MinorRebalancing($K_{src}, K_{dst}, X, x, \mathcal{Z}$) moves all tuples with the X-value x from K_{src} to K_{dst}. MajorRebalancing($\mathcal{Z}$) recomputes the relation partitions and views in $\mathcal{Z}$. StrictPartition(K_h, K_l, X, θ) constructs a strict partition of relation K on variable X with threshold θ (see Definition 6). The OnUpdate procedures for updates to relations S and T are analogous.

Minor Rebalancing

After each update $\delta R = \{(\alpha, \beta) \mapsto m\}$, IVM$^\epsilon$ checks whether the two conditions $|\sigma_{A=\alpha} R_h| \geq \frac{1}{2} N^\epsilon$ and $|\sigma_{A=\alpha} R_l| < \frac{3}{2} N^\epsilon$ still hold. If the first condition is violated, all tuples in R_h with the A-value α are moved to R_l and the affected views are updated; similarly, if the second condition is violated, all tuples with the A-value α are moved from R_l to R_h, followed by updating the affected views. Figure 4 shows the procedure for minor rebalancing, which deletes affected tuples from one part and inserts them into the other part.

▶ **Proposition 12.** *Given $\epsilon \in [0, 1]$, minor rebalancing of an IVM$^\epsilon$ state constructed from a database $\mathbf{D}$ takes $\mathcal{O}(|\mathbf{D}|^{\epsilon + \max\{\epsilon, 1-\epsilon\}})$ time.*

Proof. Consider a state $\mathcal{Z} = (\epsilon, N, \mathbf{P}, \mathbf{V})$. Minor rebalancing moves fewer than $\frac{1}{2} N^\epsilon$ tuples (from heavy to light) or fewer than $\frac{3}{2} N^\epsilon + 1$ tuples (from light to heavy). Each tuple move performs one delete and one insert and costs $\mathcal{O}(|\mathbf{D}|^{\max\{\epsilon, 1-\epsilon\}})$ by Proposition 9. Since there are $\mathcal{O}(N^\epsilon)$ such operations and $|\mathbf{D}| = \Theta(N)$, the total time is $\mathcal{O}(|\mathbf{D}|^{\epsilon + \max\{\epsilon, 1-\epsilon\}})$. ◀

The (super)linear time of minor rebalancing is amortized over $\Omega(N^\epsilon)$ updates. This lower bound on the number of updates comes from the heavy and light part conditions (cf. Definition 6), namely from the gap between the two thresholds in these conditions. Section 4.1 proves the amortized $\mathcal{O}(|\mathbf{D}|^{\max\{\epsilon, 1-\epsilon\}})$ time of minor rebalancing.

Figure 4 gives the trigger procedure OnUpdate that maintains Query (1) under a single-tuple update to relation R and, if necessary, rebalances partitions; the procedures for updates to S and T are analogous. Given an update $\delta R = \{(\alpha, \beta) \mapsto m\}$ and an IVM$^\epsilon$ state of a database $\mathbf{D}$, the procedure first checks in constant time whether the update affects the heavy or light part of R. The update targets R_h if there exists a tuple with the same A-value α already in R_h, or ϵ is set to 0; otherwise, the update targets R_l. When $\epsilon = 0$, all tuples are in R_h, while R_l remains empty. Although this behavior is not required by IVM$^\epsilon$ (without the $\epsilon = 0$ condition, R_l would contain only tuples whose A-values have the degree of 1, and R_h would contain all other tuples), it allows us to recover existing IVM approaches, such as classical IVM and factorized IVM, which do not partition relations; by setting ϵ to 0 or 1, IVM$^\epsilon$ ensures that all tuples are in R_h or respectively R_l. The procedure OnUpdate then invokes ApplyUpdate from Figure 3. If the update causes a violation of the size invariant $\lfloor \frac{1}{4}N \rfloor \leq |\mathbf{D}| < N$, the procedure invokes MajorRebalancing to recompute the relation partitions and auxiliary views (note that major rebalancing has no effect on the triangle count). Otherwise, if the heavy or light part condition is violated, MinorRebalancing moves all tuples with the given A-value α from the source part to the destination part of R.

4.1 Proof of Theorem 3

We are now ready to prove Theorem 3 that states the complexity of IVM$^\epsilon$.

Proof. The preprocessing stage constructs the initial IVM$^\epsilon$ state from a database $\mathbf{D}$ in $\mathcal{O}(|\mathbf{D}|^{\frac{3}{2}})$ time, as shown in Proposition 8. Materializing the query result ensures constant answer time. The space complexity $\mathcal{O}(|\mathbf{D}|^{1+\min\{\epsilon,1-\epsilon\}})$ follows from Proposition 10.

We next analyze the amortized update time complexity. Let $\mathcal{Z}_0 = (\epsilon, N_0, \mathbf{P}_0, \mathbf{V}_0)$ be the initial IVM$^\epsilon$ state of a database $\mathbf{D}_0$ and $u_0, u_1, \ldots, u_{n-1}$ a sequence of arbitrary single-tuple updates. The application of this update sequence to $\mathcal{Z}_0$ yields a sequence $\mathcal{Z}_0 \xrightarrow{u_0} \mathcal{Z}_1 \xrightarrow{u_1} \ldots \xrightarrow{u_{n-1}} \mathcal{Z}_n$ of IVM$^\epsilon$ states, where $\mathcal{Z}_{i+1}$ is the result of executing the procedure OnUpdate$(u_i, \mathcal{Z}_i)$ from Figure 4, for $0 \leq i < n$. Let c_i denote the actual execution cost of OnUpdate$(u_i, \mathcal{Z}_i)$. For some $\Gamma > 0$, we can decompose each c_i as:

$$c_i = c_i^{apply} + c_i^{major} + c_i^{minor} + \Gamma, \qquad \text{for } 0 \leq i < n,$$

where c_i^{apply}, c_i^{major}, and c_i^{minor} are the actual costs of the subprocedures ApplyUpdate, MajorRebalancing, and MinorRebalancing, respectively, in OnUpdate. If update u_i causes no major rebalancing, then $c_i^{major} = 0$; similarly, if u_i causes no minor rebalancing, then $c_i^{minor} = 0$. These actual costs admit the following worst-case upper bounds:

$$\begin{aligned}
c_i^{apply} &\leq \gamma N_i^{\max\{\epsilon,1-\epsilon\}} && \text{(by Proposition 9),}\\
c_i^{major} &\leq \gamma N_i^{1+\min\{\epsilon,1-\epsilon\}} && \text{(by Proposition 11), and}\\
c_i^{minor} &\leq \gamma N_i^{\epsilon+\max\{\epsilon,1-\epsilon\}} && \text{(by Proposition 12),}
\end{aligned}$$

where γ is a constant derived from their asymptotic bounds, and N_i is the threshold base of $\mathcal{Z}_i$. The actual costs of major and minor rebalancing can be superlinear in the database size.

The crux of this proof is to show that assigning a *sublinear amortized cost* $\hat{c}_i$ to each update u_i accumulates enough budget to pay for such expensive but less frequent rebalancing procedures. For any sequence of n updates, our goal is to show that the accumulated amortized cost is no smaller than the accumulated actual cost:

$$\sum_{i=0}^{n-1} \hat{c}_i \geq \sum_{i=0}^{n-1} c_i. \tag{2}$$

The amortized cost assigned to an update u_i is $\hat{c}_i = \hat{c}_i^{apply} + \hat{c}_i^{major} + \hat{c}_i^{minor} + \Gamma$, where

$$\hat{c}_i^{apply} = \gamma N_i^{\max\{\epsilon,1-\epsilon\}}, \quad \hat{c}_i^{major} = 4\gamma N_i^{\min\{\epsilon,1-\epsilon\}}, \quad \hat{c}_i^{minor} = 2\gamma N_i^{\max\{\epsilon,1-\epsilon\}}, \quad \text{and}$$

Γ and γ are the constants used to upper bound the actual cost of OnUpdate. In contrast to the actual costs c_i^{major} and c_i^{minor}, the amortized costs $\hat{c}_i^{major}$ and $\hat{c}_i^{minor}$ are always nonzero.

We prove that such amortized costs satisfy Inequality (2). Since $\hat{c}_i^{apply} \geq c_i^{apply}$ for $0 \leq i < n$, it suffices to show that the following inequalities hold:

$$\textit{(amortizing major rebalancing)} \quad \sum_{i=0}^{n-1} \hat{c}_i^{major} \geq \sum_{i=0}^{n-1} c_i^{major} \qquad \text{and} \tag{3}$$

$$\textit{(amortizing minor rebalancing)} \quad \sum_{i=0}^{n-1} \hat{c}_i^{minor} \geq \sum_{i=0}^{n-1} c_i^{minor}. \tag{4}$$

We prove Inequalities (3) and (4) by induction on the length n of the update sequence.

Major rebalancing.

- *Base case*: We show that Inequality (3) holds for $n = 1$. The preprocessing stage sets $N_0 = 2 \cdot |\mathbf{D}_0| + 1$. If the initial database $\mathbf{D}_0$ is empty, then $N_0 = 1$ and u_0 triggers major rebalancing (and no minor rebalancing). The amortized cost $\hat{c}_0^{major} = 4\gamma N_0^{\min\{\epsilon,1-\epsilon\}} = 4\gamma$ suffices to cover the actual cost $c_0^{major} \leq \gamma N_0^{1+\min\{\epsilon,1-\epsilon\}} = \gamma$. If the initial database is nonempty, u_0 cannot trigger major rebalancing (i.e., violate the size invariant) because $\lfloor \frac{1}{4} N_0 \rfloor = \lfloor \frac{1}{2} |\mathbf{D}_0| \rfloor \leq |\mathbf{D}_0| - 1$ (lower threshold) and $|\mathbf{D}_0| + 1 < N_0 = 2 \cdot |\mathbf{D}_0| + 1$ (upper threshold); then, $\hat{c}_0^{major} \geq c_0^{major} = 0$. Thus, Inequality (3) holds for $n = 1$.
- *Inductive step*: Assumed that Inequality (3) holds for all update sequences of length up to $n - 1$, we show it holds for update sequences of length n. If update u_{n-1} causes no major rebalancing, then $\hat{c}_{n-1}^{major} = 4\gamma N_{n-1}^{\min\{\epsilon,1-\epsilon\}} \geq 0$ and $c_{n-1}^{major} = 0$, thus Inequality (3) holds for n. Otherwise, if applying u_{n-1} violates the size invariant, the database size $|\mathbf{D}_n|$ is either $\lfloor \frac{1}{4} N_{n-1} \rfloor - 1$ or N_{n-1}. Let $\mathcal{Z}_j$ be the state created after the previous major rebalancing or, if there is no such step, the initial state. For the former ($j > 0$), the major rebalancing step ensures $|\mathbf{D}_j| = \frac{1}{2} N_j$ after doubling and $|\mathbf{D}_j| = \frac{1}{2} N_j - \frac{1}{2}$ or $|\mathbf{D}_j| = \frac{1}{2} N_j - 1$ after halving the threshold base N_j; for the latter ($j = 0$), the preprocessing stage ensures $|\mathbf{D}_j| = \frac{1}{2} N_j - \frac{1}{2}$. The threshold base N_j changes only with major rebalancing, thus $N_j = N_{j+1} = \ldots = N_{n-1}$. The number of updates needed to change the database size from $|\mathbf{D}_j|$ to $|\mathbf{D}_n|$ (i.e., between two major rebalancing) is at least $\frac{1}{4} N_{n-1}$ since $\min\{\frac{1}{2} N_j - 1 - (\lfloor \frac{1}{4} N_{n-1} \rfloor - 1), N_{n-1} - \frac{1}{2} N_j\} \geq \frac{1}{4} N_{n-1}$. Then,

$$\begin{aligned}
\sum_{i=0}^{n-1} \hat{c}_i^{major} &\geq \sum_{i=0}^{j-1} c_i^{major} + \sum_{i=j}^{n-1} \hat{c}_i^{major} && \textit{(by induction hypothesis)} \\
&= \sum_{i=0}^{j-1} c_i^{major} + \sum_{i=j}^{n-1} 4\gamma N_{n-1}^{\min\{\epsilon,1-\epsilon\}} && \textit{($N_j = \ldots = N_{n-1}$)} \\
&\geq \sum_{i=0}^{j-1} c_i^{major} + \frac{1}{4} N_{n-1}\, 4\gamma N_{n-1}^{\min\{\epsilon,1-\epsilon\}} && \textit{(at least } \frac{1}{4} N_{n-1} \textit{ updates)} \\
&= \sum_{i=0}^{j-1} c_i^{major} + \gamma N_{n-1}^{1+\min\{\epsilon,1-\epsilon\}} \\
&\geq \sum_{i=0}^{j-1} c_i^{major} + c_{n-1}^{major} = \sum_{i=0}^{n-1} c_i^{major} && \textit{($c_j^{major} = \ldots = c_{n-2}^{major} = 0$).}
\end{aligned}$$

Thus, Inequality (3) holds for update sequences of length n.

Minor rebalancing. When the degree of a value in a partition changes such that the heavy or light part condition no longer holds, minor rebalancing moves the affected tuples between the heavy and light parts of the partition. To prove Inequality (4), we decompose the cost of minor rebalancing per relation and data value of its partitioning variable.

$$c_i^{minor} = \sum_{a \in \mathsf{Dom}(A)} c_i^{R,a} + \sum_{b \in \mathsf{Dom}(B)} c_i^{S,b} + \sum_{c \in \mathsf{Dom}(C)} c_i^{T,c} \quad \text{and}$$

$$\hat{c}_i^{minor} = \sum_{a \in \mathsf{Dom}(A)} \hat{c}_i^{R,a} + \sum_{b \in \mathsf{Dom}(B)} \hat{c}_i^{S,b} + \sum_{c \in \mathsf{Dom}(C)} \hat{c}_i^{T,c}$$

We write $c_i^{R,a}$ and $\hat{c}_i^{R,a}$ to denote the actual and respectively amortized costs of minor rebalancing caused by update u_i, for relation R and an A-value a. If update u_i is of the form $\delta R = \{(\alpha, \beta) \mapsto m\}$ and causes minor rebalancing, then $c_i^{R,\alpha} = c_i^{minor}$; otherwise, $c_i^{R,\alpha} = 0$. If update u_i is of the form $\delta R = \{(\alpha, \beta) \mapsto m\}$, then $\hat{c}_i^{R,\alpha} = \hat{c}_i^{minor}$ regardless of whether u_i causes minor rebalancing or not; otherwise, $\hat{c}_i^{R,\alpha} = 0$. The actual costs $c_i^{S,b}$ and $c_i^{T,c}$ and the amortized costs $\hat{c}_i^{S,b}$ and $\hat{c}_i^{T,c}$ are defined similarly.

We prove that for the partition of R and any $\alpha \in \mathsf{Dom}(A)$ the following inequality holds:

$$\sum_{i=0}^{n-1} \hat{c}_i^{R,\alpha} \geq \sum_{i=0}^{n-1} c_i^{R,\alpha}. \tag{5}$$

Due to the symmetry of the triangle query, Inequality (4) follows directly from Inequality (5).

We prove Inequality (5) for an arbitrary $\alpha \in \mathsf{Dom}(A)$ by induction on the length n of the update sequence.

- *Base case*: We show that Inequality (5) holds for $n = 1$. Assume that update u_0 is of the form $\delta R = \{(\alpha, \beta) \mapsto m\}$; otherwise, $\hat{c}_0^{R,\alpha} = c_0^{R,\alpha} = 0$, and Inequality (5) follows trivially for $n = 1$. If the initial database is empty, u_0 triggers major rebalancing but no minor rebalancing, thus $\hat{c}_0^{R,\alpha} = 2\gamma N_0^{\max\{\epsilon, 1-\epsilon\}} \geq c_0^{R,\alpha} = 0$. If the initial database is nonempty, each relation is partitioned using the threshold N_0^ϵ. For update u_0 to trigger minor rebalancing, the degree of the A-value α in R_h or R_l has to either decrease from $\lceil N_0^\epsilon \rceil$ to $\lceil \frac{1}{2} N_0^\epsilon \rceil - 1$ (heavy to light) or increase from $\lceil N_0^\epsilon \rceil - 1$ to $\lceil \frac{3}{2} N_0^\epsilon \rceil$ (light to heavy). The former happens only if $\lceil N_0^\epsilon \rceil = 1$ and update u_0 removes the last tuple with the A-value α from R_h, thus no minor rebalancing is needed; the latter cannot happen since update u_0 can increase $|\sigma_{A=\alpha} R_l|$ to at most $\lceil N_0^\epsilon \rceil$, and $\lceil N_0^\epsilon \rceil < \lceil \frac{3}{2} N_0^\epsilon \rceil$. In any case, $\hat{c}_0^{R,\alpha} \geq c_0^{R,\alpha}$, which implies that Inequality (5) holds for $n = 1$.
- *Inductive step*: Assumed that Inequality (5) holds for all update sequences of length up to $n - 1$, we show it holds for update sequences of length n. Consider that update u_{n-1} is of the form $\delta R = \{(\alpha, \beta) \mapsto m\}$ and causes minor rebalancing; otherwise, $\hat{c}_{n-1}^{R,\alpha} \geq 0$ and $c_{n-1}^{R,\alpha} = 0$, and Inequality (5) follows trivially for n. Let $\mathcal{Z}_j$ be the state created after the previous major rebalancing or, if there is no such step, the initial state. The threshold changes only with major rebalancing, thus $N_j = N_{j+1} = \ldots = N_{n-1}$. Depending on whether there exist minor rebalancing steps since state $\mathcal{Z}_j$, we distinguish two cases:

  Case 1: There is no minor rebalancing caused by an update of the form $\delta R = \{(\alpha, \beta') \mapsto m'\}$ since state $\mathcal{Z}_j$; thus, $c_j^{R,\alpha} = \ldots = c_{n-2}^{R,\alpha} = 0$. From state $\mathcal{Z}_j$ to state $\mathcal{Z}_n$, the number of tuples with the A-value α either decreases from at least $\lceil N_j^\epsilon \rceil$ to $\lceil \frac{1}{2} N_{n-1}^\epsilon \rceil - 1$ (heavy to light) or increases from at most $\lceil N_j^\epsilon \rceil - 1$ to $\lceil \frac{3}{2} N_{n-1}^\epsilon \rceil$ (light to heavy). For this change to happen, the number of updates needs to be greater than $\frac{1}{2} N_{n-1}^\epsilon$ since $N_j = N_{n-1}$ and $\min\{\lceil N_j^\epsilon \rceil - (\lceil \frac{1}{2} N_{n-1}^\epsilon \rceil - 1), \lceil \frac{3}{2} N_{n-1}^\epsilon \rceil - (\lceil N_j^\epsilon \rceil - 1)\} > \frac{1}{2} N_{n-1}^\epsilon$. Then,

$$\begin{aligned}
\sum_{i=0}^{n-1} \hat{c}_i^{R,\alpha} &\geq \sum_{i=0}^{j-1} c_i^{R,\alpha} + \sum_{i=j}^{n-1} \hat{c}_i^{R,\alpha} && \textit{(by induction hypothesis)} \\
&= \sum_{i=0}^{j-1} c_i^{R,\alpha} + \sum_{i=j}^{n-1} 2\gamma N_{n-1}^{\max\{\epsilon,1-\epsilon\}} && (N_j = \ldots = N_{n-1}) \\
&> \sum_{i=0}^{j-1} c_i^{R,\alpha} + \frac{1}{2} N_{n-1}^{\epsilon} 2\gamma N_{n-1}^{\max\{\epsilon,1-\epsilon\}} && \textit{(more than } \tfrac{1}{2} N_{n-1}^{\epsilon} \textit{ updates)} \\
&\geq \sum_{i=0}^{j-1} c_i^{R,\alpha} + c_{n-1}^{R,\alpha} = \sum_{i=0}^{n-1} c_i^{R,\alpha} && (c_j^{R,\alpha} = \ldots = c_{n-2}^{R,\alpha} = 0).
\end{aligned}$$

Case 2: There is at least one minor rebalancing step caused by an update of the form $\delta R = \{(\alpha, \beta') \mapsto m'\}$ since state $\mathcal{Z}_j$. Let $\mathcal{Z}_\ell$ denote the state created after the previous minor rebalancing caused by an update of this form; thus, $c_\ell^{R,\alpha} = \ldots = c_{n-2}^{R,\alpha} = 0$. The minor rebalancing steps creating $\mathcal{Z}_\ell$ and $\mathcal{Z}_n$ move tuples with the A-value α between R_h and R_l in opposite directions. From state $\mathcal{Z}_\ell$ to state $\mathcal{Z}_n$, the number of such tuples either decreases from $\lceil \frac{3}{2} N_l^{\epsilon} \rceil$ to $\lceil \frac{1}{2} N_{n-1}^{\epsilon} \rceil - 1$ (heavy to light) or increases from $\lceil \frac{1}{2} N_l^{\epsilon} \rceil - 1$ to $\lceil \frac{3}{2} N_{n-1}^{\epsilon} \rceil$ (light to heavy). For this change to happen, the number of updates needs to be greater than N_{n-1}^{ϵ} since $N_l = N_{n-1}$ and $\min\{\lceil \frac{3}{2} N_l^{\epsilon} \rceil - (\lceil \frac{1}{2} N_{n-1}^{\epsilon} \rceil - 1), \lceil \frac{3}{2} N_{n-1}^{\epsilon} \rceil - (\lceil \frac{1}{2} N_l^{\epsilon} \rceil - 1)\} > N_{n-1}^{\epsilon}$. Then,

$$\begin{aligned}
\sum_{i=0}^{n-1} \hat{c}_i^{R,\alpha} &\geq \sum_{i=0}^{\ell-1} c_i^{R,\alpha} + \sum_{i=\ell}^{n-1} \hat{c}_i^{R,\alpha} && \textit{(by induction hypothesis)} \\
&= \sum_{i=0}^{\ell-1} c_i^{R,\alpha} + \sum_{i=\ell}^{n-1} 2\gamma N_{n-1}^{\max\{\epsilon,1-\epsilon\}} && (N_j = \ldots = N_{n-1}) \\
&> \sum_{i=0}^{\ell-1} c_i^{R,\alpha} + N_{n-1}^{\epsilon} 2\gamma N_{n-1}^{\max\{\epsilon,1-\epsilon\}} && \textit{(more than } N_{n-1}^{\epsilon} \textit{ updates)} \\
&> \sum_{i=0}^{\ell-1} c_i^{R,\alpha} + c_{n-1}^{R,\alpha} = \sum_{i=0}^{n-1} c_i^{R,\alpha} && (c_\ell^{R,\alpha} = \ldots = c_{n-2}^{R,\alpha} = 0).
\end{aligned}$$

Cases 1 and 2 imply that Inequality (5) holds for update sequences of length n.

This shows that Inequality (2) holds when the amortized cost of OnUpdate$(u_i, \mathcal{Z}_i)$ is

$$\hat{c}_i = \gamma N_i^{\max\{\epsilon,1-\epsilon\}} + 4\gamma N_i^{\min\{\epsilon,1-\epsilon\}} + 2\gamma N_i^{\max\{\epsilon,1-\epsilon\}} + \Gamma, \quad \text{for } 0 \leq i < n,$$

where Γ and γ are constants. The amortized cost $\hat{c}_i^{major}$ of major rebalancing is $4\gamma N_i^{\min\{\epsilon,1-\epsilon\}}$, and the amortized cost $\hat{c}_i^{minor}$ of minor rebalancing is $2\gamma N_i^{\max\{\epsilon,1-\epsilon\}}$. From the size invariant $\lfloor \frac{1}{4} N_i \rfloor \leq |\mathbf{D}_i| < N_i$ follows that $|\mathbf{D}_i| < N_i < 4(|\mathbf{D}_i| + 1)$ for $0 \leq i < n$, where $|\mathbf{D}_i|$ is the database size before update u_i. This implies that for any database $\mathbf{D}$, the amortized major rebalancing time is $\mathcal{O}(|\mathbf{D}|^{\min\{\epsilon,1-\epsilon\}})$, the amortized minor rebalancing time is $\mathcal{O}(|\mathbf{D}|^{\max\{\epsilon,1-\epsilon\}})$, and the overall amortized update time of IVM$^\epsilon$ is $\mathcal{O}(|\mathbf{D}|^{\max\{\epsilon,1-\epsilon\}})$. ◀

Given $\epsilon \in [0,1]$, IVM$^\epsilon$ maintains the triangle count query in $\mathcal{O}(|\mathbf{D}|^{\max\{\epsilon,1-\epsilon\}})$ amortized update time while using $\mathcal{O}(|\mathbf{D}|^{1+\min\{\epsilon,1-\epsilon\}})$ space. It thus defines a tradeoff between time and space parameterized by ϵ, as shown in Figure 1. IVM$^\epsilon$ achieves the optimal amortized update time $\mathcal{O}(|\mathbf{D}|^{\frac{1}{2}})$ at $\epsilon = \frac{1}{2}$, for which the space is $\mathcal{O}(|\mathbf{D}|^{\frac{3}{2}})$.

5 Conclusion and Future Work

This paper introduces IVM$^\epsilon$, an incremental maintenance approach to counting triangles under updates that exhibits a space-time tradeoff such that the space-time product is quadratic in the size of the database. IVM$^\epsilon$ can trade space for update time. The amortized update time can be as low as the square root of the database size, which is worst-case optimal conditioned on the Online Matrix-Vector Multiplication (OMv) conjecture. The space requirements of IVM$^\epsilon$ can be improved to linear while keeping the amortized update time optimal by using a refined partitioning that takes into account the degrees of data values for both variables (instead of one variable only) in each relation [13]. IVM$^\epsilon$ captures classical and factorized IVM as special cases with suboptimal, linear update time [13].

There are worst-case optimal algorithms for *join* queries in the *static* setting [17]. In contrast, IVM$^\epsilon$ is worst-case optimal for the *count* aggregate over the triangle join query in the *dynamic* setting. The latter setting poses challenges beyond the former. First, the optimality argument for static join algorithms follows from their runtime being linear(ithmic) in their output size; this argument does not apply to our triangle count query, since its output is a scalar and hence of constant size. Second, optimality in the dynamic setting requires a more fine-grained argument that exploits the skew in the data for different evaluation strategies, view materialization, and delta computation; in contrast, there are static worst-case optimal join algorithms that do not need to exploit skew, materialize views, nor delta computation.

This paper opens up a line of work on *dynamic worst-case optimal query evaluation algorithms*. The goal is a complete characterization of the complexity of incremental maintenance for arbitrary functional aggregate queries over various rings [1]. Different rings can be used as the domain of tuple multiplicities (or payloads). We used here the ring $(\mathbb{Z}, +, \cdot, 0, 1)$ of integers to support counting. The relational data ring supports payloads with listing and factorized representations of relations, and the degree-m matrix ring supports payloads with gradients used for learning linear regression models [19].

Towards the aforementioned goal, we would first like to find a syntactical characterization of all queries that admit incremental maintenance in (amortized) sublinear time. Using known (first-order, fully recursive, or factorized) incremental maintenance techniques, cyclic and even acyclic joins require at least linear update time. Our intuition is that this characterization is given by a notion of diameter of the query hypergraph. This class strictly contains the q-hierarchical queries, which admit constant-time updates [4].

Minor variants of IVM$^\epsilon$ can be used to maintain the counting versions of any query built using three relations, the 4-path query, and the Loomis-Whitney queries in worst-case optimal time [13]. The same conditional lower bound on the update time shown for the triangle count applies for most of the mentioned queries, too. This leads to the striking realization that, while in the static setting the counting versions of the cyclic query computing triangles and the acyclic query computing paths of length 3 have different complexities, $\mathcal{O}(|\mathbf{D}|^{\frac{3}{2}})$ and $\mathcal{O}(|\mathbf{D}|)$, and pose distinct computational challenges, they share the same complexity and can use a very similar approach in the dynamic setting. A further IVM$^\epsilon$ variant allows the constant-delay enumeration of all triangles after each update, while preserving the same optimal amortized update time as for counting triangles [13]. These variants exploit the fact that our amortization technique is agnostic to the query to maintain and the update mechanism. It relies on two prerequisites. First, rebalancing is performed by moving tuples between relation parts. Second, the number of moved tuples per rebalancing is asymptotically no more than the number of updates performed since the previous rebalancing.

References

1 Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. FAQ: Questions Asked Frequently. In *PODS*, pages 13–28, 2016.

2 N. Alon, R. Yuster, and U. Zwick. Finding and Counting Given Length Cycles. *Algorithmica*, 17(3):209–223, 1997.

3 Ziv Bar-Yossef, Ravi Kumar, and D Sivakumar. Reductions in Streaming Algorithms, with an Application to Counting Triangles in Graphs. In *SODA*, pages 623–632, 2002.

4 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering Conjunctive Queries Under Updates. In *PODS*, pages 303–318, 2017.

5 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering UCQs Under Updates and in the Presence of Integrity Constraints. In *ICDT*, pages 8:1–8:19, 2018.

6 Luciana S Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting Triangles in Data Streams. In *PODS*, pages 253–262, 2006.

7 Rada Chirkova and Jun Yang. Materialized Views. *Found. & Trends DB*, 4(4):295–405, 2012.

8 Graham Cormode and Hossein Jowhari. A Second Look at Counting Triangles in Graph Streams (Corrected). *Theor. Comput. Sci.*, 683:22–30, 2017.

9 Talya Eden, Amit Levi, Dana Ron, and C Seshadhri. Approximately Counting Triangles in Sublinear Time. In *FOCS*, pages 614–633, 2015.

10 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and Strengthening Hardness for Dynamic Problems via the Online Matrix-Vector Multiplication Conjecture. In *STOC*, pages 21–30, 2015.

11 Muhammad Idris, Martín Ugarte, and Stijn Vansummeren. The Dynamic Yannakakis Algorithm: Compact and Efficient Query Processing Under Updates. In *SIGMOD*, pages 1259–1274, 2017.

12 Hossein Jowhari and Mohammad Ghodsi. New Streaming Algorithms for Counting Triangles in Graphs. In *COCOON*, pages 710–716, 2005.

13 Ahmet Kara, Hung Q. Ngo, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Counting Triangles under Updates in Worst-Case Optimal Time. *CoRR*, abs/1804.02780, 2018. `arXiv:1804.02780`.

14 Christoph Koch, Yanif Ahmad, Oliver Kennedy, Milos Nikolic, Andres Nötzli, Daniel Lupei, and Amir Shaikhha. DBToaster: Higher-Order Delta Processing for Dynamic, Frequently Fresh Views. *VLDB J.*, 23(2):253–278, 2014.

15 Paraschos Koutris, Semih Salihoglu, and Dan Suciu. Algorithmic Aspects of Parallel Data Processing. *Found. & Trends DB*, 8(4):239–370, 2018.

16 Andrew McGregor, Sofya Vorotnikova, and Hoa T Vu. Better Algorithms for Counting Triangles in Data Streams. In *PODS*, pages 401–411, 2016.

17 Hung Q. Ngo. Worst-Case Optimal Join Algorithms: Techniques, Results, and Open Problems. In *PODS*, pages 111–124, 2018.

18 Hung Q. Ngo, Christopher Ré, and Atri Rudra. Skew Strikes Back: New Developments in the Theory of Join Algorithms. *SIGMOD Record*, 42(4):5–16, 2013.

19 Milos Nikolic and Dan Olteanu. Incremental View Maintenance with Triple Lock Factorization Benefits. In *SIGMOD*, pages 365–380, 2018.

20 Thomas Schwentick and Thomas Zeume. Dynamic Complexity: Recent Updates. *SIGLOG News*, 3(2):30–52, 2016.

21 Virginia Vassilevska Williams. On Some Fine-Grained Questions in Algorithms and Complexity. In *ICM*, volume 3, pages 3431–3472, 2018.

22 Thomas Zeume. The Dynamic Descriptive Complexity of k-Clique. *Inf. Comput.*, 256:9–22, 2017.

A Formal Framework for Complex Event Processing

Alejandro Grez
Pontificia Universidad Católica de Chile, Santiago, Chile
Millennium Institute for Foundational Research on Data, Santiago, Chile
ajgrez@uc.cl

Cristian Riveros
Pontificia Universidad Católica de Chile, Santiago, Chile
Millennium Institute for Foundational Research on Data, Santiago, Chile
cristian.riveros@uc.cl

Martín Ugarte
Millennium Institute for Foundational Research on Data, Santiago, Chile
martin@martinugarte.com

Abstract

Complex Event Processing (CEP) has emerged as the unifying field for technologies that require processing and correlating distributed data sources in real-time. CEP finds applications in diverse domains, which has resulted in a large number of proposals for expressing and processing complex events. However, existing CEP languages lack from a clear semantics, making them hard to understand and generalize. Moreover, there are no general techniques for evaluating CEP query languages with clear performance guarantees.

In this paper we embark on the task of giving a rigorous and efficient framework to CEP. We propose a formal language for specifying complex events, called CEL, that contains the main features used in the literature and has a denotational and compositional semantics. We also formalize the so-called selection strategies, which had only been presented as by-design extensions to existing frameworks. With a well-defined semantics at hand, we discuss how to efficiently process complex events by evaluating CEL formulas with unary filters. We start by studying the syntactical properties of CEL and propose rewriting optimization techniques for simplifying the evaluation of formulas. Then, we introduce a formal computational model for CEP, called complex event automata (CEA), and study how to compile CEL formulas with unary filters into CEA. Furthermore, we provide efficient algorithms for evaluating CEA over event streams using constant time per event followed by constant-delay enumeration of the results. Finally, we gather the main results of this work to present an efficient and declarative framework for CEP.

2012 ACM Subject Classification Information systems → Data streams; Theory of computation → Data structures and algorithms for data management; Theory of computation → Database query languages (principles); Theory of computation → Automata extensions

Keywords and phrases Complex event processing, streaming evaluation, constant delay enumeration

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.5

Acknowledgements Cristian and Alejandro have been funded by FONDECYT grant 11150653, and together with Martín they were partially supported by the Millennium Institute for Foundational Research on Data (IMFD). M. Ugarte also acknowledges support from the Brussels Captial Region – Innoviris (project SPICES). We also thank the anonymous referees for their helpful comments.

1 Introduction

Complex Event Processing (CEP) has emerged as the unifying field of technologies for detecting situations of interest under high-throughput data streams. In scenarios like Network Intrusion Detection [39], Industrial Control Systems [29] or Real-Time Analytics [42], CEP

systems aim to efficiently process arriving data, giving timely insights for implementing reactive responses to complex events. Prominent examples of CEP systems from academia and industry include SASE [49], EsperTech [1], Cayuga [26], TESLA/T-Rex [22, 23], among others (see [24] for a survey). The main focus of these systems has been in practical issues like scalability, fault tolerance, and distribution, with the objective of making CEP systems applicable to real-life scenarios. Other design decisions, like query languages, are generally adapted to match computational models that can efficiently process data (see for example [50]). This has produced new data management and optimization techniques, generating promising results in the area [49, 1].

Unfortunately, as has been claimed several times [27, 51, 22, 11] CEP query languages lack a simple and denotational semantics, which makes them difficult to understand, extend or generalize. Their semantics are generally defined either by examples [36, 4, 21], or by intermediate computational models [49, 44, 40]. Although there are frameworks that introduce formal semantics (e.g. [26, 15, 7, 22, 8]), they do not meet the expectations to pave the foundations of CEP languages. For instance, some of them have unintuitive behavior (e.g. sequencing is non-associative), or are severely restricted (e.g. nesting operators is not supported). One symptom of this problem is that iteration, which is fundamental in CEP, has not yet been defined successfully as a compositional operator. Since iteration is difficult to define and evaluate, it is usually restricted by not allowing nesting or reuse of variables [49, 26]. As a result of these problems, CEP languages are generally cumbersome.

The lack of simple denotational semantics makes query languages also difficult to evaluate. A common factor in CEP systems is to find sophisticated heuristics [50, 22] that cannot be replicated in other frameworks. Further, optimization techniques are usually proposed at the architecture level [37, 26, 40], which does not allow for a unifying optimization theory. Many CEP frameworks use automata-based models [26, 15, 7] for query evaluation, but these models are usually complicated [40, 44], informally defined [26] or non-standard [22, 5]. In practice this implies that, although finite state automata is a recurring approach in CEP, there is no general evaluation strategy with clear performance guarantees.

Given this scenario, the goal of this paper is to give solid foundations to CEP systems in terms of query language and query evaluation. Towards these goals, we first provide a formal language that allows for expressing the most common features of CEP systems, namely sequencing, filtering, disjunction, and iteration. We introduce complex event logic (CEL), a logic with well-defined compositional and denotational semantics. We also formalize the so-called *selection strategies*, an important notion of CEP that is usually discussed directly [50, 26] or indirectly [15] in the literature but has not been formalized at the language level.

We then focus on the evaluation of CEL. We propose a formal evaluation framework that considers three building blocks: (1) syntactic techniques for rewriting CEL queries, (2) a well-defined intermediate evaluation model, and (3) efficient translations and algorithms to evaluate this model. Regarding the rewriting techniques, we introduce the notions of well-formed and safe formulas in CEL, and show that these restrictions are relevant for query evaluation. Further, we give a general result on rewriting CEL formulas into the so-called LP-normal form, a normal form for dealing with unary filters. For the intermediate evaluation model, we introduce a formal computational model for the regular fragment of CEL, called *complex event automata* (CEA). We show that this model is closed under I/O-determinization and provide translations for CEL formulas with unary filters into CEA. More important, we show an efficient algorithm for evaluating CEA with clear performance guarantees: constant time per tuple followed by constant-delay enumeration of the output. Finally, we bring together our results to present a formal framework for evaluating CEL.

Related work. Active Database Management Systems (ADSMS) and Data Stream Management Systems (DSMS) process data streams, and thus they are usually associated with CEP systems. Both technologies aim to execute relational queries over dynamic data [19, 2, 9]. In contrast, CEP systems see data streams as a sequence of events where the arrival order is the guide for finding patterns inside streams (see [24] for a comparison between ADSMS, DSMS, and CEP). Therefore, DSMS query languages (e.g. CQL [10]) are incomparable with our framework since they do not focus on CEP operators like sequencing and iteration.

Query languages for CEP are usually divided into three approaches [24, 11]: logic-based, tree-based and automata-based models. Logic-based models have their roots in temporal logic or event calculus, and usually have a formal, declarative semantics [8, 12, 20] (see [13] for a survey). However, this approach does not include iteration as an operator or it does not model the output explicitly. Furthermore, their evaluation techniques rely on logic inference mechanisms which are radically different from our approach. Tree-based models [38, 35, 1] have also been used for CEP but their language semantics is usually non-declarative and their evaluation techniques are based on cost-models, similar to relational database systems.

Automata-based models are close to what we propose in this paper. Previous proposals (e.g. SASE[5], NextCEP[44], DistCED[40]) do not rely in a denotational semantics; their output is defined by intermediate automata models. This implies that either iteration cannot be nested [5] or its semantics is confusing [44]. Other proposals (e.g. CEDR[15], TESLA[22], PBCED[7]) are defined with a formal semantics but they do not include iteration. An exception is Cayuga[25], but its language does not allow reusing variables and sequencing is non-associative, which results in an unintuitive semantics. Our framework is comparable to these systems, but provides a well-defined language that is compositional, allowing arbitrary nesting of operators. Moreover, we present the first evaluation of CEP queries that guarantees constant time per event and constant-delay enumeration of the output.

Finally, there has been some research in theoretical aspects of CEP, e.g. in axiomatization of temporal models [48], privacy [32], and load shedding [31]. This literature does not study the semantics and evaluation of CEP and, therefore, is orthogonal to our work.

Organization. We give an intuitive introduction to CEP and our framework in Section 2. In Section 3 and 4 we formally present our logic and selection strategies. The syntactic structure of the logic is studied in Section 5. The computational model and compilation of formulas are studied in Section 6. In Section 7 we develop efficient evaluation techniques and in Section 8 we present a framework summarizing our results. Future work is discussed in Section 9. Due to space limitations all proofs are deferred to the journal version.

2 Events in action

We start by presenting the main features and challenges of CEP. The examples used in this section will also serve throughout the paper as running examples.

In a CEP setting, events arrive in a streaming fashion to a system that must detect certain *patterns* [24]. For the purpose of illustration assume there is a stream produced by wireless sensors positioned in a farm, whose main objective is to detect fires. As a first scenario, assume that there are three sensors, and each of them can measure both temperature (in Celsius degrees) and relative humidity (as the percentage of vapor in the air). Each sensor is assigned an id in $\{0, 1, 2\}$. The *events* produced by the sensors consist of the id of the sensor and a measurement of temperature or humidity. In favor of brevity, we write $T(id, tmp)$ for

type	H	T	H	H	T	T	T	H	H	...
id	2	0	0	1	1	0	1	1	0	...
value	25	45	20	25	40	42	25	70	18	
index	0	1	2	3	4	5	6	7	8	...

Figure 1 A stream S of events measuring temperature and humidity. "value" contains degrees and humidity for T- and H- events, respectively.

an event reporting temperature tmp from sensor with id id, and similarly $H(id, hum)$ for events reporting humidity. Figure 1 depicts such a stream: each column is an event and the *value* row is the temperature or humidity if the event is of type T or H, respectively.

The patterns to be detected are generally specified by domain experts. For the sake of illustration, assume that the position of sensor 0 is particularly prone to fires, and it has been detected that a temperature measurement above 40 degrees Celsius followed by a humidity measurement of less than 25% represents a fire with high probability. Let us intuitively explain how we can express this as a pattern (also called a *formula*) in our framework:

$$\varphi_1 = (T \texttt{ AS } x\,;\, H \texttt{ AS } y) \texttt{ FILTER } (x.tmp > 40 \ \wedge y.hum <= 25 \ \wedge x.id = 0 \ \wedge y.id = 0)$$

This formula is asking for two events, one of type temperature (T) and one of type humidity (H). The events of type temperature and humidity are given names x and y, respectively, and the two events are filtered to select only those pairs (x, y) representing a high temperature followed by a low humidity measured by sensor 0.

What should the evaluation of φ_1 over the stream in Figure 1 return? A first important remark is that event streams are noisy, and one does not expect the events matching a formula to be *contiguous* in the stream. Then, a CEP engine needs to be able to dismiss irrelevant events. The semantics of the *sequencing* operator (;) will thus allow for arbitrary events to occur in between the events of interest. A second remark is that in CEP the set of events matching a pattern, called a *complex event*, is particularly relevant to the end user. Every time that a formula matches a portion of the stream, the final user should retrieve the events that compose that portion of the stream. This means that the evaluation of a formula over a stream should output a set of *complex events*. In our framework, each complex event will be the set of indexes (stream positions) of the events that witness the matching of a formula. Specifically, let $S[i]$ be the event at position i of the stream S. What we expect for the output of formula φ_1 consists of sets $\{i, j\}$ such that $S[i]$ is of type T, $S[j]$ is of type H, $i < j$, and they satisfy the conditions expressed after the FILTER. By inspecting Figure 1, we can see that the pairs satisfying these conditions are $\{1, 2\}$, $\{1, 8\}$, and $\{5, 8\}$.

Formula φ_1 illustrates the two most elemental features of CEP, namely *sequencing* and *filtering* [24, 9, 50, 2, 17]. But although it detects a set of possible fires, it restricts the *order* in which the two events occur: the temperature must be measured before the humidity. Naturally, this could prevent the detection of a fire in which the humidity was measured first. This motivates the introduction of *disjunction*, another common feature in CEP engines [24, 9]. To illustrate, we extend φ_1 by allowing events to appear in arbitrary order.

$$\varphi_2 = [(T \texttt{ AS } x\,;\, H \texttt{ AS } y) \texttt{ OR } (H \texttt{ AS } y\,;\, T \texttt{ AS } x)] \texttt{ FILTER } (x.tmp > 40 \ \wedge y.hum <= 25 \wedge x.id = 0 \ \wedge y.id = 0)$$

The OR operator allows for any of the two patterns to be matched. The result evaluation φ_2 over S (Figure 1) is the same as the evaluation of φ_1 plus the complex event $\{2, 5\}$.

The previous formulas show how CEP systems raise alerts when a certain complex event occurs. However, from a wider scope the objective of CEP is to retrieve information of interest from streams. For example, assume that we want to see how does temperature change in the location of sensor 1 when there is an increase of humidity. A problem here is that we do not know a priori the amount of temperature measurements; we need to capture an unbounded amount of events. The *iteration* operator + (a.k.a. Kleene closure) [24, 9, 30] is introduced in most CEP frameworks for solving this problem. This operator introduces many difficulties in the semantics of CEP languages. For example, since events are not required to occur contiguously, the nesting of + is particularly tricky and most frameworks simply disallow this (see [49, 10, 26]). Coming back to our example, the formula for measuring temperatures whenever an increase of humidity is detected by sensor 1 is:

$$\varphi_3 = [H \texttt{ AS } x\,;\, (T \texttt{ AS } y \texttt{ FILTER } y.id = 1)+\,;\, H \texttt{ AS } z] \\ \texttt{FILTER } (x.hum < 30 \ \wedge z.hum > 60 \wedge x.id = 1 \wedge z.id = 1)$$

Intuitively, variables x and z witness the increase of humidity from less than 30% to more than 60%, and y captures temperature measures between x and z. Note that the filter for y is included inside the + operator. Some frameworks allow to declare variables inside a + and filter them outside that operator (e.g. [49]). Although it is possible to define the semantics for that syntax, this form of filtering makes the definition of nesting + difficult. Another semantic subtlety of the + operator is the association of y to an event. Given that we want to match the event $(T \texttt{ AS } y \texttt{ FILTER } y.id = 1)$ an unbounded number of times: how should the events associated to y occur in the complex events generated as output? Associating different events to the same variable during evaluation has proven to make the semantics of CEP languages hard to extend. In Section 3, we introduce a semantics that allows nesting + and associate variables (inside + operators) to different events across repetitions.

Let us now explain the evaluation of φ_3 over S (Figure 1). The only two humidity events satisfying the top-most filter are $S[3]$ and $S[7]$, and the events in between that satisfy the inner filter are $S[4]$ and $S[6]$. As expected, $\{3, 4, 6, 7\}$ is part of the output. However, there are other complex events in the output. Since, as discussed, there might be irrelevant events between relevant ones, the semantics of + must allow for *skipping* arbitrary events. This implies that the complex events $\{3, 6, 7\}$ and $\{3, 4, 7\}$ are also part of the output.

The previous discussion raises an interesting question: are users interested in all complex events? Are some complex events more informative than others? Coming back to the output of φ_3 ($\{3, 6, 7\}$, $\{3, 4, 7\}$ and $\{3, 4, 6, 7\}$), one can easily argue that the largest complex event is more informative since all events are contained in it. The complex events output by φ_1 deserve a more thorough analysis. In this scenario, the pairs that have the same second component (e.g., $\{1, 8\}$ and $\{5, 8\}$) represent a fire occurring at the same place and time, so one could argue that only one of the two is necessary. For cases like above, it is common to find CEP systems that restrict the output by using so-called *selection strategies* (see for example [49, 50, 22]). Selection strategies are a fundamental feature of CEP. Unfortunately, they have only been presented as heuristics applied to particular computational models, and thus their semantics are given by algorithms and are hard to generalize. A special mention deserves the *next* selection strategy (called skip-till-next-match in [49, 50]) which models the idea of outputting only those complex events that can be generated without skipping relevant events. Although the semantics of *next* has been mentioned in previous papers (e.g [15]), it is usually underspecified [49, 50] or complicates the semantics of other operators [26]. In Section 4, we formally define a set of selection strategies including *next*.

Before formally presenting our framework, we illustrate one more common feature of CEP, namely *correlation*. Correlation is introduced by filtering events with predicates that involve more than one event. For example, consider that we want to see how does temperature change at some location whenever there is an increase of humidity, like in φ_3. What we need is a pattern where all the events are produced by the same sensor, but that sensor is not necessarily sensor 1. This is achieved by the following pattern:

$$\varphi_4 = [H \texttt{ AS } x; (T \texttt{ AS } y \texttt{ FILTER } y.id = x.id)+; H \texttt{ AS } z] \\ \texttt{FILTER } (x.hum < 30 \ \wedge z.hum > 60 \wedge x.id = z.id)$$

Notice that here the filters contain the predicates $x.id = y.id$ and $x.id = z.id$ that force all events to have the same id. Although this might seem simple, the evaluation of formulas that correlate events introduces new challenges. Intuitively, φ_4 is more complex because the id of x must be remembered in order to compare it with future incoming events. This behavior is clearly not "regular" and it will not be captured by a finite state model [33, 43]. In this paper, we study and characterize the regular core of CEP-systems. In sections 6 and 8 we focus on formulas without correlation. As we will see, the formal analysis of this fragment already presents non-trivial challenges, which is why we defer the analysis of formulas like φ_4 for future work. It is important to mention that the semantics of our language (including selection strategies) is general and includes more involved filters like correlation.

3 A query language for CEP

Having discussed the common operators and features of CEP, we proceed to formally introduce CEL (Complex Event Logic), our pattern language for capturing complex events.

Schemas, Tuples and Streams. Let $\mathbf{A}$ be a set of *attribute names* and $\mathbf{D}$ a set of values. A database schema $\mathcal{R}$ is a finite set of relation names, where each $R \in \mathcal{R}$ is associated to a tuple of attributes in $\mathbf{A}$ denoted by $\text{att}(R)$. If R is a relation name, then an R-tuple is a function $t : \text{att}(R) \to \mathbf{D}$. The type of an R-tuple t is R, and denote this by $\text{type}(t) = R$. For any relation name R, $\text{tuples}(R)$ denotes the set of all possible R-tuples, i.e., $\text{tuples}(R) = \{t : \text{att}(R) \to \mathbf{D}\}$. Similarly, for any database schema $\mathcal{R}$, $\text{tuples}(\mathcal{R}) = \bigcup_{R \in \mathcal{R}} \text{tuples}(R)$.

Given a schema $\mathcal{R}$, an *$\mathcal{R}$-stream* S is an infinite sequence $S = t_0 t_1 \ldots$ where $t_i \in \text{tuples}(\mathcal{R})$. When $\mathcal{R}$ is clear from the context, we refer to S simply as a stream. Given a stream $S = t_0 t_1 \ldots$ and a position $i \in \mathbb{N}$, the i-th element of S is denoted by $S[i] = t_i$, and the sub-stream $t_i t_{i+1} \ldots$ of S is denoted by S_i. Note that we consider that the time of each event is given by its index, and defer a more elaborated model (like [48]) for future work.

Let $\mathbf{X}$ be a set of variables. Given a schema $\mathcal{R}$, a predicate of arity n is an n-ary relation P over $\text{tuples}(\mathcal{R})$, i.e. $P \subseteq \text{tuples}(\mathcal{R})^n$. An atom is an expression $P(\bar{x})$ where P is an n-ary predicate and $\bar{x} \in \mathbf{X}^n$. As usual, we express predicates as formulas over attributes, and use $x.a$ to reffer to the attribute a of the tuple represented by x. For example, $P(x) := x.hum < 30$ is an atom and P is the predicate of all tuples that have a humidity attribute of less than 30. We consider that checking if a tuple t is in a predicate P takes time $\mathcal{O}(|t|)$, and that every atom $P(\bar{x})$ has constant size (and thus the size of a formula is independent of the type of predicates). We assume a fixed set of predicates $\mathbf{P}$ (i.e. defined by the CEP system). Moreover, we assume that $\mathbf{P}$ is closed under intersection, union, and complement, and $\mathbf{P}$ contains the predicate $P_R(x) := \text{type}(x) = R$ for checking if a tuple is an R-tuple for every $R \in \mathcal{R}$.

CEL syntax. Now we proceed to give the syntax of what we call the *core* of CEL (core-CEL for short), a logic inspired by the operations described in the previous section. This language contains the most essential CEP features. The set of formulas in core-CEL, or core formulas for short, is given by the following grammar:

$$\varphi \;:=\; R \;\texttt{AS}\; x \;\mid\; \varphi \;\texttt{FILTER}\; P(\bar{x}) \;\mid\; \varphi \;\texttt{OR}\; \varphi \;\mid\; \varphi\,;\varphi \;\mid\; \varphi+$$

where R is a relation name, x is a variable in $\mathbf{X}$ and $P(\bar{x})$ is an atom in $\mathbf{P}$. For example, all formulas in Section 2 are CEL formulas. Throughout the paper we use $\varphi \;\texttt{FILTER}\; (P(\bar{x}) \wedge Q(\bar{y}))$ or $\varphi \;\texttt{FILTER}\; (P(\bar{x}) \vee Q(\bar{y}))$ as syntactic sugar for $(\varphi \;\texttt{FILTER}\; P(\bar{x})) \;\texttt{FILTER}\; Q(\bar{y})$ or $(\varphi \;\texttt{FILTER}\; P(\bar{x})) \;\texttt{OR}\; (\varphi \;\texttt{FILTER}\; Q(\bar{y}))$, respectively. Unlike existing frameworks, we do not restrict the syntax, allowing for arbitrary nesting (in particular of +).

CEL semantics. We proceed to define the semantics of core formulas, for which we need to introduce some further notation. A *complex event* C is defined as a non-empty and finite set of indices. As mentioned in Section 2, a complex event contains the positions of the events that witness the matching of a formula over a stream, and moreover, they are the final output of evaluating a formula over a stream. We denote by $|C|$ the size of C and by $\min(C)$ and $\max(C)$ the minimum and maximum element of C, respectively. Given two complex events C_1 and C_2, $C_1 \cdot C_2$ denotes the *concatenation* of two complex events, that is, $C_1 \cdot C_2 := C_1 \cup C_2$ whenever $\max(C_1) < \min(C_2)$ and is undefined otherwise.

In core-CEL formulas, variables are only used to filter and select particular events, i.e. they are not retrieved as part of the output. As examples in Section 2 suggest, we are only concerned with finding the events that compose the complex events, and not which position corresponds to which variable. The reason behind this is that the operator + allows for repetitions, and therefore variables under (possibly nested) + operators would have a special meaning, particularly for filtering. This discussion motivates the following definitions. Given a formula φ we denote by $\mathrm{var}(\varphi)$ the set of all variables mentioned in φ (including filters), and by $\mathrm{vdef}(\varphi)$ all variables defined in φ by a clause of the form $R \;\texttt{AS}\; x$. Furthermore, $\mathrm{vdef}_+(\varphi)$ denotes all variables in $\mathrm{vdef}(\varphi)$ that are defined outside the scope of all + operators. For example, for $\varphi = (T \;\texttt{AS}\; x\,;\, (H \;\texttt{AS}\; y)+) \;\texttt{FILTER}\; z.id = 1$ we have that $\mathrm{var}(\varphi) = \{x, y, z\}$, $\mathrm{vdef}(\varphi) = \{x, y\}$, and $\mathrm{vdef}_+(\varphi) = \{x\}$. Finally, a valuation is a function $\nu : \mathbf{X} \to \mathbb{N}$. Given a finite set of variables $U \subseteq \mathbf{X}$ and two valuations ν_1 and ν_2, the valuation $\nu_1[\nu_2/U]$ is defined by $\nu_1[\nu_2/U](x) = \nu_2(x)$ if $x \in U$ and by $\nu_1[\nu_2/U](x) = \nu_1(x)$ otherwise.

We are ready to define the semantics of a core-CEL formula φ. Given a complex event C and a stream S, we say that C is in the evaluation of φ over S under valuation ν ($C \in [\![\varphi]\!](S, \nu)$) if one of the following conditions holds:

- $\varphi = R \;\texttt{AS}\; x$, $C = \{\nu(x)\}$, and $\mathrm{type}(S[\nu(x)]) = R$.
- $\varphi = \psi \;\texttt{FILTER}\; P(x_1, \ldots, x_n)$, $C \in [\![\psi]\!](S, \nu)$ and $(S[\nu(x_1)], \ldots, S[\nu(x_n)]) \in P$.
- $\varphi = \psi_1 \;\texttt{OR}\; \psi_2$ and $C \in [\![\psi_1]\!](S, \nu)$ or $C \in [\![\psi_2]\!](S, \nu)$.
- $\varphi = \psi_1\,;\,\psi_2$ and there are $C_1 \in [\![\psi_1]\!](S, \nu)$ and $C_2 \in [\![\psi_2]\!](S, \nu)$ such that $C = C_1 \cdot C_2$.
- $\varphi = \psi+$ and there exists ν' such that $C \in [\![\psi]\!](S, \nu[\nu'/U])$ or $C \in [\![\psi\,;\,\psi+]\!](S, \nu[\nu'/U])$, where $U = \mathrm{vdef}_+(\psi)$.

There are a couple of important remarks here. First, the valuation ν can be defined over a superset of the variables mentioned in the formula. This is important for sequencing (;) because we require the complex events from both sides to be produced with the same valuation. Second, when we evaluate a subformula of the form $\psi+$, we *carry* the value of variables defined outside the subformula. For example, the subformula $(T \;\texttt{AS}\; y \;\texttt{FILTER}\; y.id = x.id)+$ of φ_4 does not define the variable x. However, from the definition of the semantics we see

that x will be *already assigned* (because R `AS` x occurs outside the subformula). This is precisely where other frameworks fail to formalize iteration, as without this construct it is not easy to correlate the variables inside $+$ with the ones outside, as we illustrate with φ_4.

As previously discussed, in core-CEL variables are just used for comparing attributes with `FILTER`, but are not relevant for the final output. In consequence, we say that C belongs to the evaluation of φ over S (denoted $C \in [\![\varphi]\!](S)$) if there is a valuation ν such that $C \in [\![\varphi]\!](S, \nu)$. As an example, the complex events presented in Section 2 are indeed the outputs of φ_1 to φ_3 over the stream in Figure 1.

4 Selection strategies

Matching complex events is a computationally intensive task. As the examples in Section 2 suggest, the main reason behind this is that the amount of complex events can grow exponentially in the size of the stream, forcing systems to process large numbers of *candidate* outputs. In order to speed up the matching processes, it is common to restrict the set of results [18, 49, 50]. Unfortunately, most proposals in the literature restrict outputs by introducing heuristics to particular computational models without describing how the semantics are affected. For a more general approach, we introduce *selection strategies* (or *selectors*) as unary operators over core-CEL formulas. Formally, we define four selection strategies called strict (`STRICT`), next (`NXT`), last (`LAST`) and max (`MAX`). `STRICT` and `NXT` are motivated by previously introduced operators [49] under the name of *strict-contiguity* and *skip-till-next-match*, respectively. `LAST` and `MAX` are useful selection strategies from a semantic point of view. We define each selection strategy below, giving the motivation and formal semantics.

STRICT. As the name suggest, `STRICT` or strict-contiguity keeps only the complex events that are contiguous in the stream. To motivate this, recall that formula φ_1 in Section 2 detects complex events composed by a temperature above 40 degrees followed by a humidity of less than 25%. As already argued, in general one could expect other events between x and y. However, it could be the case that this pattern is of interest only if the events occur contiguously in the stream, or perhaps the stream has been preprocessed by other means and irrelevant events have been thrown out already. For this purpose, `STRICT` reduces the set of outputs selecting only strictly consecutive complex events. Formally, for any CEL formula φ we have that $C \in [\![\texttt{STRICT}(\varphi)]\!](S, \nu)$ holds if $C \in [\![\varphi]\!](S, \nu)$ and for every $i, j \in C$, if $i < k < j$ then $k \in C$ (i.e., C is an interval). In our running example, $\texttt{STRICT}(\varphi_1)$ would only produce $\{1, 2\}$, although $\{1, 8\}$ and $\{5, 8\}$ are also outputs for φ_1 over S.

NEXT. The second selector, `NXT`, is similar to the previously proposed operator *skip-till-next-match* [49]. The motivation behind this operator comes from a heuristic that consumes a stream skipping those events that cannot participate in the output, but matching patterns in a *greedy* manner that selects only the first event satisfying the next element of the query. In [49] the authors gave the definition of this strategy just as

> "*a further relaxation is to remove the contiguity requirements: all irrelevant events will be skipped until the next relevant event is read*" (*).

In practice, skip-till-next-match is defined by an evaluation algorithm that greedily adds an event to the output whenever a sequential operator is used, or adds as many events as possible whenever an iteration operator is used. The fact that the semantics is only defined

by an algorithm requires a user to understand the algorithm to write meaningful queries. In other words, this operator speeds up the evaluation by sacrificing the clarity of the semantics.

To overcome the above problem, we formalize the intuition behind (*) based on a special order over complex events. As we will see later, this allows to speed up the evaluation process as much as skip-till-next-match while providing clear and intuitive semantics. Let C_1 and C_2 be complex events. The symmetric difference between C_1 and C_2 ($C_1 \triangle C_2$) is the set of all elements either in C_1 or C_2 but not in both. We say that $C_1 \leq_{\text{next}} C_2$ if either $C_1 = C_2$ or $\min(C_1 \triangle C_2) \in C_2$. For example, $\{5,8\} \leq_{\text{next}} \{1,8\}$ since the minimum element in $\{5,8\} \triangle \{1,8\} = \{1,5\}$ is 1, which is in $\{1,8\}$. Note that this is intuitively similar to skip-till-next-match, as we are selecting the first relevant event. An important property is that the $\leq_{\text{next}}$-relation forms a total order among complex events, implying the existence of a minimum and a maximum over any finite set of complex events.

▶ **Lemma 1.** *$\leq_{\text{next}}$ is a total order between complex events.*

We can define now the semantics of `NXT`: for a CEL formula φ we have $C \in [\![\texttt{NXT}(\varphi)]\!](S,\nu)$ if $C \in [\![\varphi]\!](S,\nu)$ and for every complex event $C' \in [\![\varphi]\!](S,\nu)$, if $\max(C) = \max(C')$ then $C' \leq_{\text{next}} C$. In other words, C must be the $\leq_{\text{next}}$-maximum match among all matches that end in $\max(C)$. In our running example, we have that $\{1,8\}$ matches $\texttt{NXT}(\varphi_1)$ but $\{5,8\}$ does not. Furthermore, $\{3,4,6,7\}$ matches $\texttt{NXT}(\varphi_4)$ while $\{3,4,7\}$ and $\{3,6,7\}$ do not. Note that we compare outputs that have the same final position. This way, complex events are discarded only when there is a *preferred* complex event triggered by the same last event.

LAST. The `NXT` selector is motivated by the computational benefit of skipping irrelevant events in a greedy fashion. However, from a semantic point of view it might not be what a user wants. For example, if we consider again φ_1 and stream S (Section 2), we know that every complex event in $\texttt{NXT}(\varphi_1)$ will have event 1. In this sense, the `NXT` strategy selects the *oldest* complex event for the formula. We argue here that a user might actually prefer the opposite, i.e. the most recent explanation for the matching of a formula. This is the idea captured by `LAST`. Formally, the `LAST` selector is defined exactly as `NXT`, but changing the order $\leq_{\text{next}}$ by $\leq_{\text{last}}$: if C_1 and C_2 are two complex events, then $C_1 \leq_{\text{last}} C_2$ if either $C_1 = C_2$ or $\max(C_1 \triangle C_2) \in C_2$. For example, $\{1,8\} \leq_{\text{last}} \{5,8\}$. In our running example, $\texttt{LAST}(\varphi_1)$ would select the *most recent* temperature and humidity that explain the matching of φ_1 (i.e. $\{5,8\}$), which might be a better explanation for a possible fire. Surprisingly, we show in Section 7 that `LAST` enjoys the same good computational properties as `NXT`, even though it does not come from a greedy heuristic like `NXT` does.

MAX. A more ambitious selection strategy is to keep the maximal complex events in terms of set inclusion, which could be naturally more useful because these complex events are the *most informative*. Formally, given a CEL formula φ we say that $C \in [\![\texttt{MAX}(\varphi)]\!](S,\nu)$ holds iff $C \in [\![\varphi]\!](S,\nu)$ and for all $C' \in [\![\varphi]\!](S,\nu)$, if $\max(C) = \max(C')$ then $C \subseteq C'$. Coming back to φ_1, the `MAX` selector will output both $\{1,8\}$ and $\{5,8\}$, given that both complex events are maximal in terms of set inclusion. On the contrary, formula φ_3 produced $\{3,6,7\}$, $\{3,4,7\}$, and $\{3,4,6,7\}$. Then, $\texttt{MAX}(\varphi_3)$ will only produce $\{3,4,6,7\}$ as output, which is the maximal complex event. It is interesting to note that if we evaluate both $\texttt{NXT}(\varphi_3)$ and $\texttt{LAST}(\varphi_3)$ over the stream we will also get $\{3,4,6,7\}$ as the only output, illustrating that `NXT` and `LAST` also yield complex events with maximal information.

We have formally presented the foundations of a language for recognizing complex events, and how to restrict the outputs of this language in meaningful manners. Next we study practical aspects of the CEL syntax that impact how efficiently can formulas be evaluated.

5 Syntactic analysis of CEL

We now study the syntactic form of CEL formulas. We define *well-formed* and *safe* formulas, which are syntactic restrictions that characterize semantic properties of interest. Then, we define a convenient normal form and show that any formula can be rewritten in this form.

Syntactic restrictions of formulas. Although CEL has well-defined semantics, there are some formulas whose semantics can be unintuitive. Consider for example the formula $\varphi_5 = (H \texttt{ AS } x) \texttt{ FILTER } (y.tmp \leq 30)$. Here, x will be naturally bound to the only element in a complex event, but y will not *add* a new position to the output. By the semantics of CEL, a valuation ν for φ_5 must assign a position for y that satisfies the filter, but such position is not restricted to occur in the complex event. Moreover, y is not necessarily bound to any of the events seen up to the last element, and thus a complex event could depend on future events. For example, if we evaluate φ_5 over our running example S (Figure 1), we have that $\{2\} \in [\![\varphi_5]\!](S)$, but this depends on the event at position 6. This means that to evaluate this formula we potentially need to inspect events that occur after all events composing the output complex event have been seen, an arguably undesired situation.

To avoid this problem, we introduce the notion of *well-formed* formulas. As the previous example illustrates, this requires defining where variables are *bound* by a subformula of the form $R \texttt{ AS } x$. The set of bound variables of a formula φ is denoted by $\text{bound}(\varphi)$ and is recursively defined as follows:

$$\begin{aligned}
\text{bound}(R \texttt{ AS } x) &= \{x\} & \qquad \text{bound}(\psi \texttt{ FILTER } P(\bar{x})) &= \text{bound}(\psi) \\
\text{bound}(\psi_1 \texttt{ OR } \psi_2) &= \text{bound}(\psi_1) \cap \text{bound}(\psi_2) & \qquad \text{bound}(\psi+) &= \varnothing \\
\text{bound}(\psi_1 \,;\, \psi_2) &= \text{bound}(\psi_1) \cup \text{bound}(\psi_2) & \qquad \text{bound}(\texttt{SEL}(\psi)) &= \text{bound}(\psi)
\end{aligned}$$

where SEL is any selection strategy. We say that a CEL formula φ is *well-formed* if for every subformula of the form $\psi \texttt{ FILTER } P(\bar{x})$ and every $x \in \bar{x}$, there is another subformula ψ_x such that $x \in \text{bound}(\psi_x)$ and ψ is a subformula of ψ_x. This definition allows for including filters with variables defined in a wider scope. For example, formula φ_4 in Section 2 is well-formed although it has the not-well-formed formula $(T \texttt{ AS } y \texttt{ FILTER } y.id = x.id)+$ as a subformula.

One can argue that it would be desirable to restrict the users to only write well-formed formulas. Indeed, the well-formed property can be checked efficiently by a syntactic parser and users should understand that all variables in a formula must be correctly defined. Given that well-formed formulas have a well-defined variable structure, in the future we restrict our analysis to well-formed formulas.

Another issue for CEL is that the reuse of variables can easily produce unsatisfiable formulas. For example, the formula $\psi = T \texttt{ AS } x \,;\, T \texttt{ AS } x$ is not satisfiable (i.e. $[\![\psi]\!](S) = \varnothing$ for every S) because variable x cannot be assigned to two different positions in the stream. However, we do not want to be too conservative and disallow the reuse of variables in the whole formula (otherwise formulas like φ_2 in Section 2 would not be permitted). This motivates the notion of *safe* CEL formulas. We say that a CEL formula is *safe* if for every subformula of the form $\varphi_1 \,;\, \varphi_2$ it holds that $\text{vdef}_+(\varphi_1) \cap \text{vdef}_+(\varphi_2) = \varnothing$. For example, all CEL formulas in this paper are safe except for the formula ψ above.

The safe notion is a mild restriction to help evaluating CEL, and can be easily checked during parsing time. However, safe formulas are a subclass of CEL and it could be the case that they do not capture the full language. We show that this is not the case. Formally, we say that two CEL formulas φ and ψ are equivalent, denoted by $\varphi \equiv \psi$, if $[\![\varphi]\!](S) = [\![\psi]\!](S)$ for every stream S.

▶ **Theorem 2.** *Given a core-CEL formula φ, there is a safe formula φ' such that $\varphi \equiv \varphi'$ and $|\varphi'|$ is at most exponential in $|\varphi|$.*

By this result, we can restrict our analysis to safe formulas without loss of generality. Unfortunately, we do not know if the exponential size of φ' is unavoidable. We conjecture that this is the case, but we do not know yet the corresponding lower bound.

LP-normal form. Now we study how to rewrite CEL formulas to simplify the evaluation of unary filters. Intuitively, filter operators in a CEL formula can become difficult to handle for a query engine. To illustrate this, consider again formula φ_1 in Section 2. Syntactically, this formula states "find an event x followed by an event y, and then check that they satisfy the filter conditions". However, we would like an execution engine to only consider those events x with $id = 0$ that represent temperature above 40 degrees. Only afterwards the possible matching events y should be considered. In other words, formula φ_1 can be restated as:

$$\varphi_1' = [(T \texttt{ AS } x) \texttt{ FILTER } (x.tmp > 40 \wedge x.id = 0)]; \\ [(H \texttt{ AS } y) \texttt{ FILTER } (y.hum <= 25 \ \wedge y.id = 0)]$$

This example motivates defining the *locally parametrized* normal form (LP normal form). Let $\mathbf{U}$ be the set of all predicates $P \in \mathbf{P}$ of arity 1 (i.e. $P \subseteq \text{tuples}(\mathcal{R})$). We say that a formula φ is in LP-normal form if, for every subformula φ' FILTER $P(\bar{x})$ of φ with $P \in \mathbf{U}$, it holds that $\bar{x} = \{x\}$ and $\varphi' = R$ AS x for some R and x. In other words, all filters containing unary predicates are applied directly to the definitions of their variables. For instance, formula φ_1' is in LP-normal form while formulas φ_1 and φ_2 are not. Note that non-unary predicates are not restricted, and they can be used anywhere in the formula.

One can easily see that having formulas in LP-normal form would be an advantage for an evaluation engine, because it can *filter out* some events as soon as they arrive. However, formulas that are not in LP-normal form can still be very useful for declaring patterns. To illustrate this, consider the formula:

$$\varphi_6 = (T \texttt{ AS } x); ((T \texttt{ AS } y \texttt{ FILTER } x.temp \geq 40) \texttt{ OR } (H \texttt{ AS } y \texttt{ FILTER } x.temp < 40))$$

Here, the FILTER operator works like a conditional statement: if the x-temperature is greater than 40, then the following event should be a temperature, and a humidity event otherwise. This type of conditional statements can be very useful, but also hard to evaluate. Fortunately, the next result shows that one can always rewrite a formula into LP-normal form, incurring in the worst case in an exponential blow-up in the size of the formula.

▶ **Theorem 3.** *Let φ be a CEL formula. Then, there is a CEL formula ψ in LP-normal form such that $\varphi \equiv \psi$, and $|\psi|$ is at most exponential in $|\varphi|$.*

The importance of this result and Theorem 2 will become clear in the next sections, where we show that safe formulas in LP-normal form have good properties for evaluation. Similar to Theorem 2, we do not know if the exponential blow-up is unavoidable and leave this for future work.

6 A computational model for CEL

In this section, we introduce a formal computational model for evaluating CEL formulas called *complex event automata* (CEA for short). Similar to classical database management systems (DBMS), it is useful to have a formal model that stands between the query language

and the evaluation algorithms, in order to simplify the analysis and optimization of the whole evaluation process. There are several examples of DBMS that are based on this approach like regular expressions and finite state automata [33, 6], and SQL and relational algebra [3, 41]. Here, we propose CEA as the intermediate evaluation model for CEL and show later how to compile any (unary) CEL formula into a CEA.

As its name suggests, complex event automata (CEA) are an extension of *Finite State Automata* (FSA). The first difference from FSA comes from handling streams instead of words. A CEA is said to run over a stream of tuples, unlike FSA which run over words of a certain alphabet. The second difference arises directly from the first one by the need of processing tuples, which can have infinitely many different values, in contrast to the finite input alphabet of FSA. To handle this, our model is extended the same way as a Symbolic Finite Automata (SFA) [47]. SFAs are finite state automata in which the alphabet is described implicitly by a boolean algebra over the symbols. This allows automata to work with a possibly infinite alphabet and, at the same time, use finite state memory for processing the input. CEA are extended analogously, which is reflected in transitions labeled by unary predicates over tuples. The last difference addresses the need to generate complex events instead of boolean answers. A well known extension for FSA are *Finite State Transducers* [16], which are capable of producing an output whenever an input element is read. Our computational model follows the same approach: CEA are allowed to generate and output complex events when reading a stream.

Recall from Section 5 that $\mathbf{U}$ is the subset of unary predicates of $\mathbf{P}$. Let $\bullet, \circ$ be two symbols. A *complex event automaton* (CEA) is a tuple $\mathcal{A} = (Q, \Delta, I, F)$ where Q is a finite set of states, $\Delta \subseteq Q \times (\mathbf{U} \times \{\bullet, \circ\}) \times Q$ is the transition relation, and $I, F \subseteq Q$ are the set of initial and final states, respectively. Given a stream $S = t_0 t_1 \ldots$, a run ρ of $\mathcal{A}$ over S is a sequence of transitions: $\rho : q_0 \xrightarrow{P_0/m_0} q_1 \xrightarrow{P_1/m_1} \cdots \xrightarrow{P_n/m_n} q_{n+1}$ such that $q_0 \in I$, $t_i \in P_i$ and $(q_i, P_i, m_i, q_{i+1}) \in \Delta$ for every $i \leq n$. We say that ρ is *accepting* if $q_{n+1} \in F$ and $m_n = \bullet$. We denote by $\mathrm{Run}_n(\mathcal{A}, S)$ the set of accepting runs of $\mathcal{A}$ over S of length n. Further, $\mathrm{events}(\rho)$ is the set of positions where the run *marks* S, namely $\mathrm{events}(\rho) = \{i \in [0, n] \mid m_i = \bullet\}$. Intuitively this means that when a transition is taken, if the transition has the $\bullet$ symbol then the *current* position of the stream is included in the output (similar to the execution of a transducer). Note that we require the last position of an accepting run to be marking, as otherwise an output could depend on *future* events (see the discussion about well-formed formulas in Section 5). Given a stream S and $n \in \mathbb{N}$, we define the set of complex events of $\mathcal{A}$ over S at position n as $[\![\mathcal{A}]\!]_n(S) = \{\mathrm{events}(\rho) \mid \rho \in \mathrm{Run}_n(\mathcal{A}, S)\}$ and the set of all complex events as $[\![\mathcal{A}]\!](S) = \bigcup_n [\![\mathcal{A}]\!]_n(S)$. Note that $[\![\mathcal{A}]\!](S)$ can be infinite, but $[\![\mathcal{A}]\!]_n(S)$ is finite.

Consider as an example the CEA $\mathcal{A}$ depicted in Figure 2. In this CEA, each transition $P(x)|\bullet$ marks one H-tuple and each transition $P'(x)|\bullet$ marks a T-tuple with temperature bigger than 40. Note also that the transitions labelled by `TRUE`$|\circ$ allow $\mathcal{A}$ to arbitrarily skip tuples of the stream. Then, for every stream S, $[\![\mathcal{A}]\!](S)$ represents the set of all complex events that begin and end with an H-tuple and also contain some of the T-tuples with temperature higher than 40.

It is important to stress that CEA are designed to be an evaluation model for the unary fragment of CEL (a formal definition is presented in the next paragraph). Several computational models have been proposed for complex event processing [26, 40, 49, 44], but most of them are informal and non-standard extensions of finite state automata. In our framework, we want to take a step back compared to previous proposals and define a simple but powerful model that captures the *regular core* of CEL. Intuitively, formulas like φ_1, φ_2 and φ_3 in Section 2 can be evaluated using a bounded amount of memory. In contrast,

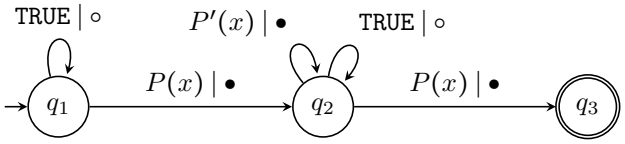


Figure 2 A CEA that can generate an unbounded amount of complex events. Here $P(x) := \text{type}(x) = H$ and $P'(x) := \text{type}(x) = T \wedge x.temp > 40$.

formula φ_4 needs unbounded memory to store *candidate* events seen in the past, and thus, it calls for a more sophisticated model (e.g. data automata [45]). Of course one would like to have a full-fledged model for CEL, but to this end we must first understand the regular fragment. A computational model for the whole CEP logic is left as future work.

Compiling unary CEL into CEA. We now show how to compile a well-formed and unary CEL formula φ into a CEA $\mathcal{A}_\varphi$. Formally, we say a CEL formula φ is equivalent to a CEA $\mathcal{A}$ if $[\![\varphi]\!](S) = [\![\mathcal{A}]\!](S)$ for every stream S. A CEL formula φ is unary if for every subformula of φ of the form φ' `FILTER` $P(\bar{x})$, it holds that $P(\bar{x})$ is a unary predicate (i.e. $P(\bar{x}) \in \mathbf{U}$). For example, formulas φ_1, φ_2, and φ_3 in Section 2 are unary, but formula φ_4 is not (the predicate $y.id = x.id$ is binary). As motivated in Section 2 and 5, despite their apparent simplicity unary formulas already present non-trivial computational challenges (see Section 7).

▶ **Theorem 4.** *For every well-formed formula φ in unary core-CEL, there is a CEA $\mathcal{A}_\varphi$ equivalent to φ. Furthermore, $\mathcal{A}_\varphi$ is of size at most linear in $|\varphi|$ if φ is safe and in LP-normal form, and at most double exponential in $|\varphi|$ otherwise.*

The proof of Theorem 4 is closely related with the safeness condition and the LP-normal form presented in Section 5. The construction first converts φ into an equivalent CEL formula φ' in LP-normal form (Theorem 3) and then builds an equivalent CEA from φ'. Unfortunately, there is an exponential blow-up for converting φ into LP-normal form. However, we show that the output is of linear size if φ' is safe, and of exponential size otherwise, suggesting that restricting the language to safe formulas allows for more efficient evaluation.

We have described the compilation process without considering selection strategies. To include them, we extend our notation and allow selection strategies to be applied over CEA. Given a CEA $\mathcal{A}$, a selection strategy $\texttt{SEL} \in \{\texttt{STRICT}, \texttt{NXT}, \texttt{LAST}, \texttt{MAX}\}$ and stream S, the set of outputs $[\![\texttt{SEL}(\mathcal{A})]\!](S)$ is defined analogously to $[\![\texttt{SEL}(\varphi)]\!](S)$ for a formula φ. Then, we say that a CEA $\mathcal{A}_1$ is equivalent to $\texttt{SEL}(\mathcal{A}_2)$ if $[\![\mathcal{A}_1]\!](S) = [\![\texttt{SEL}(\mathcal{A}_2)]\!](S)$ for every stream S.

▶ **Theorem 5.** *Let* `SEL` *be a selection strategy. For any CEA $\mathcal{A}$, there is a CEA $\mathcal{A}_{\texttt{SEL}}$ equivalent to $\texttt{SEL}(\mathcal{A})$. Furthermore, the size of $\mathcal{A}_{\texttt{SEL}}$ is, with respect to the size of $\mathcal{A}$, at most linear if* `SEL` = `STRICT`*, and at most exponential otherwise.*

At first this result might seem unintuitive, specially in the case of `NXT`, `LAST` and `MAX`. It is not immediate (and rather involved) to show that there exists a CEA for these strategies because they need to *track* an unbounded number of complex events using finite memory. Still, this can be done with an exponential blow-up in the number of states.

Theorem 5 concludes our study of the compilation of unary CEL into CEA. We have shown that not only is CEA able to evaluate CEL formulas, but it can also be exploited to evaluate selections strategies. We conclude this section by introducing the notion of I/O-determinism that will be crucial for our evaluation algorithms in the next section.

I/O-deterministic CEA. To evaluate CEA in practice we will focus on the class of the so-called *I/O-deterministic* CEA (for Input/Output deterministic). A CEA $\mathcal{A} = (Q, \Delta, I, F)$ is I/O-deterministic if $|I| = 1$ and for any two transitions (p, P_1, m_1, q_1) and (p, P_2, m_2, q_2), either P_1 and P_2 are mutually exclusive (i.e. $P_1 \cap P_2 = \varnothing$), or $m_1 \neq m_2$. Intuitively, this notion imposes that given a stream S and a complex event C, there is at most one run over S that generates C (thus the name referencing the input and the output). In contrast, the classic notion of determinism would allow for at most one run over the entire stream.

I/O-deterministic CEA are important because they allow for a simple and efficient evaluation algorithm (discussed in Section 7). But for this algorithm to be useful, we need to make sure that every CEA can be I/O determinized. Formally, we say that two CEA $\mathcal{A}_1$ and $\mathcal{A}_2$ are equivalent (denoted $\mathcal{A}_1 \equiv \mathcal{A}_2$) if for every stream S we have $[\![\mathcal{A}_1]\!](S) = [\![\mathcal{A}_2]\!](S)$.

▶ **Proposition 6.** *For every CEA $\mathcal{A}$ there is an I/O-deterministic CEA $\mathcal{A}'$ such that $\mathcal{A} \equiv \mathcal{A}'$, and $\mathcal{A}'$ is of size at most exponential over $|\mathcal{A}|$. That is, CEA are closed under I/O-determinization.*

This result and the compilation process allow us to evaluate any CEL formula by means of I/O-deterministic CEA without loss of generality.

7 Algorithms for evaluating CEA

In this section we show how to efficiently evaluate CEA. We start by formalizing the notion of *efficient evaluation* in CEP, which has not been formalized before in the CEP literature.

Efficiency in CEP. Defining a notion of efficiency for CEP is challenging since we would like to compute complex events in one pass and using a restricted amount of resources. Streaming algorithms [34, 28] are a natural starting point as they usually restrict the time allowed to process each tuple and the space needed to process the first n items of a stream (e.g., constant or logarithmic in n). However, an important difference is that in CEP the arrival of a single event might generate an exponential number of complex events as output. To overcome this problem, we propose to divide the evaluation in two parts: (1) consuming new events and updating the internal memory of the system and (2) generating complex events from the internal memory of the system. We require both parts to be as efficient as possible. First, (1) should process each event in a time that does not depend on the number of events seen in the past. Second, (2) should not spend any time *processing* and instead it should be completely devoted to generating the output. To formalize this notion, we assume that there is a special instruction $\texttt{yield}_S$ that returns the next element of a stream S. Then, given a function $f : \mathbb{N} \to \mathbb{N}$, a *CEP evaluation algorithm* with f-update time is an algorithm that evaluates a CEA $\mathcal{A}$ over a stream S such that:

1. between any two calls to $\texttt{yield}_S$, the time spent is bounded by $\mathcal{O}(f(|\mathcal{A}|) \cdot |t|)$, where t is the tuple returned by the first of such calls, and
2. maintains a data structure D in memory, such that after calling $\texttt{yield}_S$ n times, the set $[\![\mathcal{A}]\!]_n(S)$ can be enumerated from D with constant delay.

The notion of constant-delay enumeration was defined in the database community [46, 14] precisely for defining efficiency whenever generating the output might use considerable time. Formally, it requires the existence of a routine ENUMERATE that receives D as input and outputs all complex events in $[\![\mathcal{A}]\!]_n(S)$ without repetitions, while spending a constant amount of time before and after each output. Naturally, the time to generate a complex event C must be linear in $|C|$. We remark that **1.** is a natural restriction imposed in the streaming literature [34], while **2.** is the minimum requirement if an arbitrarily large set of arbitrarily large outputs must be produced [46].

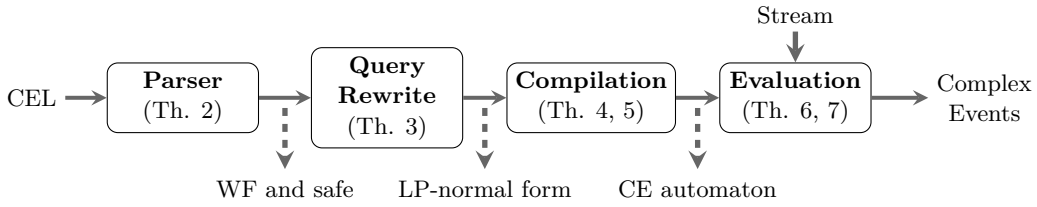


Figure 3 Evaluation framework for CEL.

Note that the update time $\mathcal{O}(f(|\mathcal{A}|) \cdot |t|)$ is linear in $|t|$ if we consider that $\mathcal{A}$ is fixed. Since this is the case in practice (i.e. the automaton is generally small with respect to the stream, and does not change during evaluation), this amounts to constant update time when measured under data complexity (tuples can also be considered of constant size).

Efficient evaluation of CEA. Having a good notion of efficiency, we proceed to show how to evaluate CEA efficiently. As it was previously discussed in Section 6, I/O deterministic CEA are specially designed for having CEP evaluation algorithms with linear update time. Furthermore, given that any CEA can be I/O-determinized (Proposition 6), this implies a CEP evaluation algorithm to evaluate any CEA. Unfortunately, the determinization procedure has an exponential blow-up in the size of the automaton, increasing the update time when the automaton is not I/O deterministic.

▶ **Theorem 7.** *For every I/O-deterministic CEA $\mathcal{A}$, there is a CEP evaluation algorithm with $|\mathcal{A}|$-update time. Furthermore, if $\mathcal{A}$ is any CEA, there is a CEP evaluation algorithm with $2^{|\mathcal{A}|}$-update time.*

We can further extend the CEP evaluation algorithm for I/O-deterministic CEA to any selection strategies by using the results of Theorem 5. However, by naively applying Theorem 5 and then I/O-determinizing the resulting automaton, we will have a double exponential blow-up. By doing the compilation of the selection strategies and the I/O-determinization together, we can lower the update time. Moreover, and rather surprisingly, we can evaluate `NXT` and `LAST` without determinizing the automaton, and therefore with linear update time.

▶ **Theorem 8.** *Let* `SEL` *be a selection strategy. For any CEA $\mathcal{A}$, there is a CEP evaluation algorithm for* $\texttt{SEL}(\mathcal{A})$. *Furthermore, the update time is $|\mathcal{A}|$ if* $\texttt{SEL} \in \{\texttt{NXT}, \texttt{LAST}\}$, $2^{|\mathcal{A}|}$ *if* $\texttt{SEL} = \texttt{STRICT}$ *and $4^{|\mathcal{A}|}$ if* $\texttt{SEL} = \texttt{MAX}$.

8 An evaluation framework for CEL

Having all the building blocks, we put all the results in perspectives and show how to evaluate unary CEL formulas. In Figure 3, we show the evaluation cycle of a CEL formula in our framework and how all the results and theorems fit together. To explain this framework, consider a unary CEL formula φ (possibly with selection strategies). The process starts in the parser module, where we check if φ is well-formed and safe. These conditions are important to ensure that φ is satisfiable and make a correct use of variables. Note that a CEP system could translate unsafe formulas (Theorem 2), incurring however in an exponential blow-up..

The next module rewrites a well-formed and safe formula φ into LP-normal form by using the rewriting process of Theorem 3. In the worst case this produces an exponentially larger formula. To avoid this, in many cases one can apply *local rewriting rules* [3, 41]. For example, in Section 2 we converted φ_1 into φ_1' by applying a *filter push*, avoiding the exponential

blow-up of Theorem 3. Unfortunately, we cannot apply this over formulas like φ_6 in Section 5. Nevertheless, formulas like φ_6 are rather uncommon in practice and local rewriting rules will usually produce LP-formulas of polynomial size.

The third module receives a formula in LP-normal form and builds a CEA $\mathcal{A}_\varphi$ of polynomial size (Theorem 4 and 5). Then, the last module runs $\mathcal{A}_\varphi$ over the stream by using our CEP evaluation procedure for I/O deterministic CEA (Theorem 7). If there is no selection strategy, $\mathcal{A}_\varphi$ must be determinized before running the CEP evaluation algorithm. In the worst case, this determinization is exponential in $\mathcal{A}_\varphi$, nevertheless, in practice the size of $\mathcal{A}_\varphi$ is rather small. If a selection strategy `SEL` is used, we can use the algorithms of Theorem 8 for evaluating $\texttt{SEL}(\mathcal{A}_\varphi)$, having a similar update time than evaluating $\mathcal{A}_\varphi$ alone. It is worth mentioning that evaluating $\texttt{NXT}(\mathcal{A}_\varphi)$ or $\texttt{LAST}(\mathcal{A}_\varphi)$ has even better performance than evaluating $\mathcal{A}_\varphi$ directly, given that the update time is linear in the size of $\mathcal{A}_\varphi$.

9 Future work

This paper settles new foundations for CEP systems, stimulating new research directions. In particular, a natural next step is to study the evaluation of non-unary CEL formulas. This requires new insight in rewriting formulas and a more powerful computational model with CEP evaluation algorithms. Another relevant problem is to understand the expressive power of different fragments of CEL and the relationship between the different operators. In this same direction, we envision as future work a generalization of the concept behind selection strategies, together with a thorough study of their expressive power.

Finally, we have focused on the fundamental features of CEP languages, leaving other features outside to keep the language and analysis simple. These features include correlation, time windows, aggregation, consumption policies, among others. We plan to extend CEL gradually with these features to establish a more complete and formal framework for CEP.

References

1 Esper Enterprise Edition website. Accessed on 2018-01-05. URL: `http://www.espertech.com/`.

2 D. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, C. Erwin, E. Galvez, M. Hatoun, A. Maskey, A. Rasin, A. Singer, M. Stonebraker, N. Tatbul, Y. Xing, R. Yan, and S. Zdonik. Aurora: A Data Stream Management System. In *SIGMOD*, 2003.

3 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases: the logical level*. Addison-Wesley, 1995.

4 Asaf Adi and Opher Etzion. Amit-the situation manager. *VLDB Journal*, 2004.

5 Jagrati Agrawal, Yanlei Diao, Daniel Gyllstrom, and Neil Immerman. Efficient pattern matching over event streams. In *SIGMOD*, 2008.

6 Alfred V. Aho. Algorithms for Finding Patterns in Strings. In *Handbook of Theoretical Computer Science*. Elsevier, 1990.

7 Mert Akdere, Uğur Çetintemel, and Nesime Tatbul. Plan-based complex event detection across distributed sources. *VLDB*, 2008.

8 Darko Anicic, Paul Fodor, Sebastian Rudolph, Roland Stühmer, Nenad Stojanovic, and Rudi Studer. A rule-based language for complex event processing and reasoning. In *RR*, 2010.

9 Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Keith Ito, Itaru Nishizawa, Justin Rosenstein, and Jennifer Widom. STREAM: The Stanford Stream Data Manager (Demonstration Description). In *SIGMOD*, 2003.

10 Arvind Arasu, Shivnath Babu, and Jennifer Widom. The CQL Continuous Query Language: Semantic Foundations and Query Execution. *The VLDB Journal*, 2006.

11 Alexander Artikis, Alessandro Margara, Martin Ugarte, Stijn Vansummeren, and Matthias Weidlich. Complex Event Recognition Languages: Tutorial. In *DEBS*, pages 7–10. ACM, 2017.

12 Alexander Artikis, Marek Sergot, and Georgios Paliouras. An event calculus for event recognition. *IEEE Transactions on Knowledge and Data Engineering*, 27(4):895–908, 2015.

13 Alexander Artikis, Anastasios Skarlatidis, François Portet, and Georgios Paliouras. Logic-based event recognition. *The Knowledge Engineering Review*, 27(4):469–506, 2012.

14 Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On Acyclic Conjunctive Queries and Constant Delay Enumeration. In *CSL*, 2007.

15 Roger S. Barga, Jonathan Goldstein, Mohamed H. Ali, and Mingsheng Hong. Consistent Streaming Through Time: A Vision for Event Stream Processing. In *CIDR*, 2007.

16 Jean Berstel. *Transductions and context-free languages*. Springer-Verlag, 2013.

17 Alejandro Buchmann and Boris Koldehofe. Complex event processing. *IT-Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik*, 2009.

18 Jan Carlson and Björn Lisper. A resource-efficient event algebra. *Science of Computer Programming*, 2010.

19 Jianjun Chen, David J. DeWitt, Feng Tian, and Yuan Wang. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In *SIGMOD*, 2000.

20 Federico Chesani, Paola Mello, Marco Montali, and Paolo Torroni. A logic-based, reactive calculus of events. *Fundamenta Informaticae*, 105(1-2):135–161, 2010.

21 Gianpaolo Cugola and Alessandro Margara. Raced: an adaptive middleware for complex event detection. In *Middleware*, 2009.

22 Gianpaolo Cugola and Alessandro Margara. TESLA: a formally defined event specification language. In *DEBS*, 2010.

23 Gianpaolo Cugola and Alessandro Margara. Complex Event Processing with T-REX. *The Journal of Systems and Software*, 2012.

24 Gianpaolo Cugola and Alessandro Margara. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys (CSUR)*, 2012.

25 Alan Demers, Johannes Gehrke, Mingsheng Hong, Mirek Riedewald, and Walker White. A general algebra and implementation for monitoring event streams. Technical report, Cornell University, 2005.

26 Alan Demers, Johannes Gehrke, Mingsheng Hong, Mirek Riedewald, and Walker White. Towards expressive publish/subscribe systems. In *EDBT*, 2006.

27 Antony Galton and Juan Carlos Augusto. Two approaches to event definition. In *DEXA*, 2002.

28 Lukasz Golab and M Tamer Özsu. Issues in data stream management. *Sigmod Record*, 2003.

29 Mikell P Groover. *Automation, production systems, and computer-integrated manufacturing*. Prentice Hall, 2007.

30 Daniel Gyllstrom, Jagrati Agrawal, Yanlei Diao, and Neil Immerman. On supporting kleene closure over event streams. In *ICDE 2008*, pages 1391–1393. IEEE, 2008.

31 Yeye He, Siddharth Barman, and Jeffrey F. Naughton. On Load Shedding in Complex Event Processing. In *ICDT*, pages 213–224, 2014.

32 Yeye He, Siddharth Barman, Di Wang, and Jeffrey F Naughton. On the complexity of privacy-preserving complex event processing. In *PODS*, pages 165–174, 2011.

33 John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

34 Elena Ikonomovska and Mariano Zelke. Algorithmic Techniques for Processing Data Streams. *Dagstuhl Follow-Ups*, 2013.

35 Mo Liu, Elke Rundensteiner, Kara Greenfield, Chetan Gupta, Song Wang, Ismail Ari, and Abhay Mehta. E-cube: multi-dimensional event sequence analysis using hierarchical pattern query sharing. In *SIGMOD*, pages 889–900, 2011.

36 D Luckham. Rapide: A language and toolset for simulation of distributed systems by partial orderings of events, 1996.

37 Masoud Mansouri-Samani and Morris Sloman. GEM: A generalized event monitoring language for distributed systems. *Distributed Systems Engineering*, 1997.

38 Yuan Mei and Samuel Madden. Zstream: a cost-based query processor for adaptively detecting composite events. In *SIGMOD*, pages 193–206. ACM, 2009.

39 Biswanath Mukherjee, L Todd Heberlein, and Karl N Levitt. Network intrusion detection. *IEEE network*, 1994.

40 Peter Pietzuch, Brian Shand, and Jean Bacon. A framework for event composition in distributed systems. In *Middleware*, 2003.

41 Raghu Ramakrishnan and Johannes Gehrke. *Database management systems (3 ed.)*. McGraw-Hill, 2003.

42 BS Sahay and Jayanthi Ranjan. Real time business intelligence in supply chain analytics. *Information Management & Computer Security*, 2008.

43 Jacques Sakarovitch. *Elements of automata theory*. Cambridge University Press, 2009.

44 Nicholas Poul Schultz-Møller, Matteo Migliavacca, and Peter Pietzuch. Distributed complex event processing with query rewriting. In *DEBS*, 2009.

45 Luc Segoufin. Automata and logics for words and trees over an infinite alphabet. In *CSL*, 2006.

46 Luc Segoufin. Enumerating with constant delay the answers to a query. In *ICDT 2013*, pages 10–20, 2013.

47 Margus Veanes. Applications of symbolic finite automata. In *CIAA*, 2013.

48 Walker White, Mirek Riedewald, Johannes Gehrke, and Alan Demers. What is next in event processing? In *PODS*, pages 263–272, 2007.

49 Eugene Wu, Yanlei Diao, and Shariq Rizvi. High-performance complex event processing over streams. In *SIGMOD*, 2006.

50 Haopeng Zhang, Yanlei Diao, and Neil Immerman. On complexity and optimization of expensive queries in complex event processing. In *SIGMOD*, 2014.

51 Detlef Zimmer and Rainer Unland. On the semantics of complex events in active database management systems. In *ICDE*, 1999.

A Formal Framework for Probabilistic Unclean Databases

Christopher De Sa
Cornell University, Ithacan, NY, USA
cdesa@cs.cornell.edu

Ihab F. Ilyas
University of Waterloo, Waterloo, ON, Canada
ilyas@uwaterloo.ca

Benny Kimelfeld
Technion - Israel Institute of Technology, Haifa, Israel
bennyk@cs.technion.ac.il

Christopher Ré
Stanford University, Stanford, CA, USA
chrismre@cs.stanford.edu

Theodoros Rekatsinas
University of Wisconsin - Madison, Madison, WI, USA
thodrek@cs.wisc.edu

Abstract

Most theoretical frameworks that focus on data errors and inconsistencies follow logic-based reasoning. Yet, practical data cleaning tools need to incorporate statistical reasoning to be effective in real-world data cleaning tasks. Motivated by empirical successes, we propose a formal framework for unclean databases, where two types of statistical knowledge are incorporated: The first represents a belief of how intended (clean) data is generated, and the second represents a belief of how noise is introduced in the actual observed database. To capture this noisy channel model, we introduce the concept of a Probabilistic Unclean Database (PUD), a triple that consists of a probabilistic database that we call the *intention*, a probabilistic data transformator that we call the *realization* and captures how noise is introduced, and an observed unclean database that we call the *observation*. We define three computational problems in the PUD framework: cleaning (infer the most probable intended database, given a PUD), probabilistic query answering (compute the probability of an answer tuple over the unclean observed database), and learning (estimate the most likely intention and realization models of a PUD, given examples as training data). We illustrate the PUD framework on concrete representations of the intention and realization, show that they generalize traditional concepts of repairs such as cardinality and value repairs, draw connections to consistent query answering, and prove tractability results. We further show that parameters can be learned in some practical instantiations, and in fact, prove that under certain conditions we can learn a PUD directly from a single dirty database without any need for clean examples.

2012 ACM Subject Classification Theory of computation → Data modeling; Theory of computation → Incomplete, inconsistent, and uncertain databases

Keywords and phrases Unclean databases, data cleaning, probabilistic databases, noisy channel

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.6

Related Version `https://arxiv.org/abs/1801.06750`

Funding *Ihab F. Ilyas*: This work was supported by NSERC under a Discovery Grant.
Benny Kimelfeld: This work was supported by the Israel Science Foundation (ISF) Grant 1295/15.
Theodoros Rekatsinas: This work was supported by the Wisconsin Alumni Association, Amazon under an ARA Award, and by NSF under grant IIS-1755676.

1 Introduction

Managing errors and inconsistency in databases is traditionally viewed as a challenge of a logical nature. It is typical that errors in a database are defined with respect to *integrity constraints* that capture normative aspects of downstream applications. The aim of integrity constraints is to guarantee the consistency of data used by these applications. Typically, an unclean database is defined as a database J that violates the underlying set of integrity constraints. In turn, a *repair* of a database J is a clean database I wherein all integrity constraints hold, and is obtained from J by a set of operations (e.g., deletions of tuples or updates of tuple values) that feature some form of non-redundancy [4, 2].

Various computational problems around unclean databases have been investigated in prior work [4, 28, 32, 29]. Past theoretical research has established fundamental results that concentrate on tractability boundaries for repair-checking and consistent query answering [2, 15, 26]. In their majority, these works adopt a deterministic interpretation of data repairs and cast all repairs equally likely. These theoretical developments have inspired practical tools that aim to automate data cleaning [7, 43, 12, 42, 20]. The majority of proposed methods assume as input a set of integrity constraints and use those to identify possible repairs via search-based procedures. To prioritize across possible repairs during search, the proposed methods rely on the notion of *minimality* [11, 23, 30]. Informally, minimality states that given two candidate sets of repairs, the one with fewer changes with respect to the original database is preferable. The use of minimality as an operational principle to find data repairs is a practical artifact that is used to limit the search space. These approaches to data cleaning suffer from two major drawbacks: First, they do not permit concrete statements about the "likelihood" of possible repairs. Consequently, they categorize query answers to a limited set of validity labels (e.g., certain, possible, and impossible); these labels might be unsuitable for downstream applications. Second, combinatorial principles such as minimality, while desired, do not entail the richness of the arguments and evidences (e.g., statistical features of data) that are needed to reason about and generate correct repairs.

Effective data cleaning needs to incorporate statistical reasoning. Our recent work on HoloClean [34] casts data repairing as a statistical learning and inference problem and reasons about a *most probable* repair instead of a *minimal* repair. Our study shows that HoloClean obtains more accurate data cleaning results than competing minimality-based data cleaning tools for a diverse array of real-world data cleaning scenarios [34]. HoloClean uses training data to learn a probabilistic model for how clean data is generated and how data errors are injected. HoloClean's model follows the *noisy channel model* [22], the de-facto probabilistic framework used in natural language tasks, such as spell checking and speech recognition, to reason about noisy data. To the best of our knowledge, existing theoretical frameworks for data cleaning do not capture this type of probabilistic reasoning.

Goals. We aim to establish a formal framework for *probabilistic unclean databases* (PUD) that adopts a statistical view of database cleaning. We do so by following the aforementioned noisy channel paradigm of HoloClean. Within the PUD framework, we formalize fundamental computational problems: *cleaning*, *query answering*, and *learning*. With that, we aim to draw connections between theoretical database research and important aspects of practical systems. In particular, our goal is to open the way for analyses and algorithms with theoretical guarantees for such systems. We argue that our framework is basic enough to allow for nontrivial theoretical advances, as illustrated by our preliminary results that *(a)* draw connections to traditional deterministic concepts, *and (b)* devise algorithms for special cases.

Probabilistic unclean databases. We view an unclean database as if a clean database I had been "distorted" via a noisy channel into a dirty database J; we aim to establish a model of this channel. Given the observed unclean database J, we seek the true database I from which J is produced. This model adopts Bayesian inference: out of all possible I, we seek the one for which the probability, given J, is highest. Following Bayes' rule, our objective is to find $\arg\max_I \Pr(I) \cdot \Pr(J|I)$. This objective decomposes in two parts: (1) the *prior* model for a clean database captured by $\Pr(I)$, and (2) the *channel* or *error* model characterized by $\Pr(J|I)$. To capture that, we define a *Probabilistic Unclean Database* (*PUD*) as a triple $(\mathcal{I}, \mathcal{R}, J^\star)$ where: (1) $\mathcal{I}$, referred to as the *intention model*, is a distribution that produces intended clean databases; (2) $\mathcal{R}$, referred to as the *realization model*, is a function that maps each clean database I to a distribution $\mathcal{R}_I$ that defines how noise is introduced into I; and (3) $J^\star$ is an observed unclean database. The distribution $\mathcal{I}$ defines the prior $\Pr(I)$ over clean databases, while the distribution $\mathcal{R}_I$ defines the aforementioned noisy channel $\Pr(J|I)$.

Computational problems. We define and study three computational problems in the PUD framework: (1) *data cleaning*, where given a PUD $(\mathcal{I}, \mathcal{R}, J)$, we seek to compute a database I that maximizes the probability $\mathcal{I}(I) \times \mathcal{R}_I(J)$; (2) *probabilistic query answering*, that is, the problem of evaluating a query Q over a PUD following the traditional possible tuple semantics [13, 40]; and (3) *learning* a PUD, where we consider *parametric* representations $\mathcal{I}_\Xi$ and $\mathcal{R}_\Theta$ of the intention and realization models, and seek to estimate the parameter vectors Ξ^* and Θ^* that maximize the likelihood of training data.

Preliminary analysis. PUDs allow for different instantiations of the intention and realization models. To establish preliminary complexity and convergence results, we focus on specific instantiations of the intention and realization models. We study intention models that can describe the distribution of tuple values as well as both soft and hard integrity constraints. We also focus on simple noise models. We study (1) realizations that introduce new tuples, hence, the clean database is a subset of the observed unclean database, and (2) realizations that update table cells, hence, the clean database is obtained via value repairs over the observed unclean database.

We present PUD instantiations for which solving the data cleaning problem has polynomial-time complexity. For instance, we show that in the presence of only one key constraint, soft or hard, data cleaning in PUDs can be solved in polynomial time. This result extends results for deterministic repairs that focus on hard integrity constraints to *weak* (soft) key constraints (e.g., two people are unlikely to, but might, have the same first and last name). Here, the most probable repair under the PUD framework may violate weak key constraints. We also draw connections between data cleaning in the PUD framework and *minimal repairs*. We identify conditions under which data cleaning in the PUD framework is equivalent to *cardinality repairs* [32] and *optimal V-repairs* [23]. For PUD learning, we consider both *supervised* and *unsupervised* learning. In the former case, we are given intension-realization pairs, and in the former, we are given only realizations (i.e., dirty databases). Our results discuss convexity and gradient computation for the optimization problem underlying the learning problem.

Our PUD model can be viewed as a generalization of the approach of Gribkoff et al. [19], who view the dirty database as a tuple-independent probabilistic database [13], and seek the *most-probable database* that satisfies a set of underlying integrity constraints (e.g., functional dependencies). In contrast, our modeling allows for arbitrary distributions over the intention, including ones with *weak* constraints that we discuss later on. Interestingly, our PUD model

goes in the reverse direction of the *operational* approach of Calautti et al. [10], who view the dirty database as a deterministic object and its *cleaning* (rather than the *error*) as a probabilistic process (namely a Markov chain of repairing operations).

Vision. This paper falls within the bigger vision of bridging database theory with learning theory as outlined in a recent position article [1]. We aim to draw connections between the rich theory on inconsistency management by the database community, and fundamentals of statistical learning theory with emphasis on *structured prediction* [5]. Structured prediction typically focuses on problems where, given a collection of observations, one seeks to predict the most likely assignment of values to structured objects. In most practical structured prediction problems, structure is encoded via logic-based constraints [18] in a way similar to how consistency is enforced in data cleaning. It is our hope that this paper will commence a line of work towards theoretical developments that take the benefit of both worlds, and will lead to new techniques that are both practical and rooted in strong foundations.

Organization. We begin with preliminary definitions in Section 2. In Section 3 we present the concept of PUDs. We present the three fundamental computational problems in Section 4, and describe preliminary results in Sections 5 and 6. We conclude with a discussion in Section 7. For space limitations, all proofs are in the extended version of our paper [36].

2 Preliminaries

We first introduce concepts, definitions and notation that we need throughout the paper.

Schemas and databases. A *relation signature* is a sequence $\alpha = (A_1, \ldots, A_k)$ of distinct *attributes* A_i, where k is the *arity* of α. A (*relational*) *schema* $\mathbf{S}$ has a finite set of *relation symbols*, and it associates each relation symbol R with a signature that we denote by $sig_{\mathbf{S}}(R)$, or just $sig(R)$ if $\mathbf{S}$ is clear from the context. We assume an infinite domain Const of *constants*. Let $\mathbf{S}$ be a schema, and let R be a relation symbol of $\mathbf{S}$. A *tuple* t over R is a sequence $(c_1, \ldots, c_k)$ of constants, where k is the arity of $sig(R)$. If $t = (c_1, \ldots, c_k)$ is a tuple over R and $sig(R) = (A_1, \ldots, A_k)$, then we refer to the value c_j as $t.A_j$ (where $j = 1, \ldots, k$). We denote by $\mathrm{tuples}(R)$ the set of all tuples over R.

In our databases, tuples have unique record identifiers. Formally, a table r over R is associated with a finite set $ids(r)$ of identifiers, and it maps each identifier i to a tuple $r[i]$ over R. A *database* I over $\mathbf{S}$ consists of a table R^I over each relation symbol R of $\mathbf{S}$, such that no two occurrences of tuples have the same identifier; that is, if R_1 and R_2 are distinct relation symbols in $\mathbf{S}$, then $ids(R_1^I)$ and $ids(R_2^I)$ are disjoint sets. We denote by $ids(I)$ the union of the sets $ids(R^I)$ over all relation symbols R of $\mathbf{S}$. If $i \in ids(R^I)$, then we may refer to the tuple $R^I[i]$ simply as $I[i]$.

A *cell* of a database I is a pair (i, A), where $i \in ids(R^I)$ for a relation symbol R, and A is an attribute inside $sig(R)$. We denote the cell (i, A) also by $i.A$, and we denote by $cells(I)$ the set of all cells of I.

Let I and J be databases over the same schema $\mathbf{S}$. We say that I is a *subset* of J if I can be obtained from J by deleting tuples, that is, $ids(R^I) \subseteq ids(R^J)$ for all relation symbols R of $\mathbf{S}$ (hence, $ids(I) \subseteq ids(J)$) and $I[i] = J[i]$ for all $i \in ids(I)$. We say that I is an *update* of J if I can be obtained from J by changing attribute values, that is, $ids(R^J) = ids(R^I)$ for all relation symbols R of $\mathbf{S}$.

A *query* Q over a schema $\mathbf{S}$ is associated with fixed arity, and it maps every database D over $\mathbf{S}$ into a finite set $Q(D)$ of tuples of constants over the fixed arity.

Integrity constraints. Various types of logical conditions are used for declaring integrity constraints, including *Functional Dependencies* (FDs), *conditional FDs* [7], *Denial Constraints* (DCs) [17], referential constraints [14], and so on. In this paper, by *integrity constraint* over a schema $\mathbf{S}$ we refer to a general expression φ of the form $\forall x_1, \ldots, x_m[\gamma(x_1, \ldots, x_m)]$, where $\gamma(x_1, \ldots, x_m)$ is a safe expression in Tuple Relational Calculus (TRC) over $\mathbf{S}$. For example, an FD $R : A \to B$ is expressed here as the integrity constraint

$$\forall x, y\, [(x \in R y \in R) \to (x.A = y.A \to x.B = y.B)]\ .$$

A *violation* of $\varphi = \forall x_1, \ldots, x_m[\gamma(x_1, \ldots, x_m)]$ in the database I is a sequence $i_1, \ldots, i_m$ of tuple identifiers in $ids(I)$ such that I violates $\gamma(I[i_1], \ldots, I[i_m])$, and we denote by $V(\varphi, I)$ the set of violations of φ in I. We say that I *satisfies* φ if I has no violations of φ, that is, $V(\varphi, I)$ is empty. Finally, I *satisfies* a set Φ of integrity constraints if I satisfies every integrity constraint φ in Φ.

Minimum repairs. Traditionally, database *repairs* are defined over inconsistent databases, where *inconsistencies* are manifested as violations of integrity constraints. A repair is a consistent database that is obtained from the inconsistent one by applying a *minimal* change, and we recall two types of repairs: *subset* (obtained by deleting tuples) and *update* (obtained by changing values). Moreover, the repairing operations may be weighted by tuple weights (in the first case) and cell weights (in the second case).

Formally, let $\mathbf{S}$ be a schema, Φ a set of integrity constraints over $\mathbf{S}$, and J a database that does not necessarily satisfy Φ. A *consistent subset* (resp., *consistent update*) of J is a subset (resp., update) I of J such that I satisfies Φ. A *minimum subset repair* of J w.r.t. a weight function $w : ids(J) \to [0, \infty)$ is a consistent subset I of J that minimizes the sum $\sum_{i \in ids(J) \setminus ids(I)} w(i)$. As a special case, a *cardinality repair* of J is a minimum subset repair w.r.t. a constant weight (e.g., $w(i) = 1$), that is, a consistent subset with a maximal number of tuples. A *minimum update repair* of J w.r.t. a weight function $w : cells(J) \times \mathsf{Const} \to [0, \infty)$ is a consistent update I of J that minimizes the sum $\sum_{i.A \in cells(I)} w(i.A, I[i].A)$.

Probabilistic databases. A *probabilisitic database* is a probability distribution over ordinary databases. As a representation system, our model is a generalization of the *Tuple-Independent probabilistic Database* (TID) wherein each tuple might either exist (with an associated probability) or not [13, 40]. In our model, each tuple comes from a general probability distribution over tuples (where inexistence is one of the options). This allows us to incorporate beliefs about the likelihood of tuples and cell values.

We now give the formal definition. Let $\mathbf{S}$ be a schema. A *generalized TID* is a database $\mathcal{K}$ that is defined similarly to an ordinary database over $\mathbf{S}$, except that instead of a tuple, the entry $R^{\mathcal{K}}[i]$ is a discrete probability distribution over the set $tuples(R) \cup \{\bot\}$, where the special value $\bot$ denotes that no tuple is generated. Hence, for every tuple t over R, the probability that $R^{\mathcal{K}}[i]$ produces t is given by $R^{\mathcal{K}}[i](t)$, or just $\mathcal{K}[i](t)$; moreover, the number $\mathcal{K}[i](\bot)$ is the probability that no tuple is generated for the identifier i. Therefore, $\mathcal{K}$ defines a probability distribution over databases I over $\mathbf{S}$ such that $ids(I) \subseteq ids(\mathcal{K})$ and the probability $\mathcal{K}(I)$ of a database I is defined as follows:

$$\mathcal{K}(I) \stackrel{\text{def}}{=} \prod_{i \in ids(I)} \mathcal{K}[i](I[i]) \times \prod_{i \in ids(\mathcal{K}) \setminus ids(I)} \mathcal{K}[i](\bot)$$

Table 1 Main symbols used in the framework.

$\mathbf{S}$	A schema.
$\mathcal{U}$	A PUD $(\mathcal{I}, \mathcal{R}, J^\star)$.
$\mathcal{I}$	An intention model (probabilistic database).
$\mathcal{R}$	A realization model, maps every I into a probabilistic database $\mathcal{R}_I$.
$J^\star$	An observed unclean database.
$\mathcal{R}\circ\mathcal{I}$	Distribution over pairs (I, J) given by $\mathcal{R}\circ\mathcal{I}\,(I, J) = \mathcal{I}(I) \cdot \mathcal{R}_I(J)$.
$\mathcal{U}^\star$	A probabilistic database given by $\mathcal{U}^\star(I') = \Pr_{(I,J)\sim\mathcal{R}\circ\mathcal{I}}(I = I' \mid J = J^\star)$.
$(\mathcal{D}, \tau, J^\star)$	A parfactor/subset PUD.
$(\mathcal{D}, \kappa, J^\star)$	A parfactor/update PUD.
$\mathcal{D}$	A parfactor database $(\mathcal{K}, \Phi, w)$ with $w : \Phi \to (0, \infty)$.
$\mathcal{K}$	A generalized tuple-independent database (generalized TID).
Φ	A set of integrity constraints φ.
τ	Maps $i \in \mathit{ids}(R^{\mathcal{D}})$ to a discrete distribution $\tau[i]$ over tuples$(R) \cup \{\bot\}$.
κ	Maps $(i, t) \in \mathit{ids}(R^{\mathcal{D}}) \times$ tuples(R) to a discrete distribution $\kappa[i, t]$ over tuples(R).

We incorporate weak integrity constraints by adopting the standard concept of *parametric factors* (or *parfactors* for short), which has been used in the *soft keys* of Jha et al. [21] and the *PrDB* model of Sen et al. [38], and which can be viewed as a special case of the *Markov Logic Network* (MLN) [35]. Under this concept, each constraint φ is associated with a weight $w(\varphi) > 0$ and each violation of φ contributes a factor of $\exp(-w(\varphi))$ to the probability of a random database I. Formally, a *parfactor database* over a schema $\mathbf{S}$ is a triple $\mathcal{D} = (\mathcal{K}, \Phi, w)$, where $\mathcal{K}$ is a generalized TID, Φ is a finite set of integrity constraints, both over $\mathbf{S}$, and $w : \Phi \to (0, \infty)$ is a weight function over Φ. The probability $\mathcal{D}(I)$ of a database I is defined as follows.

$$\mathcal{D}(I) \stackrel{\text{def}}{=} \frac{1}{Z} \times \mathcal{K}(I) \times \exp\left(-\sum_{\varphi\in\Phi} w(\varphi) \times |V(\varphi, I)|\right)$$

Recall that $V(\varphi, I)$ the set of violations of φ in I. The number Z is a *normalization factor* (also called the *partition function*) that normalizes the sum of probabilities to one:

$$Z \stackrel{\text{def}}{=} \sum_I \mathcal{K}(I) \times \exp\left(-\sum_{\varphi\in\Phi} w(\varphi) \times |V(\varphi, I)|\right)$$

Observe that the above sum is over a countable domain, since we assume that every $R^{\mathcal{K}}[i]$ is discrete (hence, there are countably many random databases I). Since we normalize the probability, it is not really necessarily for $\mathcal{K}$ to be normalized, as $\mathcal{D}$ would be a probability distribution even if $\mathcal{K}$ is *not* normalized. In fact, in our analysis, we will *not* make the assumption that $\mathcal{K}$ is normalized.

3 Probabilistic Unclean Databases

We introduce the Probabilistic Unclean Database (PUD) framework and describe examples of PUD instantiations that correspond to data cleaning applications in the HoloClean system [34]. In our framework, a PUD consists of three components following a noisy-channel model: *(1)* an *intention* model for generating clean databases, *(2)* a noisy *realization* model that can distort the intended clean database, *and (3)* an observed unclean database. The formal definition follows.

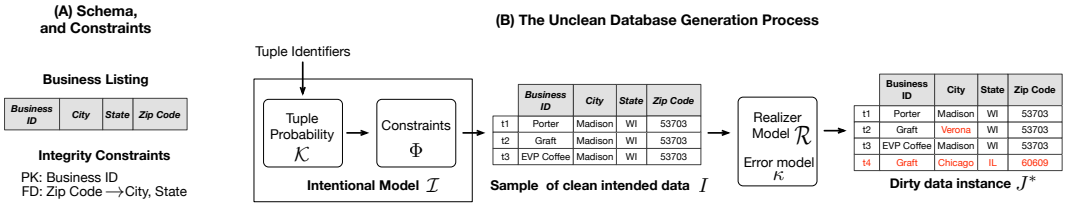


Figure 1 Overview of the PUD framework.

▶ **Definition 1.** *Let* $\mathbf{S}$ *be a schema. A PUD (over* $\mathbf{S}$*) is a triple* $\mathcal{U} = (\mathcal{I}, \mathcal{R}, J^\star)$ *where:*

1. $\mathcal{I}$ *is a probabilistic database, referred to as the* intention *model;*
2. $\mathcal{R}$*, referred to as the* realization *model, is a function that maps each database* I *to a probabilistic database* $\mathcal{R}_I$*;*
3. $J^\star$ *is a database referred to as the* observed *or* unclean *database.*

▶ **Example 2.** Figure 1 illustrates a high-level example of the PUD framework. We use a running example from business listings. Figure 1(A) depicts the schema $\mathbf{S}$ of the example. The constraints include a primary key and a functional dependency. Figure 1(B) depicts the unclean database generation process. Intention $\mathcal{I}$ outputs a valid database I with three tuples. The realizer $\mathcal{R}$ takes as input this database I, injects the new tuple $t4$ and updates the `City` value of tuple $t2$ from "Madison" to "Verona."

A PUD $\mathcal{U} = (\mathcal{I}, \mathcal{R}, J^\star)$ defines a probability distribution, denoted $\mathcal{R} \circ \mathcal{I}$, over pairs (I, J). Conditioning on $J = J^\star$, the PUD $\mathcal{U}$ also defines a probability distribution, denoted $\mathcal{U}^\star$, over intentions I (i.e., a probabilistic database). In the generative process of $\mathcal{R} \circ \mathcal{I}$, we sample the intention I from $\mathcal{I}$, and then we sample J from the realization $\mathcal{R}_I$. Hence, the probability of (I, J) is given by

$$\mathcal{R} \circ \mathcal{I}\,(I, J) \stackrel{\text{def}}{=} \mathcal{I}(I) \cdot \mathcal{R}_I(J)\,.$$

In the probabilistic database $\mathcal{U}^\star$, the probability of each candidate intention I' is given by

$$\mathcal{U}^\star(I') \stackrel{\text{def}}{=} \Pr_{(I,J)\sim\mathcal{R}\circ\mathcal{I}}(I = I' \mid J = J^\star) = \frac{\mathcal{R} \circ \mathcal{I}\,(I', J^\star)}{\sum_I \mathcal{R} \circ \mathcal{I}\,(I, J^\star)}$$

that is, the probability conditioned on the random J being $J^\star$. For this distribution to be well defined, we require $J^\star$ to have a nonzero probability; that is, there exists I such that $\mathcal{R} \circ \mathcal{I}\,(I, J^\star) > 0$. Table 1 lists the main symbols in the framework, along with their meaning.

3.1 Example Instantiations of PUDs

Our definition of a PUD is abstract, and not associated with any specific representation model. We now present concrete instantiations of PUD representations. These instantiations are probabilistic generalizations of the (deterministic) concepts of *subset repairs* [32, 2] and *update repair* [23, 30], respectively. More precisely, in both instantiations, the PUD $\mathcal{U} = (\mathcal{I}, \mathcal{R}, J^\star)$ is such that $\mathcal{I}$ is represented as a parfactor database $\mathcal{D}$ (as defined in Section 2) and $J^\star$ is an ordinary database (as expected); the two differ in the representation of the realization model $\mathcal{R}$. In the first instantiation, $\mathcal{R}$ is allowed to introduce new random tuples (hence, the intended database is a *subset* of the unclean one) and in the second, $\mathcal{R}$ is allowed to randomly change tuples (hence, the intended database is an *update* of the unclean one). Formally, let $\mathbf{S}$ be a schema.

Intended Database

Business ID	City	State	Zip Code
Porter	Madison	WI	53703
Graft	Madison	WI	53703
EVP Coffee	Madison	WI	53703

Dirty Database under Subset Realizer

Business ID	City	State	Zip Code
Porter	Madison	WI	53703
Graft	Madison	WI	53703
EVP Coffee	Madison	WI	53703
EVP Coffee	Madison	WI	53703

Dirty Database under Update Realizer

Business ID	City	State	Zip Code
Porter	Madison	WI	53703
Graft	Madison	WI	53704
EVP Coffee	adison	WI	53703

Figure 2 Examples of a subset realizer and an update realizer.

- A *parfactor/subset* PUD is a triple $(\mathcal{D}, \tau, J^\star)$ where $\mathcal{D}$ is a parfactor database, τ maps every identifier $i \in ids(R^{\mathcal{D}})$, where $R \in \mathbf{S}$, to a discrete distribution $\tau[i]$ over $tuples(R) \cup \{\bot\}$, and $J^\star$ is an ordinary database. As usual, $\bot$ means that no tuple is generated.
- A *parfactor/update* PUD is a triple $(\mathcal{D}, \kappa, J^\star)$ where $\mathcal{D}$ is a parfactor database, κ maps every identifier $i \in ids(R^{\mathcal{D}})$ and tuple $t \in tuples(R)$, where $R \in \mathbf{S}$, to a discrete distribution $\kappa[i,t]$ over $tuples(R)$, and $J^\star$ is an ordinary database.

In a parfactor/subset PUD $\mathcal{U} = (\mathcal{D}, \tau, J^\star)$, the probability $\mathcal{R}\circ\mathcal{I}(I, J)$ is then defined as follows. If I is not a subset of J, then $\mathcal{R}\circ\mathcal{I}(I, J) = 0$; otherwise:

$$\mathcal{R}\circ\mathcal{I}(I, J) \stackrel{\text{def}}{=} \mathcal{D}(I) \times \prod_{\substack{i \in ids(J) \setminus \\ ids(I)}} \tau[i](J[i]) \times \prod_{\substack{i \in ids(\mathcal{D}) \setminus \\ ids(J)}} \tau[i](\bot)$$

That is, $\mathcal{R}\circ\mathcal{I}(I, J)$ is the probability of I (i.e., $\mathcal{D}(I)$), multiplied by the probability that each new tuple of J is produced by τ (i.e., $\tau[i](J[i])$), multiplied by the probability that each tuple identifier i missing in J is indeed not produced (i.e., $\tau[i](\bot)$).

In a parfactor/update PUD $\mathcal{U} = (\mathcal{D}, \kappa, J^\star)$, the probability $\mathcal{R}\circ\mathcal{I}(I, J)$ is then defined as follows. If I is not an update of J, then $\mathcal{R}\circ\mathcal{I}(I, J) = 0$; otherwise:

$$\mathcal{R}\circ\mathcal{I}(I, J) \stackrel{\text{def}}{=} \mathcal{D}(I) \times \prod_{i \in ids(I)} \kappa[i, I[i]](J[i])$$

That is, $\mathcal{R}\circ\mathcal{I}(I, J)$ is the probability of I (i.e., $\mathcal{D}(I)$), multiplied by the probability that κ changes each tuple $I[i]$ to $J[i]$ (i.e., $\kappa[i, I[i]](J[i])$).

▶ **Example 3.** Figure 2 shows the intended database from Example 2 and two unclean versions obtained by a subset realizer and an update realizer. The subset realizer introduces a duplicate, while the update realizer introduces two typos. These correspond to two types of common errors in relational data. Our PUD framework can naturally model such cases.

In Section 5, we discuss connections between these PUD instantiations and the deterministic models. Finally, in Section 6, we provide more concrete cases of PUD instantiations.

4 Computational Problems

We define three computational problems over PUDs that are motivated by the need to clean and query unclean data, and learn the intention and realization models from observed data.

Data Cleaning. Given a PUD $(\mathcal{I}, \mathcal{R}, J^\star)$, we wish to compute a Most Likely Intention (MLI) database I, given the observed unclean database $J^\star$. We refer to this problem as *data cleaning* in PUDs.

▶ **Definition 4** (Cleaning). *Let* $\mathbf{S}$ *be a schema and* $\mathbf{R}$ *a representation system for PUDs. The problem* $(\mathbf{S}, \mathbf{R})$-cleaning *is that of computing an* MLI *of a given PUD* $\mathcal{U} = (\mathcal{I}, \mathcal{R}, J^\star)$*, that is, computing a database* I *such that the probability* $\mathcal{U}^\star(I)$ *is maximal (or, equivalently, the probability* $\mathcal{R} \circ \mathcal{I}\,(I, J^\star)$ *is maximal).*

Probabilistic query answering. A PUD defines a probabilistic database – a probability space over the intensions I. The problem of *Probabilistic Query Answering* (*PQA*) is that of evaluating a query over this probabilistic database. We adopt the standard semantics of query evaluation over probabilistic databases [13, 40], where the confidence in an answer tuple is its marginal probability.

▶ **Definition 5** (PQA). *Let* $\mathbf{S}$ *be a schema,* Q *a query over* $\mathbf{S}$*, and* $\mathbf{R}$ *a representation system for PUDs. The problem* $(\mathbf{S}, Q, \mathbf{R})$-PQA *is the following. Given a PUD* $\mathcal{U}$ *and a tuple* $\mathbf{a}$*, compute the* confidence *of* $\mathbf{a}$*, that is, the probability* $\Pr_{I \sim \mathcal{U}^\star}(\mathbf{a} \in Q(I))$.

For now, we assume that both $\mathcal{I}$ and $\mathcal{R}$ are fully specified. We next define the problem of learning models $\mathcal{I}$ and $\mathcal{R}$ using training (potentially labeled) data.

PUD learning. For a PUD $(\mathcal{I}, \mathcal{R}, J^\star)$, the models $\mathcal{I}$ and $\mathcal{R}$ are typically represented using numeric *parameters*. For example, the parameters of a parfactor/subset PUD $(\mathcal{D}, \tau, J^\star)$ are those needed to represent $\mathcal{D}$ (e.g., the weights of the constraints), and the parameters that define the distributions over the tuples in both $\mathcal{D}$ and τ. By a *parametric intention* we refer to an intension model $\mathcal{I}_\Xi$ with a vector Ξ of uninitialized parameters, and by $\mathcal{I}_{\Xi/\mathbf{c}}$ we denote the actual intention model where Ξ is assigned the values in the vector $\mathbf{c}$. Similarly, by a *parametric realization* we refer to a realization model $\mathcal{R}_\Theta$ with a vector Θ of uninitialized parameters, and by $\mathcal{R}_{\Theta/\mathbf{d}}$ we denote the actual realization model where Θ is set to $\mathbf{d}$.

Following the concept of *maximum likelihood estimation*, the goal in learning is to find the parameters that best explain (i.e., maximize the probability) of the training examples. In the *supervised* variant, we are given examples of both unclean databases and their clean versions; in the *unsupervised* variant, we are given only unclean databases.

▶ **Definition 6** (Learning). *Let* $\mathbf{S}$ *be a schema, and* $\mathbf{R}$ *a representation system for parametric intensions and realizations. In the following problems we are given, as part of the input, the parametric intention and realization models* $\mathcal{I}_\Xi$ *and* $\mathcal{R}_\Theta$*, respectively.*

- *In the* supervised $(\mathbf{S}, \mathbf{R})$-learning *problem, we are also given a collection* $(I_j, J_j)_{j=1}^n$ *of database pairs (intention-realization examples), and the goal is to find parameter values* $\mathbf{c}$ *and* $\mathbf{d}$ *that maximize* $\prod_{j=1}^n \mathcal{R} \circ \mathcal{I}\,(I_j, J_j)$ *for* $\mathcal{I} = \mathcal{I}_{\Xi/\mathbf{c}}$ *and* $\mathcal{R} = \mathcal{R}_{\Theta/\mathbf{d}}$.
- *In the* unsupervised $(\mathbf{S}, \mathbf{R})$-learning *problem, we are also given a collection* $(J_j)_{j=1}^n$ *of databases (realization examples), and the goal is to find parameter values* $\mathbf{c}$ *and* $\mathbf{d}$ *that maximize* $\prod_{j=1}^n \mathcal{R} \circ \mathcal{I}\,(J_j)$ *for* $\mathcal{I} = \mathcal{I}_{\Xi/\mathbf{c}}$ *and* $\mathcal{R} = \mathcal{R}_{\Theta/\mathbf{d}}$*, where* $\mathcal{R} \circ \mathcal{I}\,(J_j)$ *is the marginal probability of* J_j*, that is,* $\sum_I \mathcal{R} \circ \mathcal{I}\,(I, J_j)$.

Note that the summation in the unsupervised variant is over the sample space of the intention model $\mathcal{I}$. While the reader might be concerned about the source of many examples (I_j, J_j) and J_j in the phrasing of the learning problems, it is oftentimes the case that a single large example (I, J) (or just J in the unsupervised variant) can be decomposed into many smaller examples. This depends on the independence assumptions in the parametric models $\mathcal{I}_\Xi$ and $\mathcal{R}_\Theta$ as we discuss in Section 6.1. In the next sections, we give preliminary results on the introduced problems, focusing on parfactor/subset and parfactor/update PUDs.

5 Cleaning and Querying Unclean Data

In this section, we draw connections between data cleaning in the PUD framework (MLIs) and traditional *minimum repairs*. We also give preliminary results on the complexity of cleaning. Finally, we draw a connection between probabilistic query answering and certain answers.

5.1 Generalizing Minimum Repairs

We now show that the concept of an MLI in parfactor/subset PUDs generalizes the concept of a minimum subset repair, and the concept of an MLI in parfactor/update PUDs generalizes the concept of an optimal update repair. Minimum subset repairs correspond to MLIs of PUDs with hard (or heavy) constraints. Minimum update repairs correspond to MLIs over PUDs that assume both hard (or heavy) constraints and assumptions of independence among the attributes. From the viewpoint of computational complexity, this means that finding an exact MLI is not easier than finding a minimum repair, which is often computationally hard [30]. Therefore, we should aim for *approximation* guarantees (which have clear semantics in the probabilistic setting) if we wish to avoid restricting the generality of the input.

Subset repairs and parfactor/subset PUDs. Recall that in parfactor/subset PUDs (as defined in Section 3.1), every intention I with a nonzero probability is a subset of the observed unclean database $J^\star$. In particular, every MLI is subset of $J^\star$. Our first result relates cleaning in parfactor/subset PUDs to the traditional minimum subset (or *cardinality*) repairs. This result states, intuitively, that the notion of an MLI in a parfactor/subset PUD coincides with the notion of a minimum subset repair if the weight of the formulas is high enough and the probability of introducing error is small enough.

▶ **Theorem 7.** *Let $(\mathcal{D}, \tau, J^\star)$ be a parfactor/subset PUD with $\mathcal{D} = (\mathcal{K}, \Phi, w)$. For $i \in ids(J^\star)$, assume that $\mathcal{K}[i](\bot) > 0$ and $\tau[i](J^\star[i]) > 0$, let $q(i) = \mathcal{K}[i](J^\star[i])/(\mathcal{K}[i](\bot) \cdot \tau[i](J^\star[i]))$, and assume that $q(i) \geq 1$. There is a number M such that if $w(\varphi) > M$ for all $\varphi \in \Phi$ then the following are equivalent for all $I \subseteq J^\star$:*

1. *I is an MLI.*
2. *I is a minimum subset repair of $J^\star$ w.r.t. the weight function $w(i) = \log(q(i))$.*

Note that in the theorem, $q(i)$ is the ratio between $\mathcal{K}[i](J^\star[i])$, namely the probability that $\mathcal{K}$ produces the ith tuple of $J^\star$, and $\mathcal{K}[i](\bot) \cdot \tau[i](J^\star[i])$, namely the probability that $\mathcal{K}$ does not generate the ith tuple of $J^\star$ but τ does.

Next, we draw a similar connection between minimum update repairs and MLIs of parfactor/update PUDs.

Update repairs and parfactor/update PUDs. We now turn our attention to update repairs. Recall that in a parfactor/update PUD (defined in Section 3.1), the intended clean database I is assumed to be an update of the observed unclean database $J^\star$. We establish a result analogous to Theorem 7, stating conditions under which MLIs for parfactor/update PUDs coincide with traditional minimum update repairs.

Let $\mathbf{S}$ be a schema, and let $\mathcal{U} = (\mathcal{D}, \kappa, J^\star)$ be a parfactor/update PUD with $\mathcal{D} = (\mathcal{K}, \Phi, w)$. We say that $\mathcal{U}$ is *attribute independent* if $\mathcal{K}$ and κ feature probabilistic independence among the attributes. More precisely, if $i \in R^{\mathcal{D}}$ for $R \in \mathbf{S}$ with $sig(R) = (A_1, \ldots, A_k)$, then we assume that $\mathcal{K}[i](a_1, \ldots, a_k)$ can be written as $\mathcal{K}[i](a_1, \ldots, a_k) = \prod_{j=1}^{k} \mathcal{K}_{A_j}[i](a_j)$ and, for $t =$

$(b_1, \ldots, b_k)$, that $\kappa[i,t](a_1, \ldots, a_k)$ can be written as $\kappa[i,t](a_1, \ldots, a_k) = \prod_{j=1}^{k} \kappa_{A_j}[i, b_j](a_j)$. In particular, the choice of the value a_j depends only on b_j and not on other values $b_{j'}$.

The following theorem states that the concept of an MLI of a parfactor/update PUD coincides with the concept of a minimum update repair when the PUD is attribute independent and, moreover, the weight of the integrity constraints is high.

▶ **Theorem 8.** *Let $\mathcal{U} = (\mathcal{D}, \tau, J^\star)$ be an attribute-independent parfactor/update PUD with $\mathcal{D} = (\mathcal{K}, \Phi, w)$ such that $\mathcal{U}^\star(I) > 0$ for at least one consistent update I of $J^\star$. There is a number M such that if $w(\varphi) > M$ for all $\varphi \in \Phi$, then the following statements are equivalent for all $I \subseteq J^\star$:*

1. *I is an MLI.*
2. *I is a minimum update repair w.r.t. the weight function*

$$w(i.A, a) = -\log(\mathcal{K}_A[i](a) \cdot \kappa_A[i, a](J^\star[i].A)) \,.$$

Note that $\mathcal{K}_A[i](a) \cdot \kappa_A[i, a](J^\star[i].A)$ is the probability that a is produced by $\mathcal{K}$ for the cell $i.A$, and that a is then changed to $J^\star[i].A$ via κ. Also note that in the case where this product is zero, we slightly abuse the notation by assuming that the weight is infinity.

5.2 Complexity of Cleaning with Key Constraints

We now present a complexity result on computing an MLI of a parfactor/subset PUD in the presence of key constraints. The following theorem states that in the case of a single key constraint per relation (which is the common setup, e.g., for the analysis of certain query answering [26, 3, 24]), an MLI can be found in polynomial time. Note that we do not make any assumption about the parameters; in particular, it may be the case that an MLI violates the key constraints since the constraints are weak. Regarding the representation of the probability spaces $\mathcal{K}$ and τ, the only assumption we make is that, given a tuple t, the probabilities $\mathcal{K}[i](t)$ and $\tau[i](t)$ can be computed in polynomial time.

▶ **Theorem 9.** *Let $(\mathcal{D}, \tau, J^\star)$ be a parfactor/subset PUD with $\mathcal{D} = (\mathcal{K}, \Phi, w)$. If Φ consists of (at most) one key constraint per relation, and no relation of $J^\star$ has duplicate tuples, then an MLI can be computed in polynomial time.*

It is left for future investigation to seek additional constraints (e.g., functional dependencies) for which an MLI can be found in polynomial time. Note that Theorem 7 implies that (under conventional complexity assumptions) we cannot generalize the polynomial-time result to all sets of functional dependencies, since finding a minimum subset repair might be computationally hard [32, 30].

5.3 Probabilistic Query Answering

For probabilisitic query answering, we again focus on the parametric/subset PUDs, and now we draw a connection to *consistent query answering* over the cardinality repairs. Recall that a *consistent answer* for a query Q over an inconsistent database J is a tuple t that belongs to $Q(I)$ for every cardinality repair I of J.

Let $\mathbf{S}$ be a schema, $J^\star$ a database, and Φ a set of integrity constraints. Let $M = |ids(J^\star)|$. The *uniform* parfactor/subset PUD for $J^\star$ and Φ with the parameters p and u, denoted $\mathcal{U}_{p,u}(J^\star, \Phi)$ or just $\mathcal{U}_{p,u}$ if $J^\star$ and Φ are clear from the context, is the parfactor/subset PUD $(\mathcal{D}, \tau, J^\star)$ with $\mathcal{D} = (\mathcal{K}, \Phi, w)$ such that the following hold.

- For all $R \in \mathbf{S}$ we have $ids(R^{\mathcal{K}}) = ids(R^{J^\star})$ and $\mathcal{K}[i](x) = 1/M$ for every tuple identifier i and argument x in $R^{\mathcal{K}} \cup \{\bot\}$. The remaining mass (required to reaching 1) is given to an arbitrary tuple outside of $J^\star$.
- $w(\varphi) = u$ for every $\varphi \in \Phi$.
- $\tau[i](\bot) = p$, and $\tau[i](t) = (1-p)/M$ for every identifier i and tuple t in $R^{\mathcal{K}}$. Again, the remaining mass is given to an arbitrary tuple outside of $J^\star$.

Observe that $\mathcal{D}$ is defined in such a way that every subset I of $J^\star$ has the same prior probability $\mathcal{K}(I)$, namely $1/M^{|J^\star|}$.

The following theorem states that, for $\mathcal{U}_{p,u}$, the consistent answers are precisely the answers whose probability approaches one when all of the following hold: *(1)* the probability of introducing error (i.e., $1-p$) approaches zero; *and (2)* the weight of the weak constraints (i.e., u) approaches infinity, that is, the constraints strengthen towards hardness.

▶ **Theorem 10.** *Let $J^\star$ be a database, Φ a set of integrity constraints, Q a query, and t a tuple. The following are equivalent:*

1. *t is a consistent answer over the cardinality repairs.*
2. $\lim_{p \longrightarrow 1} \lim_{w \longrightarrow \infty} \Pr_{I \sim \mathcal{U}^\star_{p,u}}(t \in Q(I)) = 1$.

Therefore, Theorem 10 sheds light on the role that the consistent answers have in probabilisitic query answering over parfactor/subset PUDs.

6 Learning Probabilistic Unclean Databases

We now give preliminary results on PUD learning, focusing on parfactor/update PUDs. We begin by describing the setup we consider in this section and the representation system $\mathbf{R}$ we use to describe the parametric intention and realization models $\mathcal{I}_\Xi$ and $\mathcal{R}_\Theta$.

6.1 Setup

To discuss the learning of parameters, we need to specify the actual parametric model we assume. Let $\mathbf{S}$ be a schema, and let $\mathcal{U} = (\mathcal{D}, \kappa, J^\star)$ be a parfactor/update PUD with $\mathcal{D} = (\mathcal{K}, \Phi, w)$. Since we restrict the discussion to parfactor/update PUDs, we assume (without loss of generality) that the identifiers in $\mathcal{D}$ are exactly those in $J^\star$, that is, $ids(\mathcal{D}) = ids(J^\star)$.

In our setup, $\mathcal{K}$ of $\mathcal{D}$ and κ of $\mathcal{U}$ are expressed in a parametric form that allows us to define the parametric intention and realization models $\mathcal{I}_\Xi$ and $\mathcal{R}_\Theta$ as *Gibbs* distributions. We refer to these as *Gibbs parfactor/update PUD models*, and define them as follows.

Parametric intention. To specify $\mathcal{K}$, we assume that for each relation symbol $R \in \mathbf{S}$, the probability $\mathcal{K}[i](t)$, with $i \in ids(R^{\mathcal{D}})$, is expressed in the form of an exponential distribution $\mathcal{K}[i](t) = \exp\left(\sum_{f \in F} w_f f(t)\right)$ where each $f \in F$ is an arbitrary function (*feature*) over t, and each weight w_f is a real number.

For example, a feature $f \in F$ may be a function that takes as input a tuple and returns a value in $\{-1, 1\}$. An example feature f can state that $f(t) = 1$ if $t[\texttt{gender}] = \texttt{female}$ and, otherwise, $f(t) = -1$. Another example is $f[t] = 1$ if $t[\texttt{zip}]$ starts with $\texttt{53}$ and $t[\texttt{state}] = \texttt{WI}$, and otherwise $f[t] = -1$. Additional examples of such features include the ones used in our prior work on HoloClean [34] to capture the co-occurrence probability of attribute value pairs. Each weight w_f corresponds to a parameter of the model. An assignment to these weights gives as a probability distribution, similarly to probabilistic graphical models [25].

The parameter vector Ξ of the parametric intention model $\mathcal{I}_\Xi$ consists of two sets of parameters: the weights w_f for each feature $f \in F$, and the weights $w(\varphi)$, which we write as w_φ for uniformity of presentation, for each constraint $\varphi \in \Phi$. Thus, the overall parametric intention model $\mathcal{I}_\Xi$ is expressed as a parametric Gibbs distribution.

We will take a special interest in the case where the integrity constraints are *unary*, which means that they have the form $\forall x[\gamma(x)]$, where γ is quantifier free; hence, a unary constraint is a statement about a single tuple. Examples of unary constraints are restricted cases of conditional functional dependencies [7, 16]. An example of such a constraint can be "age smaller than 10 cannot co-occur with a salary greater than \$100k."

Parametric realization. We consider a parametric realization model that is similar to the parametric intention model presented above, which is again a parametric Gibbs distribution. For each relation symbol $R \in \mathbf{S}$ and every pair $(t, t') \in \text{tuples}(R) \times \text{tuples}(R)$, the probability $\kappa[i,t](t')$, with $i \in ids(\mathcal{D})$, is expressed in the form of the Gibbs distribution $\kappa[i,t](t') = \frac{1}{Z_\kappa(t)} \exp\left(\sum_{g \in G} w_g g(t,t')\right)$, where G is the set of features, and each g is an arbitrary function (*feature*) over (t,t'), each weight w_g is a real number, and $Z_\kappa(t)$ is a normalization constant defined as $Z_\kappa(t) = \sum_{t' \in \text{tuples}(R)} \exp\left(\sum_{g \in G} w_g g(t,t')\right)$. Hence, we get a parametric model for the probability distribution κ.

As an example, a feature function $g(t,t')$ may capture spelling errors: $g(t,t') = 1$ if $t'[\mathsf{city}]$ can be obtained by deleting one character from $t[\mathsf{city}]$, and otherwise, $g(t,t') = -1$. The parameter vector Θ of the parametric realization model $\mathcal{R}_\Theta$ consists of the set of weights w_g for each feature $g \in G$.

Assumptions. We make two assumption here. First, we assume that the attributes of all relation symbols in $\mathbf{S}$ take values over a finite and given set. This means that for each relation symbol $R \in \mathbf{S}$, $\text{tuples}(R)$ is also finite and given as input. Second, all features describing $\mathcal{K}$ and κ can be computed efficiently, that is, in polynomial time in the size of the input.

Obtaining examples for learning. For supervised learning, we require a training collection $(I_j, J_j)_{j=1}^n$ and for unsupervised learning, a training collection $(J_j)_{j=1}^n$. We would like to make the case that, oftentimes, a single large example can be broken down into many small examples. Recall that in our setup (Gibbs parfactor/update PUDs), cross-tuple correlations can be introduced only by the integrity constraints in Φ. Consider, for instance, the case where all constraints in Φ are unary. Then, we get *tuple-independent* parfactor/update PUDs, and each tuple identifier i can become an example database: $(I[i], J[i])$ in the supervised case, and $J[i]$ in the unsupervised case. For general constraints, cross-tuple correlations exist, and each example (I_j, J_j) and (J_j) can be obtained by taking correlated groups of tuples from $J^\star$ by considering different values for the attributes participating in each constraint $\varphi \in \Phi$. The number of tuples contained in each example depends on the constraints. This is a standard practice with parameterized probabilistic models as the ones we consider here [31].

6.2 Supervised Learning

We begin by considering supervised $(\mathbf{S}, \mathbf{R})$-learning. All results presented in this section build upon standard tools from statistical learning. We are given a collection $(I_j, J_j)_{j=1}^n$ of intention-realization examples, and the goal is to find parameter values $\mathbf{c}$ and $\mathbf{d}$ that maximize the likelihood of pairs $(I_j, J_j)_{j=1}^n$, that is, $\prod_{j=1}^n \mathcal{R} \circ \mathcal{I}(I_j, J_j)$ for $\mathcal{I} = \mathcal{I}_{\Xi/\mathbf{c}}$ and $\mathcal{R} = \mathcal{R}_{\Theta/\mathbf{d}}$. To facilitate the analysis, we write this objective function as a sum over terms by considering the negative log-likelihood $l(\Xi = \mathbf{c}, \Theta = \mathbf{d}; (I_j, J_j)_{j=1}^n) = -\log \prod_{j=1}^n \mathcal{R} \circ \mathcal{I}(I_j, J_j) = -\sum_{j=1}^n \log \mathcal{R} \circ \mathcal{I}(I_j, J_j)$, and seek parameter values $\mathbf{c}$ and $\mathbf{d}$ that minimize it. For parfactor/update PUDs we have that $\mathcal{R} \circ \mathcal{I}(I, J) = \mathcal{D}(I) \times \prod_{i \in ids(I)} \kappa[i, I[i]](J[i])$ where $\mathcal{D}(I) = \mathcal{K}(I) \times 1/Z \times \exp(-\sum_{\varphi \in \Phi} w(\varphi) \times |V(\varphi, I)|)$. For the Gibbs parfactor/update PUD, $\mathbf{c}$

corresponds to an assignment of parameters w_f describing $\mathcal{K}$ and parameters $w_\varphi = w(\varphi)$ for the constraints $\varphi \in \Phi$. Similarly, $\mathbf{d}$ corresponds to an assignment of parameters w_g describing κ. We use $Z(\mathbf{c})$ to denote the partition function Z under the parameters $\mathbf{c}$. We have:

$$l(\mathbf{c}, \mathbf{d}; (I_j, J_j)_{j=1}^n) = -\sum_{j=1}^{n} \log \left(\mathcal{K}(I_j; \mathbf{c}) \ \times \ \exp \left(-\sum_{\varphi \in \Phi} w_\varphi \times |V(\varphi, I_j)| \right) \right) + n \log Z(\mathbf{c})$$
$$- \sum_{j=1}^{n} \sum_{i \in ids(I_j)} \log \left(\kappa[i, I[i]; \mathbf{d}](J_j[i]) \right) \tag{1}$$

where $\mathcal{K}(I_j; \mathbf{c})$ denotes that $\mathcal{K}$ is parametrized by $\mathbf{c}$ and $\kappa[i, I[i]; \mathbf{d}]$ denotes that κ is parametrized by $\mathbf{d}$.

Our goal becomes to minimize the expression in (1). It is well-known from the ML literature that there is no analytical solution to such minimization problems, and one needs to use iterative *gradient-based* methods [25]. We investigate whether gradient-based methods can indeed find a global minimum, and whether computing the gradient of this objective during each iteration is tractable. For the first question, the answer is positive.

▶ **Proposition 11.** *$l(\Xi = \mathbf{c}, \Theta = \mathbf{d}; (I_j, J_j)_{j=1}^n)$ is a convex function of (Ξ, Θ).*

This proposition implies that the optimization objective for supervised learning has only global optima. Hence, it is guaranteed that any gradient-based optimization method will converge to a global optimum. Next, we study when the gradient of $l(\Xi = \mathbf{c}, \Theta = \mathbf{d}; (I_j, J_j)_{j=1}^n)$ with respect to $\mathbf{c}$ and $\mathbf{d}$ can be computed efficiently.

To compute the gradient of $l(\Xi = \mathbf{c}, \Theta = \mathbf{d}; (I_j, J_j)_{j=1}^n)$, one has to compute $\frac{\partial}{\partial c_l} \log Z(\mathbf{c})$. To compute this derivative a full inference step is required [25]. This is because computing this gradient amounts to computing the expected value for each feature (corresponding to each parameter c_l) according to the distribution defined by $\mathbf{c}$ [25, Proposition 20.2]. However, marginal inference is often #P-hard [18]. In our setup, the constraints in Φ correspond to features, and it is not clear whether the gradient can be efficiently computable. In general, one can still estimate the aforementioned gradient by using approximate inference methods such as *Markov Chain Monte Carlo* (MCMC) methods [9, 39] or *belief propagation* [41]. While effective in practice, these methods do not come with guarantees on the quality of the obtained solution. Next, we focus on an instance of PUD learning where exact inference is tractable (linear on tuples(R)), hence, we can compute the exact gradients of the aforementioned optimization objective efficiently.

Tuple independence. We focus on Gibbs parfactor/update PUD models where all constraints in Φ are unary. Here, $(I_j, J_j)_{j=1}^n$ corresponds to a collection of n examples, each of which having one tuple identifier. We use $I_j[0]$ and $J_j[0]$ to denote the tuples in the example (I_j, J_j). In the extended version of our paper [36], we show that the negative log-likelihood $l(\Xi = \mathbf{c}, \Theta = \mathbf{d}; (I_j, J_j)_{j=1}^n)$ factorizes as $l(\mathbf{c}, \mathbf{d}; (I_j, J_j)_{j=1}^n) = \sum_{j=1}^n l'(\mathbf{c}, \mathbf{d}; I_j[0], J_j[0])$, where each $l'(\mathbf{c}, \mathbf{d}; I_j[0], J_j[0])$ is a convex function of Ξ and Θ (see the extended version of our paper [36]). Moreover, we show that the gradient of each $l'(\mathbf{c}, \mathbf{d}; I_j[0], J_j[0])$ can be evaluated in time linear to tuples(R) where R is the relation corresponding to the tuple identifier associated with example (I_j, J_j) (see the extended version of our paper [36]). Hence, we get the following.

▶ **Theorem 12.** *Given a training collection* $(I_j, J_j)_{j=1}^n$ *and a Gibbs parfactor/update PUD model with unary constraints, the exact gradient of* $l(\Xi = \mathbf{c}, \Theta = \mathbf{d}; (I_j, J_j)_{j=1}^n)$ *can be evaluated in* $O(n \cdot \max_{R \in \mathbf{S}} |\text{tuples}(R)|)$ *time.*

The above theorem implies that convex-optimization techniques such as *stochastic gradient descent* [8] can be used to scale to large PUD learning instances (i.e., for large n).

A question that arises is about the number of examples (i.e., n) required to learn a PUD model. To answer this question, we study the convergence of *supervised* $(\mathbf{S}, \mathbf{R})$*-learning* for Gibbs parfactor/update PUD models with unary constraints. For that, we view the collection $(I_j, J_j)_{j=1}^n$ as independent and identically distributed (i.i.d.) examples, drawn from a distribution that corresponds to a Gibbs parfactor/update PUD model with unary constraints and true parameters $\mathbf{c}^\star$ and $\mathbf{d}^\star$. By the law of large numbers, the *Maximum Likelihood Estimates* (MLE) $\mathbf{c}$ and $\mathbf{d}$ are guaranteed to converge to $\mathbf{c}^\star$ and $\mathbf{d}^\star$ in probability. This means that for arbitrarily small $\epsilon > 0$ we have that $P(|\mathbf{c} - \mathbf{c}^\star| > \epsilon) \to 0$ as $n \to \infty$. The same holds for $\mathbf{d}$. Moreover, we show that the MLE $\mathbf{c}$ and $\mathbf{d}$ satisfy the property of *asymptotic normality* [27]. Intuitively, asymptotic normality states that the estimator not only converges to the unknown parameter, but it converges fast enough at a rate of $1/\sqrt{n}$. This implies that to achieve the error ϵ for $\mathbf{c}$ and $\mathbf{d}$, one only needs $n = O(\epsilon^{-2})$ training examples.

▶ **Theorem 13.** *Consider a training collection* $(I_j, J_j)_{j=1}^n$ *drawn i.i.d. from a Gibbs parfactor/update PUD model with unary constraints and true parameters* $\mathbf{c}^\star$ *and* $\mathbf{d}^\star$*. The maximum likelihood estimates* $\mathbf{c}$ *and* $\mathbf{d}$ *satisfy* asymptotic normality*, that is,*

$$\sqrt{n}\,(\mathbf{c} - \mathbf{c}^\star) \to \mathcal{N}\left(0, \Sigma^2_{\mathbf{c}^\star}\right) \text{ as } n \to \infty \text{ and } \sqrt{n}\,(\mathbf{d} - \mathbf{d}^\star) \to \mathcal{N}\left(0, \Sigma^2_{\mathbf{d}^\star}\right) \text{ as } n \to \infty$$

where $\Sigma^2_{\mathbf{c}^\star}$ *and* $\Sigma^2_{\mathbf{d}^\star}$ *are the asymptotic variance of the estimates* $\mathbf{c}$ *and* $\mathbf{d}$*.*

Note that both the *multivariate Gaussian* distribution $\mathcal{N}(\mu, \Sigma^2)$ and the *asymptotic variance* are defined in classic statistics literature [27].

6.3 Unsupervised Learning

We now present preliminary results for unsupervised $(\mathbf{S}, \mathbf{R})$-learning. We are given a training collection $(J_j)_{j=1}^n$ and seek to find $\mathbf{c}$ and $\mathbf{d}$ that minimize the negative log-likelihood $l(\Xi = \mathbf{c}, \Theta = \mathbf{d}; (J_j)_{j=1}^n) = -\sum_{j=1}^n \log \sum_I \mathcal{R} \circ \mathcal{I}(I, J_j)$. Again, there is no analytical solution for finding optimal $\mathbf{c}$ and $\mathbf{d}$. Hence, one needs to use iterative gradient-based approaches, and again the questions of convexity and gradient computation arise. In the general case, this function is *not* necessarily convex. Hence, gradient-based methods are not guaranteed to converge to a global optimum. However, one can still solve the corresponding optimization problem using non-convex optimization methods [6]. Nevertheless, we show next that when realizers do not introduce *too much error*, we can establish guarantees.

Low-noise condition. Consider a Gibbs parfactor/update PUD model. We say that a PUD defined by $\mathcal{I}_{\Xi/\mathbf{c}}$ and $\mathcal{R}_{\Theta/\mathbf{d}}$ satisfies the *low-noise condition* with probability p if the realizer introduces an error with probability at most p. That is, for all intensions I and identifiers $i \in ids(I)$, it is the case that $\Pr(J[i] = I[i] \mid I) \geq 1 - p$. We have the following.

▶ **Theorem 14.** *Consider a Gibbs parfactor/update PUD model where* Ξ *takes values from a compact convex set. Given a training collection* $(J_j)_{j=1}^n$*, there exists a fixed probability* $p > 0$ *such that, under the low-noise condition with probability* p*, the negative log-likelihood* $l(\Xi = \mathbf{c}, \Theta = \mathbf{d}; (J_j)_{j=1}^n)$ *is a convex function of* Ξ*.*

Hence, in certain cases, it is possible to find a global optimum of the overall negative log-likelihood. For that, the low-noise condition should hold with probability p that is also bounded, that is, it cannot be arbitrarily large. We can show that, if the low-noise condition holds with probability p, it is indeed bounded for Gibbs parfactor/update PUDs with unary constraints (see the extended version of our paper [36]). We then find a global optimum as follows. We assume a simple parametric realization $\mathcal{R}_\Theta$, that is, a model for which we can efficiently perform *grid search* over the space of parameter values $\mathbf{d}$. To find the global optimum for the negative log-likelihood, we solve a series of convex optimization problems over Ξ for different fixed $\Theta = \mathbf{d}$. For each of these problems, we are guaranteed to find a corresponding global optimum $\mathbf{c}$, and by performing a grid search we are guaranteed to find the overall global optima $\mathbf{c}$ and $\mathbf{d}$. This approach has been shown to converge for similar simple non-convex problems [37, 33].

Finally, similarly to supervised learning, the negative log-likelihood for fixed $\Theta = \mathbf{d}$ decomposes into a sum of convex losses over $(J_j)_{i=1}^n$ where each example J_j contains a single tuple. We use $J_j[0]$ to denote that tuple. We have that $l(\Xi = \mathbf{c}, \Theta = \mathbf{d}, (J_j)_{j=1}^n) = \sum_{j=1}^n l'(\mathbf{c}, \mathbf{d}; J_j[0])$ where $\mathbf{d}$ is fixed. We show that the gradient of each $l'(\mathbf{c}, \mathbf{d}; J[0])$ can be evaluated in polynomial time to tuples(R) where R is the relation corresponding to the tuple identifier associated with the example (J_j).

It is left for future work to find sufficient conditions for p to be bounded for PUD models with more general constraints, as well as the complexity and convergence aspects.

7 Concluding Remarks

Taking inspiration from our experience with the HoloClean system [34], we introduced the concept of Probabilistic Unclean Databases (PUDs), a framework for unclean data that follows a noisy channel approach to model how errors are introduced in data. We defined three fundamental problems in the framework: cleaning, probabilistic query answering, and PUD learning (parameter estimation). We introduced PUD instantiations that generalize the deterministic concepts of subset repairs and update repairs, presented preliminary complexity, convergence, and learnability results.

This paper opens up many research directions for future exploration. One is to investigate the complexity of cleaning in more general configurations than the ones covered here. Moreover, in cases where probabilistic cleaning is computationally hard, it is of natural interest to find *approximate* repairs that have a probability (provably) close to the maximum. Another direction is the complexity of probabilistic query answering and approximation thereof, starting with the most basic constraints (e.g., primary keys) and queries (e.g., determine the marginal probability of a fact). Finally, an important direction is to devise learning algorithms for cases beyond the ones we discussed here. In particular, it is of high importance to understand when we can learn parameters without training data, based only on the given dirty database, under more general noisy realization models than the ones discussed in this paper.

References

1 Serge Abiteboul, Marcelo Arenas, Pablo Barceló, Meghyn Bienvenu, Diego Calvanese, Claire David, Richard Hull, Eyke Hüllermeier, Benny Kimelfeld, Leonid Libkin, Wim Martens, Tova Milo, Filip Murlak, Frank Neven, Magdalena Ortiz, Thomas Schwentick, Julia Stoyanovich, Jianwen Su, Dan Suciu, Victor Vianu, and Ke Yi. Research Directions for Principles of Data Management (Abridged). *SIGMOD Record*, 45(4):5–17, 2016. doi:10.1145/3092931.3092933.

2 Foto N. Afrati and Phokion G. Kolaitis. Repair checking in inconsistent databases: algorithms and complexity. In *ICDT*, pages 31–41. ACM, 2009.

3 Periklis Andritsos, Ariel Fuxman, and Renée J. Miller. Clean Answers over Dirty Databases: A Probabilistic Approach. In *ICDE*, page 30. IEEE Computer Society, 2006.

4 Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent Query Answers in Inconsistent Databases. In *PODS*, pages 68–79. ACM, 1999. doi:10.1145/303976.303983.

5 Gükhan H. Bakir, Thomas Hofmann, Bernhard Schölkopf, Alexander J. Smola, Ben Taskar, and S. V. N. Vishwanathan. *Predicting Structured Data (Neural Information Processing)*. The MIT Press, 2007.

6 Dimitri P Bertsekas. *Nonlinear programming*. Athena scientific Belmont, 1999.

7 Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. Conditional Functional Dependencies for Data Cleaning. In *ICDE*, pages 746–755. IEEE, 2007.

8 Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

9 N. E. Breslow and D. G. Clayton. Approximate Inference in Generalized Linear Mixed Models. *Journal of the American Statistical Association*, 88(421):9–25, 1993.

10 Marco Calautti, Leonid Libkin, and Andreas Pieris. An Operational Approach to Consistent Query Answering. In *PODS*, pages 239–251. ACM, 2018.

11 Jan Chomicki and Jerzy Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Information and Computation*, 197(1):90–121, 2005.

12 Xu Chu, Ihab F. Ilyas, and Paolo Papotti. Holistic Data Cleaning: Putting Violations into Context. In *ICDE*, pages 458–469, 2013.

13 Nilesh N. Dalvi and Dan Suciu. Efficient Query Evaluation on Probabilistic Databases. In *VLDB*, pages 864–875. Morgan Kaufmann, 2004.

14 C. J. Date. Referential Integrity. In *VLDB*, pages 2–12. VLDB Endowment, 1981.

15 Ronald Fagin, Benny Kimelfeld, and Phokion G. Kolaitis. Dichotomies in the Complexity of Preferred Repairs. In *PODS*, pages 3–15, New York, NY, USA, 2015. ACM.

16 Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. Conditional Functional Dependencies for Capturing Data Inconsistencies. *ACM Trans. Database Syst.*, 33(2):6:1–6:48, June 2008.

17 Terry Gaasterland, Parke Godfrey, and Jack Minker. An Overview of Cooperative Answering. *J. Intell. Inf. Syst.*, 1(2):123–157, 1992.

18 Amir Globerson, Tim Roughgarden, David Sontag, and Cafer Yildirim. How Hard is Inference for Structured Prediction? In *ICML*, pages 2181–2190. JMLR.org, 2015.

19 Eric Gribkoff, Guy Van den Broeck, and Dan Suciu. The Most Probable Database Problem. In *BUDA*, 2014.

20 Ihab F. Ilyas. Effective Data Cleaning with Continuous Evaluation. *IEEE Data Eng. Bull.*, 39:38–46, 2016.

21 Abhay Kumar Jha, Vibhor Rastogi, and Dan Suciu. Query evaluation with soft-key constraints. In *PODS*, pages 119–128, 2008.

22 Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2009.

23 Solmaz Kolahi and Laks V. S. Lakshmanan. On approximating optimum repairs for functional dependency violations. In *ICDT*, volume 361, pages 53–62. ACM, 2009.

24 Phokion G. Kolaitis and Enela Pema. A dichotomy in the complexity of consistent query answering for queries with two atoms. *Inf. Process. Lett.*, 112(3):77–85, 2012.

25 Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009.

26 Paraschos Koutris and Jef Wijsen. Consistent Query Answering for Self-Join-Free Conjunctive Queries Under Primary Key Constraints. *ACM Trans. Database Syst.*, 42(2):9:1–9:45, 2017.

27 Solomon Kullback. *Information theory and statistics*. Courier Corporation, 1997.

28 Maurizio Lenzerini. Data Integration: A Theoretical Perspective. In *PODS*, pages 233–246, New York, NY, USA, 2002. ACM.

29 Leonid Libkin. Incomplete Data: What Went Wrong, and How to Fix It. In *PODS*, pages 1–13, New York, NY, USA, 2014. ACM.

30 Ester Livshits, Benny Kimelfeld, and Sudeepa Roy. Computing Optimal Repairs for Functional Dependencies. In *PODS*, pages 225–237. ACM, 2018.

31 Ben London, Bert Huang, Ben Taskar, and Lise Getoor. Collective Stability in Structured Prediction: Generalization from One Example. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 828–836, 17–19 June 2013.

32 Andrei Lopatenko and Leopoldo E. Bertossi. Complexity of Consistent Query Answering in Databases Under Cardinality-Based and Incremental Repair Semantics. In *ICDT*, pages 179–193, 2007.

33 Cong Ma, Kaizheng Wang, Yuejie Chi, and Yuxin Chen. Implicit Regularization in Nonconvex Statistical Estimation: Gradient Descent Converges Linearly for Phase Retrieval, Matrix Completion and Blind Deconvolution. *arXiv preprint*, 2017. `arXiv:1711.10467`.

34 Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. HoloClean: Holistic Data Repairs with Probabilistic Inference. *PVLDB*, 10(11), 2017.

35 Matthew Richardson and Pedro Domingos. Markov Logic Networks. *Mach. Learn.*, 62(1-2):107–136, February 2006.

36 Christopher De Sa, Ihab F. Ilyas, Benny Kimelfeld, Christopher Ré, and Theodoros Rekatsinas. A Formal Framework For Probabilistic Unclean Databases. *CoRR*, abs/1801.06750, 2018. `arXiv:1801.06750`.

37 Christopher De Sa, Christopher Ré, and Kunle Olukotun. Global Convergence of Stochastic Gradient Descent for Some Non-convex Matrix Problems. In *ICML*, volume 37 of *JMLR Proceedings*, pages 2332–2341. JMLR.org, 2015. URL: `http://jmlr.org/proceedings/papers/v37/sa15.html`.

38 Prithviraj Sen, Amol Deshpande, and Lise Getoor. PrDB: managing and exploiting rich correlations in probabilistic databases. *VLDB J.*, 18(5):1065–1090, 2009.

39 Sameer Singh, Michael Wick, and Andrew McCallum. Monte Carlo MCMC: Efficient Inference by Approximate Sampling. In *MNLP-CoNLL*, pages 1104–1113. Association for Computational Linguistics, 2012.

40 Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Morgan & Claypool Publishers, 1st edition, 2011.

41 Martin J. Wainwright, Tommi S. Jaakkola, and Alan S. Willsky. Tree-reweighted belief propagation algorithms and approximate ML estimation via pseudo-moment matching. In *AISTATS*, January 2003.

42 Jiannan Wang and Nan Tang. Towards dependable data repairing with fixing rules. In *SIGMOD*, pages 457–468. ACM, 2014.

43 Mohamed Yakout, Ahmed K Elmagarmid, Jennifer Neville, Mourad Ouzzani, and Ihab F Ilyas. Guided data repair. *PVLDB*, 4(5):279–289, 2011.

On the Expressive Power of Linear Algebra on Graphs

Floris Geerts
University of Antwerp, Antwerp, Belgium
http://adrem.uantwerpen.be/floris.geerts
floris.geerts@uantwerpen.be

Abstract

Most graph query languages are rooted in logic. By contrast, in this paper we consider graph query languages rooted in linear algebra. More specifically, we consider MATLANG, a matrix query language recently introduced, in which some basic linear algebra functionality is supported. We investigate the problem of characterising equivalence of graphs, represented by their adjacency matrices, for various fragments of MATLANG. A complete picture is painted of the impact of the linear algebra operations in MATLANG on their ability to distinguish graphs.

2012 ACM Subject Classification Information systems → Query languages; Mathematics of computing → Graph theory; Theory of computation → Logic

Keywords and phrases matrix query languages, graph queries, graph theory, logics

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.7

Related Version Full version: https://arxiv.org/abs/1812.04379.

1 Introduction

Motivated by the importance of linear algebra for machine learning on big data [7, 8, 12, 46, 51] there is a current interest in languages that combine matrix operations with relational query languages in database systems [22, 35, 40, 41, 43]. Such hybrid languages raise many interesting questions from a database theoretical point of view. It seems natural, however, to first consider query languages for matrices alone. These are the focus of this paper.

More precisely, we continue the investigation of the expressive power of the matrix query language MATLANG, recently introduced as an analog for matrices of the relational algebra on relations [9]. Intuitively, queries in MATLANG are built-up by composing several linear algebra operations. The language MATLANG was shown to be subsumed by aggregate logic with only three non-numerical variables. Conversely, MATLANG can express all queries from graph databases to binary relations that can be expressed in first-order logic with three variables. The four-variable query asking if a graph contains a four-clique, however, is not expressible [9].

In this paper, we further zoom in on the expressive power of MATLANG on graphs. In particular, we investigate when two graphs are *equivalent* relative to some fragment of MATLANG. These fragments are defined by allowing only certain linear algebra operations in the queries and are denoted by $\mathsf{ML}(\mathcal{L})$, with $\mathcal{L}$ the list of allowed operations. A total of six (sensible) fragments are considered and $\mathsf{ML}(\mathcal{L})$-equivalence of graphs, i.e., their agreement on all sentences in $\mathsf{ML}(\mathcal{L})$, is characterised. Our results are as follows.

- For starters, we have the fragment $\mathsf{ML}(\cdot, \mathsf{tr})$ that allows for matrix multiplication $(\cdot)$ and trace (tr) computation (i.e., taking the sum of diagonal elements of a matrix). Equivalence of graphs relative to $\mathsf{ML}(\cdot, \mathsf{tr})$ coincides with being co-spectral, or equivalently, to having the same number of closed walks of any length (Section 5).

- Another fragment, $\mathsf{ML}(\cdot, {}^*, \mathbb{1})$, allows for matrix multiplication, conjugate transposition (*) and the introduction of the vector $\mathbb{1}$, consisting of all ones. Here, equivalence coincides with having the same number of (not necessarily closed) walks of any length (Section 6).
- When allowing both tr and $\mathbb{1}$, equivalence relative to $\mathsf{ML}(\cdot, \mathsf{tr}, \mathbb{1})$ coincides, not surprisingly, to having the same number of closed and non-closed walks of any length (Section 6).
- More interesting is the fragment $\mathsf{ML}(\cdot, {}^*, \mathbb{1}, \mathsf{diag})$, which also allows for the operation $\mathsf{diag}(\cdot)$ that turns a vector into a diagonal matrix with that vector on its diagonal. In this case, equivalence coincides with having a so-called common equitable partition, or equivalently, to C^2-equivalence. Here, C^2 denotes the two-variable fragment of C, the extension of first-order logic with counting (Section 7).
- The combination of tr with diag results in a stronger notion of equivalence: Graphs are equivalent relative to $\mathsf{ML}(\cdot, \mathsf{tr}, \mathbb{1}, \mathsf{diag})$ when they are C^2-equivalent *and* co-spectral (Section 7).
- Finally, equivalence relative to $\mathsf{MATLANG}$ is shown to correspond to C^3-equivalence, the three-variable fragment of C (Section 8). This is in agreement with the results from Brijder et al. [9] mentioned earlier.

We remark that each of these fragments can be extended with addition and scalar multiplication at no increase in distinguishing power. We exhibit examples *separating* all fragments.

The characterisations are shown in a purely algebraic way, without relying on simulations in logic. Underlying are reductions of $\mathsf{ML}(\mathcal{L})$-equivalence of graphs to *similarity notions* of their adjacency matrices. For example, it is known that two graphs G and H are C^2-equivalent if and only if they are fractionally isomorphic [48, 53, 54]. This means that the adjacency matrices A_G of G and A_H of H satisfy $A_G \cdot S = S \cdot A_H$ for some doubly stochastic matrix S. As another example, C^3-equivalence of graphs corresponds to $A_G \cdot O = O \cdot A_H$ for some orthogonal matrix O that is also an isomorphism between the cellular algebras of G and H [20]. We provide similar characterisations for all our matrix query language fragments. It is worth pointing out that beyond $\mathsf{MATLANG}$, C^k-equivalence, for $k \geq 4$, can also be characterised in terms of solutions to linear problems [3, 29, 44].

Moreover, whenever possible, we also provide characterisations in terms of *spectral properties* of graphs. A wealth of results exists in spectral graph theory on what information can be obtained from the adjacency matrix, or from other matrices like the Laplacian, of a graph [10, 16, 26]. We rely quite a bit on known results in that area. Nevertheless, we believe that the connections made in this paper are of interest in their own right. They relate combinatorial and spectral graph invariants by means of query languages. We refer to work by Fürer [24, 25] for more examples of the power of graph invariants and to Dawar et al. [20] for connections between logic, combinatorial and spectral invariants.

Although links to logics such as C^2 and C^3 are made, the connection between $\mathsf{MATLANG}$, rank logics and fixed-point logics with counting, as studied in the context of the descriptive complexity of linear algebra [18, 17, 19, 27, 30, 34], is yet to be explored. Similarly for connections to logic-based graph query languages [2, 5].

2 Background

We denote the set of real numbers by $\mathbb{R}$ and the set of complex numbers by $\mathbb{C}$. The set of $m \times n$-matrices over the real (resp., complex) numbers is denoted by $\mathbb{R}^{m \times n}$ (resp., $\mathbb{C}^{m \times n}$). Vectors are elements of $\mathbb{R}^{m \times 1}$ (or $\mathbb{C}^{m \times 1}$). The entries of an $m \times n$-matrix A are denoted by A_{ij}, for $i = 1, \ldots, m$ and $j = 1, \ldots, n$. The entries of a vector v are denoted by v_i, for $i = 1, \ldots, m$. We often identify $\mathbb{R}^{1 \times 1}$ with $\mathbb{R}$, and $\mathbb{C}^{1 \times 1}$ with $\mathbb{C}$. The following classes of

matrices are of interest in this paper: square matrices (elements in $\mathbb{R}^{n\times n}$ or $\mathbb{C}^{n\times n}$), symmetric matrices (such that $A_{ij} = A_{ji}$ for all i and j), *doubly stochastic* matrices ($A_{ij} \in \mathbb{R}$, $A_{ij} \geq 0$, $\sum_{j=1}^{n} A_{ij} = 1$ and $\sum_{i=1}^{m} A_{ij} = 1$ for all i and j), *doubly quasi-stochastic* matrices ($A_{ij} \in \mathbb{R}$, $\sum_{j=1}^{n} A_{ij} = 1$ and $\sum_{i=1}^{m} A_{ij} = 1$ for all i and j), and *orthogonal* matrices ($O \in \mathbb{R}^{n\times n}$, $O^{\mathsf{t}} \cdot O = I = O \cdot O^{\mathsf{t}}$, where O^{t} denotes the transpose of O obtained by switching rows and columns, $\cdot$ denotes matrix multiplication, and I is the identity matrix in $\mathbb{R}^{n\times n}$).

We only need a couple of notions of linear algebra. We refer to the textbook by Axler [4] for more background. An *eigenvalue* of a matrix A is a scalar λ in $\mathbb{C}$ for which there is a non-zero vector v satisfying $A \cdot v = \lambda v$. Such a vector is called an *eigenvector* of A for eigenvalue λ. The *eigenspace* of an eigenvalue is the vector space obtained as the span of a maximal set of linear independent eigenvectors for this eigenvalue. Here, the *span* of a set of vectors just refers to the set of all linear combinations of vectors in that set. A set of vectors is linear independent if no vector in that set can be written as a linear combination of other vectors. The *dimension* of an eigenspace is the minimal number of eigenvectors that span the eigenspace.

We will only consider undirected graphs without self-loops. Let $G = (V, E)$ be such a graph with vertices $V = \{1, \ldots, n\}$ and unordered edges $E \subseteq \{\{i, j\} \mid i, j \in V\}$. The *order* of G is simply the number of vertices. Then, the *adjacency matrix* of a graph G of order n, denoted by A_G, is an $n \times n$-matrix whose entries $(A_G)_{ij}$ are set to 1 if and only if $\{i, j\} \in E$, all other entries are set to 0. The matrix A_G is a symmetric real matrix with zeroes on its diagonal. The *spectrum* of an undirected graph can be represented as $\mathsf{spec}(G) = \begin{pmatrix} \lambda_1 & \lambda_2 & \cdots & \lambda_p \\ m_1 & m_2 & \cdots & m_p \end{pmatrix}$, where $\lambda_1 < \lambda_2 < \cdots < \lambda_p$ are the distinct real eigenvalues of the adjacency matrix A_G of G, and where $m_1, m_2, \ldots, m_p$ denote the dimensions of the corresponding eigenspaces. Two graphs are said to be *co-spectral* if they have the same spectrum. We introduce other relevant notions throughout the paper.

3 Matrix Query Languages

As described in Brijder et al. [9], matrix query languages can be formalised as compositions of linear algebra operations. Intuitively, a linear algebra operation takes a number of matrices as input and returns another matrix. Examples of operations are matrix multiplication, conjugate transposition, computing the trace, just to name a few. By closing such operations under composition "matrix query languages" are formed. More specifically, for linear algebra operations $\mathsf{op}_1, \ldots, \mathsf{op}_k$ the corresponding matrix query language is denoted by $\mathsf{ML}(\mathsf{op}_1, \ldots, \mathsf{op}_k)$ and consists of expressions formed by the following grammar:

$$e := X \mid \mathsf{op}_1(e_1, \ldots, e_{p_1}) \mid \cdots \mid \mathsf{op}_k(e_1, \ldots, e_{p_k}),$$

where X denotes a *matrix variable* which serves to indicate the input to expressions and p_i denotes the number of inputs required by operation op_i. We focus on the case when only a single matrix variable X is present. The treatment of multiple variables is left for future work.

The semantics of an expression $e(X)$ in $\mathsf{ML}(\mathsf{op}_1, \ldots, \mathsf{op}_k)$ is defined inductively, relative to an *assignment* ν of X to a matrix $\nu(X) \in \mathbb{C}^{m\times n}$, for some dimensions m and n. We denote by $e\big(\nu(X)\big)$ the result of evaluating $e(X)$ on $\nu(X)$. As expected, we define $\mathsf{op}_i(e_1(X), \ldots, e_{p_i}(X))(\nu(X)) := \mathsf{op}_i\big(e_1(\nu(X)), \ldots, e_{p_i}(\nu(X))\big)$ for linear algebra operation op_i. In Table 1 we list the operations constituting the basis matrix query language MATLANG, introduced in Brijder et al. [9]. In the table we also show their semantics. We note that

Table 1 Linear algebra operations (supported in MATLANG [9]) and their semantics. In the first operation, for $A_{ji} \in \mathbb{C}$, A^*_{ji} denotes complex conjugation. In the last operation, $\Omega = \bigcup_{k>0} \Omega_k$, where Ω_k consists of functions $f : \mathbb{C}^k \to \mathbb{C}$.

conjugate transposition $(\mathsf{op}(e) = e^*)$		
$e(\nu(X)) = A \in \mathbb{C}^{m\times n}$	$e(\nu(X))^* = A^* \in \mathbb{C}^{n\times m}$	$(A^*)_{ij} = A^*_{ji}$
one-vector $(\mathsf{op}(e) = \mathbb{1}(e))$		
$e(\nu(X)) = A \in \mathbb{C}^{m\times n}$	$\mathbb{1}(e(\nu(X)) = \mathbb{1} \in \mathbb{C}^{m\times 1}$	$\mathbb{1}_i = 1$
diagonalization of a vector $(\mathsf{op}(e) = \mathsf{diag}(e))$		
$e(\nu(X)) = A \in \mathbb{C}^{m\times 1}$	$\mathsf{diag}(e(\nu(X)) = \mathsf{diag}(A) \in \mathbb{C}^{m\times m}$	$\mathsf{diag}(A)_{ii} = A_i$, $\mathsf{diag}(A)_{ij} = 0,\ i \neq j$
matrix multiplication $(\mathsf{op}(e_1, e_2) = e_1 \cdot e_2)$		
$e_1(\nu(X)) = A \in \mathbb{C}^{m\times n}$, $e_2(\nu(X)) = B \in \mathbb{C}^{m\times o}$	$e_1(\nu(X)) \cdot e_2(\nu(X)) = C \in \mathbb{C}^{m\times o}$	$C_{ij} = \sum_{k=1}^{n} A_{ik} \times B_{kj}$
matrix addition $(\mathsf{op}(e_1, e_2) = e_1 + e_2)$		
$e_i(\nu(X)) = A^{(i)} \in \mathbb{C}^{m\times n}$	$e_1(\nu)(X) + e_2(\nu(X)) = B \in \mathbb{C}^{m\times n}$	$B_{ij} = A^{(1)}_{ij} + A^{(2)}_{ij}$
scalar multiplication $(\mathsf{op}(e) = c \times e,\ c \in \mathbb{C})$		
$e(\nu(X)) = A \in \mathbb{C}^{m\times n}$	$c \times e(\nu(X)) = B \in \mathbb{C}^{m\times n}$	$B_{ij} = c \times A_{ij}$
trace $(\mathsf{op}(e) = \mathsf{tr}(e))$		
$e(\nu(X)) = A \in \mathbb{C}^{m\times m}$	$\mathsf{tr}(e(\nu(X)) = c \in \mathbb{C}$	$c = \sum_{i=1}^{m} A_{ii}$
$e(\nu(X)) = A \in \mathbb{C}^{m\times 1}$	$\mathsf{tr}(e(\nu(X)) = c \in \mathbb{C}$	$c = \sum_{i=1}^{m} A_i$
pointwise function application $(\mathsf{op}(e_1, \ldots, e_p) = \mathsf{apply}[f](e_1, \ldots, e_p)),\ f : \mathbb{C}^p \to \mathbb{C} \in \Omega$		
$e_i(\nu(X)) = A^{(i)} \in \mathbb{C}^{m\times n}$	$\mathsf{apply}[f]\big(e_1(\nu(X)), \ldots, e_p(\nu(X))\big) = B \in \mathbb{C}^{m\times n}$	$B_{ij} = f(A^{(1)}_{ij}, \ldots, A^{(p)}_{ij})$

restrictions on the dimensions are in place to ensure that operations are well-defined. Using a simple type system one can formalise a notion of well-formed expressions which guarantees that the semantics of such expressions is well-defined [9]. We only consider well-formed expressions from here on.

▶ Remark 3.1. The list of operations in Table 1 differs slightly from the list presented in Brijder et al. [9]: We explicitly mention scalar multiplication ($\times$) and addition ($+$), and the trace operation (tr), all of which can be expressed in MATLANG. Hence, MATLANG and $\mathsf{ML}(\cdot, {}^*, \mathsf{tr}, \mathbb{1}, \mathsf{diag}, +, \times, \mathsf{apply}[f], f \in \Omega)$ are equivalent.

4 Expressive Power

As mentioned in the introduction, we are interested in the expressive power of matrix query languages. In this paper, we consider sentences in these languages. We define an expression $e(X)$ in $\mathsf{ML}(\mathsf{op}_1, \ldots, \mathsf{op}_k)$ to be a *sentence* if $e(\nu(X))$ returns a 1×1-matrix for any assignment ν of X. We note that the type system of MATLANG allows to check whether an expression in $\mathsf{ML}(\mathcal{L})$ is a sentence (see Brijder et al. [9] for more details). Having defined sentences, a notion of equivalence naturally follows.

▶ **Definition 4.1.** *Two matrices A and B in $\mathbb{C}^{m\times n}$ are said to be* $\mathsf{ML}(\mathsf{op}_1, \ldots, \mathsf{op}_k)$-equivalent, *denoted by $A \equiv_{\mathsf{ML}(\mathsf{op}_1,\ldots,\mathsf{op}_k)} B$, if and only if $e(A) = e(B)$ for all sentences $e(X)$ in* $\mathsf{ML}(\mathsf{op}_1, \ldots, \mathsf{op}_k)$.

In other words, equivalent matrices cannot be distinguished by sentences in the matrix query language under consideration. Equivalence with regards to sentences resembles the standard notion of equivalence used in logic.

We aim to *characterise* equivalence for various matrix query languages. We will, however, not treat this problem in full generality and instead, to gain intuition, start by considering *adjacency matrices of undirected graphs*. The corresponding notion of equivalence on graphs is defined, as expected:

▶ **Definition 4.2.** *Two graphs G and H of the same order are said to be* $\mathsf{ML}(\mathsf{op}_1, \ldots, \mathsf{op}_k)$-equivalent*, denoted by* $G \equiv_{\mathsf{ML}(\mathsf{op}_1,\ldots,\mathsf{op}_k)} H$*, if and only if their adjacency matrices are* $\mathsf{ML}(\mathsf{op}_1, \ldots, \mathsf{op}_k)$*-equivalent.*

▶ Remark 4.3. One could imagine defining equivalence with regards to arbitrary expressions, i.e., expressions in MATLANG that are not necessarily sentences. Such a notion would be too strong, however. Indeed, requiring that $e(A_G) = e(A_H)$ for arbitrary expressions $e(X)$ would imply that $A_G = A_H$ and hence $G = H$.

In the following sections we consider graph equivalence for various fragments, starting from simple fragments only supporting a couple of operations, up to the full MATLANG matrix query language.

5 Expressive Power of the Matrix Query Language $\mathsf{ML}(\cdot, \mathsf{tr})$

The smallest fragment we consider is $\mathsf{ML}(\cdot, \mathsf{tr})$. This is a very restrictive fragment since the only sentences that one can express are of the form (i) $\#\mathsf{cwalk}_k(X) := \mathsf{tr}(X^k)$, where X^k stands for the kth power of X, i.e., X multiplied k times with itself, and (ii) products of such sentences. We note that, when evaluated on an adjacency matrix A_G, $\#\mathsf{cwalk}_k(A_G)$ counts the number of *closed walks of length k* in G.

Indeed, the entries of the powers A_G^k of adjacency matrix A_G are known to correspond to the number of walks of length k in G. Recall that a *walk of length k* in a graph $G = (V, E)$ is a sequence $(v_0, v_1, \ldots, v_k)$ of vertices of G such that consecutive vertices are adjacent in G, i.e., $\{v_{i-1}, v_i\} \in E$ for all $i = 1, \ldots, k$. Furthermore, a *closed walk* is a walk that starts in and ends at the same vertex. Hence, $\#\mathsf{cwalk}_k(A_G) = \sum_i (A_G^k)_{ii}$ indeed counts closed walks of length k in G. Closed walks of length 0 correspond, as usual, to vertices in G.

The following characterisations are known to hold.

▶ **Proposition 5.1** ([10, 16])**.** *Let G and H be two graphs of the same order. The following are equivalent:*

- *G and H have the same total number of closed walks of length k, for all $k \geq 0$,*
- *$\mathsf{tr}(A_G^k) = \mathsf{tr}(A_H^k)$ for all $k \geq 0$,*
- *G and H are co-spectral, and*
- *there exists a real orthogonal matrix O such that $A_G \cdot O = O \cdot A_H$.*

▶ **Example 5.2.** The graphs G_1 (⚀) and H_1 (⚅) are the smallest pair (in terms of number of vertices) of non-isomorphic co-spectral graphs of the same order [13]. Note that the isolated vertex in G_1 ensures that G_1 and H_1 have the same number of vertices (and thus the same number of closed walks of length 0).
A characterisation of $\mathsf{ML}(\cdot, \mathsf{tr})$-equivalence now easily follows.

▶ **Proposition 5.3.** *For two graphs G and H of the same order, $G \equiv_{\mathsf{ML}(\cdot,\mathsf{tr})} H$ if and only if there exists a real orthogonal matrix O such that $A_G \cdot O = O \cdot A_H$ if and only if G and H have the same number of closed walks of any length.*

Proof. By definition, if $G \equiv_{\mathsf{ML}(\cdot,\mathsf{tr})} H$, then $e(A_G) = e(A_H)$ for any sentence $e(X)$ in $\mathsf{ML}(\cdot, \mathsf{tr})$. This holds in particular for the sentences $\#\mathsf{cwalk}_k(X) := \mathsf{tr}(X^k)$ in $\mathsf{ML}(\cdot, \mathsf{tr})$, for

$k \geq 1$. That is, $G \equiv_{\mathsf{ML}(\cdot,\mathsf{tr})} H$ implies that $\mathsf{tr}(A_G^k) = \mathsf{tr}(A_H^k)$ for all $k \geq 1$. Since G and H are of the same order and $A_G^0 = A_H^0 = I$ (by convention), $\mathsf{tr}(A_G^0) = \mathsf{tr}(A_H^0) = \mathsf{tr}(I) = n$. From the previous proposition it then follows that there exists an orthogonal matrix O such that $A_G \cdot O = O \cdot A_H$.

For the converse, assume that $A_G \cdot O = O \cdot A_H$ for some orthogonal matrix O. We already remarked that sentences in $\mathsf{ML}(\cdot,\mathsf{tr})$ are products of sentences of the form $\#\mathsf{cwalk}_k(X) := \mathsf{tr}(X^k)$. It now suffices to observe that $\mathsf{tr}(P \cdot A \cdot P^{-1}) = \mathsf{tr}(A)$ for any matrix A and any invertible matrix P. In particular, $\mathsf{tr}(A_G^k) = \mathsf{tr}(O^\mathsf{t} \cdot A_G^k \cdot O) = \mathsf{tr}(O^\mathsf{t} \cdot O \cdot A_H^k) = \mathsf{tr}(A_H^k)$. ◀

From an expressiveness point of view, it tells that $\mathsf{ML}(\cdot,\mathsf{tr})$-equivalence of two graphs implies that their adjacency matrices share the same rank, characteristic polynomial, determinant, eigenvalues, and their algebraic multiplicities, geometric multiplicities of eigenvalues, just to name a few.

Given that the trace operation is a linear mapping, i.e., $\mathsf{tr}(cA + dB) = c\mathsf{tr}(A) + d\mathsf{tr}(B)$ for matrices A and B and complex numbers c and d, one would expect that matrix addition $(+)$ and scalar multiplication $(\times)$ can be added to $\mathsf{ML}(\cdot,\mathsf{tr})$ without an increase in expressiveness. Indeed, one can rewrite sentences in $\mathsf{ML}(\cdot,\mathsf{tr},+,\times)$ as a linear combination of sentences in $\mathsf{ML}(\cdot,\mathsf{tr})$. Combined with the linearity of $\mathsf{tr}(\cdot)$, Proposition 5.3 can be extended as follows.

▶ **Corollary 5.4.** *For two graphs G and H of the same order, we have that $G \equiv_{\mathsf{ML}(\cdot,\mathsf{tr})} H$ if and only if $G \equiv_{\mathsf{ML}(\cdot,\mathsf{tr},+,\times)} H$.*

We can further strengthen Corollary 5.4 by allowing the application of *any* function $f : \mathbb{C}^p \to \mathbb{C}$ in Ω, *provided* that $\mathsf{apply}[f](e_1, \ldots, e_p)$ is only allowed when each e_i is a sentence. That is, we only allow pointwise function applications on "scalars". The restriction of such function applications is denoted by $\mathsf{apply}_\mathsf{s}[f]$, for $f \in \Omega$. Indeed, $G \equiv_{\mathsf{ML}(\cdot,\mathsf{tr},+,\times)} H$ implies that $e(A_G) = e(A_H)$ for any sentence $e(X)$ in $\mathsf{ML}(\cdot,\mathsf{tr},+,\times)$. Clearly, when $e_i(A_G) = e_i(A_H)$ for all $i = 1, \ldots, p$, $\mathsf{apply}_\mathsf{s}[f](e_1(A_G), \ldots, e_p(A_G)) = \mathsf{apply}_\mathsf{s}[f](e_1(A_H), \ldots, e_p(A_H))$.

▶ **Corollary 5.5.** *For two graphs G and H of the same order, $G \equiv_{\mathsf{ML}(\cdot,\mathsf{tr},+,\times)} H$ if and only if $G \equiv_{\mathsf{ML}(\cdot,\mathsf{tr},+,\times,\mathsf{apply}_\mathsf{s}[f],f\in\Omega)} H$.*

Finally, we can also add conjugate transposition $(^*)$ without increasing the expressive power, *provided* that we mildly restrict the class Ω of pointwise functions. More precisely, we assume that Ω is *closed under complex conjugation* in the sense that for every $f \in \Omega$ also the composition * and f is in Ω. This assumption, together with standard properties of complex conjugation and conjugate transposition (in particular, $(A \cdot B)^* = B^* \cdot A^*$, $(A^*)^* = A$ and linearity) and using the fact that adjacency matrices of undirected graphs are symmetric, allows one to rewrite expressions in $\mathsf{ML}(\cdot,{}^*,\mathsf{tr},+,\times,\mathsf{apply}_\mathsf{s}[f], f \in \Omega)$ such that * is only applied on scalars. As a consequence, any expression in $\mathsf{ML}(\cdot,{}^*,\mathsf{tr},+,\times,\mathsf{apply}_\mathsf{s}[f], f \in \Omega)$ is equivalent to an expression in $\mathsf{ML}(\cdot,\mathsf{tr},+,\times,\mathsf{apply}_\mathsf{s}[f], f \in \Omega)$.

▶ **Corollary 5.6.** *Let Ω be a class of pointwise functions that is closed under complex conjugation. Then, for two graphs G and H of the same order, $G \equiv_{\mathsf{ML}(\cdot,\mathsf{tr},+,\times,\mathsf{apply}_\mathsf{s}[f],f\in\Omega)} H$ if and only if $G \equiv_{\mathsf{ML}(\cdot,{}^*,\mathsf{tr},+,\times,\mathsf{apply}_\mathsf{s}[f],f\in\Omega)} H$.*

As a consequence, the graphs G_1 (⊡) and H_1 (⨯) from Example 5.2 cannot be distinguished by sentences in $\mathsf{ML}(\cdot,{}^*,\mathsf{tr},+,\times,\mathsf{apply}_\mathsf{s}[f], f \in \Omega)$.

As we will see later, including any other operation from Table 1, such as $\mathbb{1}(\cdot)$, $\mathsf{diag}(\cdot)$ or pointwise function applications on vector or matrices, requires additional constraints on the orthogonal matrix O linking A_G with A_H.

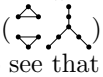
▶ Remark 5.7. Corollaries 5.4, 5.5 and 5.6 hold for *any* fragment that we will consider.

6 The Impact of the $\mathbb{1}(\cdot)$ Operation

The $\mathbb{1}(\cdot)$ operation, which returns the all-ones vector $\mathbb{1}$[1], allows to extract other information from graphs than just the number of closed walks. Indeed, consider the sentences

$$\#\mathsf{walk}_k(X) := (\mathbb{1}(X))^* \cdot X^k \cdot \mathbb{1}(X) \text{ and } \#\mathsf{walk}'_k(X) := \mathsf{tr}(X^k \cdot \mathbb{1}(X)),$$

in $\mathsf{ML}(\cdot, {}^*, \mathbb{1})$ and $\mathsf{ML}(\cdot, \mathsf{tr}, \mathbb{1})$, respectively. When applied on adjacency matrix A_G of a graph G, $\#\mathsf{walk}_k(A_G)$ (and also $\#\mathsf{walk}'_k(A_G)$) returns the *number of (not necessarily closed) walks* in G of length k. In relation to the previous section, co-spectral graphs do not necessarily have the same number of walks of any length. Similarly, graphs with the same number of walks of any length are not necessarily co-spectral.

▶ **Example 6.1.** It can be verified that the co-spectral graphs G_1 () and H_1 () of Example 5.2 have 16 versus 20 walks of length 2, respectively. As a consequence, $\mathsf{ML}(\cdot, {}^*, \mathbb{1})$ and $\mathsf{ML}(\cdot, \mathsf{tr}, \mathbb{1})$ can distinguish G_1 from H_1 by means of the sentences $\#\mathsf{walk}_2(X)$ and $\#\mathsf{walk}'_2(X)$, respectively. By contrast, the graphs G_2 () and H_2 () are not co-spectral, yet have the same number of walks of any length. It is easy to see that G_2 and H_2 are not co-spectral (apart from verifying that their spectra are different): H_2 has 12 closed walks of length 3 (because of the triangles), whereas G_2 has none. We argue below why they have the same number of walks. As a consequence, $\mathsf{ML}(\cdot, \mathsf{tr})$ (and thus also $\mathsf{ML}(\cdot, \mathsf{tr}, \mathbb{1})$) can distinguish G_2 and H_2. It follows from Proposition 6.6 below that these graphs cannot be distinguished by $\mathsf{ML}(\cdot, {}^*, \mathbb{1})$.

Graphs sharing the same number of walks of any length have been investigated before in spectral graph theory [14, 15, 32, 49]. To state a spectral characterisation, the so-called *main spectrum* of a graph needs to be considered. The main spectrum of a graph is the set of eigenvalues whose eigenspace is not orthogonal to the $\mathbb{1}$ vector. More formally, for an eigenvalue λ and corresponding eigenspace, represented by a matrix V whose columns are eigenvectors of λ that span its eigenspace, the *main angle* β_λ of λ's eigenspace is $\frac{1}{\sqrt{n}}\|V^\mathsf{t} \cdot \mathbb{1}\|_2$, where $\|\cdot\|_2$ is the Euclidean norm. Hence, main eigenvalues are those with a non-zero main angle. Two graphs are said to be *co-main* if they have the same set of main eigenvalues and corresponding main angles. Intuitively, the importance of the orthogonal projection on $\mathbb{1}$ stems from the observation that $\#\mathsf{walk}_k(A_G)$ can be expressed as $\sum_i \lambda_i^k \beta_{\lambda_i}^2$ where the λ_i's are eigenvalues of A_G. Clearly, only those eigenvalues λ_i for which $\beta_{\lambda_i} > 0$ matter when computing $\#\mathsf{walk}_k(A_G)$. This results in the following characterisation.

▶ **Proposition 6.2** (Theorem 1.3.5 in Cvetković et al. [16])**.** *Two graphs G and H of the same order are co-main if and only if they have the same total number of walks of length k, for every $k \geq 0$.*

Furthermore, the following proposition follows implicitly from the proof of Theorem 3 in van Dam et al. [56] (and is also shown in Theorem 1.2 in Dell et al. [21] in the context of distinguishing graphs by means of *homomorphism vectors*).

▶ **Proposition 6.3.** *Two graphs G and H of the same order have the same total number of walks of length k, for every $k \geq 0$, if and only if there is a doubly quasi-stochastic matrix Q such that $A_G \cdot Q = Q \cdot A_H$, i.e., such that $A_G \cdot Q = Q \cdot A_H$, $Q \cdot \mathbb{1} = \mathbb{1}$ and $Q^\mathsf{t} \cdot \mathbb{1} = \mathbb{1}$ hold.*

[1] We use $\mathbb{1}$ to denote the all ones *vector* (of appropriate dimension) and use $\mathbb{1}(\cdot)$ (with brackets) for the corresponding all ones *operator*.

▶ **Example 6.4** (Continuation of Example 6.1). Consider the subgraph G_3 () of G_2 and the subgraph H_3 () of H_2. We have that $A_{G_3} \cdot Q = Q \cdot A_{H_3}$ for

$$A_{G_3} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}, A_{H_3} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}, Q = \begin{bmatrix} 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 \end{bmatrix},$$

and hence by Proposition 6.3, G_3 and H_3 have the same number of walks on any length.

As it turns out, the values of the sentences $\#\mathsf{walk}_k(A_G)$ mentioned earlier, that count the number of walks of length k in G, fully determine the value of any sentence in $\mathsf{ML}(\cdot, {}^*, \mathbb{1})$.

▶ **Lemma 6.5.** *Let G and H be two graphs of the same order. Then, $G \equiv_{\mathsf{ML}(\cdot,{}^*,\mathbb{1})} H$ if and only if $\#\mathsf{walk}_k(A_G) = \#\mathsf{walk}_k(A_H)$ for all $k \geq 1$.*

The proof involves an analysis of expressions in $\mathsf{ML}(\cdot, {}^*, \mathbb{1})$. We may thus conclude from Proposition 6.3 and Lemma 6.5 that:

▶ **Proposition 6.6.** *For two graphs G and H of the same order, $G \equiv_{\mathsf{ML}(\cdot,{}^*,\mathbb{1})} H$ if and only if there exists a doubly quasi-stochastic matrix Q such that $A_G \cdot Q = Q \cdot A_H$ if and only if G and H have the same number of walks of any length.*

When it comes to $\mathsf{ML}(\cdot, \mathsf{tr}, \mathbb{1})$, we know from Propositions 5.1 and 5.3 that $G \equiv_{\mathsf{ML}(\cdot,\mathsf{tr},\mathbb{1})} H$ implies that G and H are co-spectral. Combined with Proposition 6.2 and the fact that the sentence $\#\mathsf{walk}'_k(X)$ counts the number of walks of length k, we have that $G \equiv_{\mathsf{ML}(\cdot,\mathsf{tr},\mathbb{1})} H$ implies that G and H are co-spectral and co-main. The following is known about such graphs.

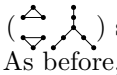
▶ **Proposition 6.7** ([38, 56]). *Two co-spectral graphs G and H of the same order are co-main if and only if there exists an orthogonal matrix O such that $A_G \cdot O = O \cdot A_H$ and $O \cdot \mathbb{1} = \mathbb{1}$.*

In other words, $G \equiv_{\mathsf{ML}(\cdot,\mathsf{tr},\mathbb{1})} H$ implies the existence of an orthogonal matrix O such that $O \cdot \mathbb{1} = \mathbb{1}$ (i.e., O is also doubly quasi-stochastic) and $A_G \cdot O = O \cdot A_H$. An analysis of expressions in $\mathsf{ML}(\cdot, \mathsf{tr}, \mathbb{1})$ shows that the converse also holds.

▶ **Proposition 6.8.** *For two graphs G and H of the same order, $G \equiv_{\mathsf{ML}(\cdot,\mathsf{tr},\mathbb{1})} H$ if and only if there exists an orthogonal matrix O such that $A_G \cdot O = O \cdot A_H$ and $O \cdot \mathbb{1} = \mathbb{1}$ if and only if G and H have the same number of closed walks and the same number of walks of any length.*

An alternative characterisation (also in van Dam et al. [56]) is that G and H are co-spectral and co-main if and only if both G and H *and* their complement graphs $\bar{G}$ and $\bar{H}$ are co-spectral. Here, the complement graph $\bar{G}$ of G is the graph with adjacency matrix given by $J - A_G - I$, where J is the all ones matrix, and similarly for $\bar{H}$.

▶ **Example 6.9** (Continuation of Example 6.1). Consider the subgraph G_4 () of G_2 and the subgraph H_4 () of H_2. These are known to be the smallest non-isomorphic co-spectral graphs with co-spectral complements [31]. From Proposition 6.8 it then follows that G_4 and H_4 have the same number of walks of any length. Combined with our earlier observation in Example 6.4 that also G_3 and H_3 have this property, we may conclude that $G_2 = G_3 \cup G_4$ () and $H_2 = H_3 \cup H_4$ () have the same number of walks of any length, as anticipated in Example 6.1.

We remark that as a consequence of Propositions 6.6 and 6.8, $G \equiv_{\mathsf{ML}(\cdot,\mathsf{tr},\mathbb{1})} H$ implies that $G \equiv_{\mathsf{ML}(\cdot,{}^*,\mathbb{1})} H$. We already mentioned in Example 6.1 that the graphs G_2 () and H_2 () show that the converse does not hold.

As before, we observe that addition, scalar multiplication, conjugate transposition and pointwise function application on scalars can be included at no increase in expressiveness.

▶ **Corollary 6.10.** *Let G and H be two graphs of the same order. Then,*

- $G \equiv_{\mathsf{ML}(\cdot,{}^*,\mathbb{1},+,\times,\mathsf{apply}_s[f],f\in\Omega)} H$ *if and only if* $G \equiv_{\mathsf{ML}(\cdot,{}^*,\mathbb{1})} H$, *and*
- $G \equiv_{\mathsf{ML}(\cdot,{}^*,\mathsf{tr},\mathbb{1},+,\times,\mathsf{apply}_s[f],f\in\Omega)} H$ *if and only if* $G \equiv_{\mathsf{ML}(\cdot,\mathsf{tr},\mathbb{1})} H$,

where Ω is assumed to be closed under complex conjugation.

7 The Impact of the diag(·) Operation

We next consider the operation $\mathsf{diag}(\cdot)$ which takes a vector as input and returns a diagonal matrix with the input vector on its diagonal. The smallest fragments in which vectors (and sentences) can be defined are $\mathsf{ML}(\cdot,\mathsf{tr},\mathbb{1})$ and $\mathsf{ML}(\cdot,{}^*,\mathbb{1})$. Therefore, in this section we consider equivalence with regards to $\mathsf{ML}(\cdot,\mathsf{tr},\mathbb{1},\mathsf{diag})$ and $\mathsf{ML}(\cdot,{}^*,\mathbb{1},\mathsf{diag})$.

Using $\mathsf{diag}(\cdot)$ we can again extract new information from graphs.

▶ **Example 7.1.** Consider graphs G_4 () and H_4 (). In G_4 we have vertices of degrees 0 and 2, and in H_4 vertices of degrees 1, 2 and 3. We will count the number of vertices of degree 3. To this aim consider the sentence $\#\mathsf{3degr}(X)$ given by

$$\left(\frac{1}{6}\right) \times \mathbb{1}(X)^* \cdot \big(\mathsf{diag}(X \cdot \mathbb{1}(X)) \cdot \mathsf{diag}(X \cdot \mathbb{1}(X) - \mathbb{1}(X)) \cdot \mathsf{diag}(X \cdot \mathbb{1}(X) - 2 \times \mathbb{1}(X))\big) \cdot \mathbb{1}(X),$$

in which we, for convenience, allow addition and scalar multiplications. Each of the subexpressions $\mathsf{diag}(X \cdot \mathbb{1}(X) - d \times \mathbb{1}(X))$, for $d = 0, 1$ and 2, sets the diagonal entry corresponding to vertex v to 0 when v has degree d. By taking the product of these diagonal matrices, entries that are set to 0 will remain zero in the resulting diagonal matrix. This implies that the only non-zero diagonal entries are those corresponding to vertices of degree different from 0, 1 and 2. In other words, only for vertices of degree 3 the diagonal entries carry a non-zero value, i.e., value $3(3-1)(3-2)$. By appropriately rescaling by the factor $\frac{1}{6} = \frac{1}{3(3-1)(3-2)}$, the diagonal entries for the degree three vertices are set to 1, and then summed up. Hence, $\#\mathsf{3degr}(X)$ indeed counts the number vertices of degree three in G_4 and H_4. Since $\#\mathsf{3degr}(A_{G_4}) = [0] \neq [1] = \#\mathsf{3degr}(A_{H_4})$ we can distinguish these graphs.

The use of the diagonal matrices and their products as in our example sentence $\#\mathsf{3degr}(X)$ can be generalised to obtain information about so-called *iterated degrees* of vertices in graphs, e.g., to identify and/or count vertices that have a number of neighbours each of which have neighbours of specific degrees. Such iterated degree information is closely related to *equitable partitions* of graphs (see e.g., Scheinerman et al. [50]). We phrase our results in terms of such partitions instead of iterated degree sequences.

7.1 Equitable Partitions

Formally, an *equitable partition* $\mathcal{V} = \{V_1, \ldots, V_\ell\}$ of G is partition of the vertex set of G such that for all $i, j = 1, \ldots, \ell$ and $v, v' \in V_i$, $\mathsf{deg}(v, V_j) = \mathsf{deg}(v', V_j)$. Here, $\mathsf{deg}(v, V_j)$ is the number of vertices in V_j that are adjacent to v. In other words, an equitable partition is such that the graph is regular within each part, and is bi-regular between any two different parts.

A graph always has a *trivial* equitable partition: simply treat each vertex as a part by its own. Most interesting is the *coarsest* equitable partition of a graph, i.e., the *unique* equitable partition for which any other equitable partition of the graph is a refinement thereof [50]. Two graphs G and H are said to have a *common* equitable partition if there exists an equitable partition $\mathcal{V} = \{V_1, \ldots, V_\ell\}$ of G and an equitable partition $\mathcal{W} = \{W_1, \ldots, W_\ell\}$ of H such that (a) the sizes of the parts agree, i.e., $|V_i| = |W_i|$ for each $i = 1, \ldots, \ell$, and (b) $\mathsf{deg}(v, V_j) = \mathsf{deg}(w, W_j)$ for any $v \in V_i$ and $w \in W_i$ and any $i, j = 1, \ldots, \ell$. We note that, due to condition (b) the trivial partitions of graphs do not always result in a common equitable partition. In other words, not every two graphs (of the same order) have a common equitable partition. Proposition 7.2 below characterises when they do have a common partition. Equitable partitions naturally arise as the result of the *colour refinement procedure* [6, 28, 57], also known as the 1-dimensional Weisfeiler-Lehman algorithm, used as a subroutine in graph isomorphism solvers. Furthermore, there is a close connection to the study of *fractional isomorphisms* of graphs [50, 53], already mentioned in the introduction. We recall: two graphs G and H are said to be fractional isomorphic if there exists a doubly stochastic matrix S such that $A_G \cdot S = S \cdot A_H$. Furthermore, a logical characterisation of graphs with a common equitable partition exists.

▶ **Proposition 7.2** ([53], [36]). *Let G and H be two graphs of the same order. Then, G and H are fractional isomorphic if and only if G and H have a common equitable partition if and only if $G \equiv_{\mathsf{C}^2} H$.*

▶ **Example 7.3.** The matrix linking the adjacency matrices of G_3 () and H_3 () in Example 6.4 is in fact a doubly stochastic matrix (all its entries are either 0 or $\frac{1}{2}$). Hence, G_3 and H_3 have a common equitable partition, which in this case consists of a single part consisting of all vertices. This generally holds for graphs that are k-regular (meaning, each vertex is adjacent to k vertices) for the same k [48, 50]. By contrast, graphs G_2 () and H_2 () do not have a common equitable partition. Indeed, fractional isomorphic graphs must have the same degree sequence [50], which does not hold for G_2 and H_2. For the same reason, G_1 () and H_1 (), and G_4 () and H_4 () are not fractional isomorphic.

To related equitable partitions to $\mathsf{ML}(\cdot, \mathsf{tr}, \mathbb{1}, \mathsf{diag})$- and $\mathsf{ML}(\cdot, {}^*, \mathbb{1}, \mathsf{diag})$-equivalence, we show that the presence of $\mathsf{diag}(\cdot)$ allows to formulate a number of expressions, denoted by $\mathsf{eqpart}_i(X)$, for $i = 1, \ldots, \ell$, that together extract the *coarsest equitable partition* from a given graph.

In the following, $\mathcal{L}$ can be either $\{\cdot, \mathsf{tr}, \mathbb{1}, \mathsf{diag}\}$ or $\{\cdot, {}^*, \mathbb{1}, \mathsf{diag}\}$. Furthermore, we denote by $\mathcal{L}^+$ the extension of $\mathcal{L}$ with linear combinations (i.e., $+$ and $\times$), pointwise function applications on scalars (i.e., $\mathsf{apply}_\mathsf{s}[f]$, $f \in \Omega$) and conjugate transposition (*). The corresponding matrix query languages are denoted by $\mathsf{ML}(\mathcal{L})$ and $\mathsf{ML}(\mathcal{L}^+)$, respectively.

We start by reducing the problem of $\mathsf{ML}(\mathcal{L}^+)$-equivalence to $\mathsf{ML}(\mathcal{L})$-equivalence.

▶ **Lemma 7.4.** *Let G and H be two graphs of the same order. Then, $G \equiv_{\mathsf{ML}(\mathcal{L})} H$ if and only if $G \equiv_{\mathsf{ML}(\mathcal{L}^+)} H$.*

This lemma is verified by showing that expressions in $\mathsf{ML}(\mathcal{L}^+)$ can be seen as linear combinations of expressions in $\mathsf{ML}(\mathcal{L})$, in an analogous way as in the proof of Corollary 6.10. For example, it is clear that $\#\mathsf{3degr}(X)$ can be written as such a linear combination.

We next relate $G \equiv_{\mathsf{ML}(\mathcal{L}^+)} H$ and common equitable partitions of G and H.

▶ **Proposition 7.5.** *Let G and H be two graphs of the same order. Then, $G \equiv_{\mathsf{ML}(\mathcal{L}^+)} H$ implies that G and H have a common equitable partition.*

Proof. We show that the algorithm $\textsc{cgcr}(A_G)$, described in Kersting et al. [39], which computes the coarsest equitable partition of a graph can be simulated by expressions in $\mathsf{ML}(\mathcal{L}^+)$. To describe a partition $\mathcal{V} = \{V_1, \ldots, V_\ell\}$ of the vertex set of G we use *indicator vectors*. More precisely, we define $\mathbb{1}_{V_i}$ as the $n \times 1$-vector which has a "1" for those entries corresponding to vertices in V_i and has all its other entries set to "0". It is clear that we can also recover partitions from indicator vectors. The simulation of $\textsc{cgcr}(A_G)$ results in a number of expressions, denoted by $\mathsf{eqpart}_i(X)$ for $i = 1, \ldots, \ell$, in $\mathsf{ML}(\mathcal{L}^+)$ that *depend on* G and such that the set $\{\mathsf{eqpart}_i(A_G)\}$ consists of indicator vectors of the coarsest equitable partition of G. Since the algorithm $\textsc{cgcr}(A_G)$ is phrased in linear algebra terms [39], its simulation follows easily. Underlying this simulation is the use of products of diagonal matrices as a means of taking conjunctions of indicator vectors, similar to the propagation of zeroes used in $\#\mathsf{3degr}(X)$.

The expressions $\mathsf{eqpart}_i(X)$ are constructed based on G. Next, using our assumption $G \equiv_{\mathsf{ML}(\mathcal{L}^+)} H$, we show that the vectors $\mathsf{eqpart}_i(A_H)$, for $i = 1, \ldots, \ell$, also correspond to the coarsest equitable partition of H. This is done in a number of steps:

1. We verify that each $\mathsf{eqpart}_i(A_H)$ is also an indicator vector containing the same number of 1's as $\mathsf{eqpart}_i(A_G)$.
2. We verify that any distinct pair of indicator vectors in $\{\mathsf{eqpart}_i(A_H)\}$ have no common entry holding value "1". This implies that the set $\{\mathsf{eqpart}_i(A_H)\}$ also represents a *partition*.
3. Finally, we verify that the set $\{\mathsf{eqpart}_i(A_H)\}$ corresponds to an *equitable* partition of H which, together with the partition corresponding to $\{\mathsf{eqpart}_i(A_G)\}$, witnesses that G and H have a *common* equitable partition. Since $\{\mathsf{eqpart}_i(A_G)\}$ is an equitable partition,

$$\mathsf{diag}(\mathsf{eqpart}_i(A_G)) \cdot A_G \cdot \mathsf{diag}(\mathsf{eqpart}_j(A_G)) = \mathsf{deg}(v, V_j) \times \mathsf{diag}(\mathsf{eqpart}_i(A_G)),$$

for some $v \in V_i$. Here, $\mathcal{V} = \{V_1, \ldots, V_\ell\}$ denotes the equitable partition corresponding to indicator vectors $\{\mathsf{eqpart}_i(A_G)\}$. Then, $G \equiv_{\mathsf{ML}(\mathcal{L}^+)} H$ implies that $\mathsf{diag}(\mathsf{eqpart}_i(A_H)) \cdot A_H \cdot \mathsf{diag}(\mathsf{eqpart}_j(A_H))$ is the diagonal matrix $\mathsf{deg}(v, V_j) \times \mathsf{diag}(\mathsf{eqpart}_i(A_H))$. Hence, we have that $\mathsf{deg}(w, W_j) = \mathsf{deg}(w', W_j)$, for any $w, w' \in W_i$, and furthermore, $\mathsf{deg}(v, V_j) = \mathsf{deg}(w, W_j)$. Here, we denote by $\mathcal{W} = \{W_1, \ldots, W_\ell\}$ the partition corresponding to $\{\mathsf{eqpart}_i(A_H)\}$.

All combined, we may conclude that G and H have indeed a common equitable partition. ◀

7.2 Characterisations

For $\mathsf{ML}(\cdot, {}^*, \mathbb{1}, \mathsf{diag}, +, \times, \mathsf{apply}_\mathsf{s}[f], f \in \Omega)$ we also have the converse.

▶ **Proposition 7.6.** *Let G and H be two graphs of the same order. If G and H have a common equitable partition, then $e(A_G) = e(A_H)$ for any sentence $e(X)$ in $\mathsf{ML}(\cdot, {}^*, \mathbb{1}, \mathsf{diag}, +, \times, \mathsf{apply}_\mathsf{s}[f], f \in \Omega)$.*

Proof. Let $\mathcal{V} = \{V_1, \ldots, V_\ell\}$ and $\mathcal{W} = \{W_1, \ldots, W_\ell\}$ be the common coarsest equitable partitions of G and H, respectively. Denote by $\{\mathbb{1}_{V_i}\}$ and $\{\mathbb{1}_{W_i}\}$, for $i = 1, \ldots, \ell$, the corresponding indicator vectors. We know from Proposition 7.2 that there exists a doubly stochastic matrix S such that $A_G \cdot S = S \cdot A_H$. In fact, S can be assumed to have a *block structure* in which the only non-zero blocks are those relating $\mathbb{1}_{V_i}$ and $\mathbb{1}_{W_i}$ [50]. As a consequence, $\mathbb{1}_{V_i} = S \cdot \mathbb{1}_{W_i}$ and $\mathbb{1}_{V_i}^\mathsf{t} \cdot S = \mathbb{1}_{W_i}^\mathsf{t}$ for $i = 1, \ldots, \ell$. The key insight in the proof is that when $e(A_G)$ is an $n \times 1$-vector, it can be written as a linear combination of $\mathbb{1}_{V_i}$'s,

say $\sum a_i \times \mathbb{1}_{V_i}$. Moreover, also $e(A_H) = \sum a_i \times \mathbb{1}_{W_i}$. As a consequence, $e(A_G) = S \cdot e(A_H)$ meaning that $e(A_G)$ is just a permutation of $e(A_H)$. For this to hold, it is essential that we work with equitable partitions common to G and H. For example, if $e(X) := X \cdot \mathbb{1}(X)$ then

$$e(A_G) = A_G \cdot \mathbb{1} = \sum_{i=1}^{\ell} A_G \cdot \mathbb{1}_{V_i} = \sum_{i,j=1}^{\ell} \mathsf{deg}(v_i, V_j) \times \mathbb{1}_{V_i}$$
$$= \sum_{i,j=1}^{\ell} \mathsf{deg}(w_i, W_j) \times (S \cdot \mathbb{1}_{W_i}) = S \cdot e(A_H),$$

for some $v_i \in V_i$ and $w_i \in W_i$. The challenging case in the proof is when $e(X) := \mathsf{diag}(e'(X))$. Based on the decomposition of $n \times 1$-vectors and the block structure of S, we have

$$\mathsf{diag}(e'(A_G)) \cdot S = \sum_{i=1}^{\ell} a_i \times \mathsf{diag}(\mathbb{1}_{V_i}) \cdot S = \sum_{i=1}^{\ell} a_i \times (S \cdot \mathsf{diag}(\mathbb{1}_{W_i})) = S \cdot \mathsf{diag}(e'(A_H)),$$

which allows to prove that $A_G \cdot S = S \cdot A_H$ implies that $e(A_G) = e(A_H)$ for all sentences in our fragment. ◀

All combined, we obtain the following characterisation.

▶ **Theorem 7.7.** *Let G and H be two graphs of the same order. Then, $G \equiv_{\mathsf{ML}(\cdot,^*,\mathbb{1},\mathsf{diag})} H$ if and only if $G \equiv_{\mathsf{ML}(\cdot,^*,\mathbb{1},\mathsf{diag},+,\times,\mathsf{apply}_s[f],f\in\Omega)} H$ if and only if there is a doubly stochastic matrix S such that $A_G \cdot S = S \cdot A_H$ if and only if $G \equiv_{\mathsf{C}^2} H$.*

As a consequence, following Example 7.3, sentences in $\mathsf{ML}(\cdot,^*, \mathbb{1}, \mathsf{diag})$ can distinguish G_1 () and H_1 (), G_2 () and H_2 (), G_4 () and H_4 (), but cannot distinguish G_3 () and H_3 ().

We next turn our attention to $\mathsf{ML}(\cdot, \mathsf{tr}, \mathbb{1}, \mathsf{diag})$- and $\mathsf{ML}(\cdot, ^*, \mathsf{tr}, \mathbb{1}, \mathsf{diag}, +, \times, \mathsf{apply}_s[f], f \in \Omega)$-equivalence. Theorem 5.3 implies that G and H are co-spectral and we thus need to combine the existence of a common equitable partition with the existence of an orthogonal matrix O such that $A_G \cdot O = O \cdot A_H$. We remark that we cannot simply require O to be doubly stochastic as this would imply that O is a permutation matrix[2], which in turn would imply that G and H are isomorphic, contradicting that our fragments cannot go beyond C^3-equivalence, as we see later.

A characterisation is obtained inspired by a characterisation of simultaneous equivalence of the so-called 1-dimensional Weisfeiler-Lehman closure of adjacency matrices [52]. Let $\mathcal{V} = \{V_1, \ldots, V_\ell\}$ and $\mathcal{W} = \{W_1, \ldots, W_\ell\}$ be common equitable partitions of G and H. Following Thüne [52], we say that an orthogonal matrix O such that $A_G \cdot O = O \cdot A_H$ is *compatible with $\mathcal{V}$ and $\mathcal{W}$* if O can be block partitioned into ℓ orthogonal matrices O_i of size $|V_i|$ such that $\mathbb{1}_{V_i} = O \cdot \mathbb{1}_{W_i}$, for all $i = 1, \ldots, \ell$. Given this notion, we have the following characterisation.

▶ **Theorem 7.8.** *Let G and H be graphs of the same order. Then the following holds: $G \equiv_{\mathsf{ML}(\cdot,\mathsf{tr},\mathbb{1},\mathsf{diag})} H$ if and only if $G \equiv_{\mathsf{ML}(\cdot,^*,\mathsf{tr},\mathbb{1},\mathsf{diag},+,\times,\mathsf{apply}_s[f],f\in\Omega)} H$ if and only if G and H have a common equitable partition, say $\mathcal{V}$ and $\mathcal{W}$, and furthermore $A_G \cdot O = O \cdot A_H$ for some orthogonal matrix O that is compatible with $\mathcal{V}$ and $\mathcal{W}$.*

[2] This is an immediate consequence of the Birkhoff-von Neumann Theorem which states that any doubly stochastic matrix lies in the convex hull of permutation matrices [45].

Proof. If $G \equiv_{\mathsf{ML}(\cdot,^*,\mathsf{tr},\mathbb{1},\mathsf{diag},+,\times,\mathsf{apply}_s[f],f\in\Omega)} H$, then for any k, $\mathsf{tr}(e(A_G)^k) = \mathsf{tr}(e(A_H)^k)$ for any expression $e(X)$ such that $e(A_G)$ (and thus also $e(A_H)$) is an $n \times n$-matrix. As argued in Thüne [52] this implies the existence of a single orthogonal matrix O such that $A_G \cdot O = O \cdot A_H$ and $e(A_G) \cdot O = O \cdot e(A_H)$. (The proof relies on Specht's Theorem which relates the existence of an orthogonal matrix *simultaneously* linking sets of matrices to trace equality conditions [37].) In particular, $\mathsf{diag}(\mathsf{eqpart}_i(A_G)) \cdot O = O \cdot \mathsf{diag}(\mathsf{eqpart}_i(A_H))$, for $i = 1, \ldots, \ell$, where $\mathsf{eqpart}_i(X)$ are the expressions computing the equitable partition given in the proof of Proposition 7.5. Lemma 6 in Thüne [52] shows that O must be compatible with the common equitable partitions represented by $\mathsf{eqpart}_i(A_G)$ and $\mathsf{eqpart}_i(A_H)$.

For the converse, we argue as in Proposition 7.6, using orthogonal matrices (which preserve the trace operation) instead of doubly stochastic matrices. ◀

Note that $G \equiv_{\mathsf{ML}(\cdot,\mathsf{tr},\mathbb{1},\mathsf{diag})} H$ implies $G \equiv_{\mathsf{ML}(\cdot,^*,\mathbb{1},\mathsf{diag})} H$. The converse does not hold.

▶ **Example 7.9.** Consider G_3 () and H_3 (). These graphs are fractional isomorphic but are not co-spectral. Hence, $G_3 \not\equiv_{\mathsf{ML}(\cdot,\mathsf{tr},\mathbb{1},\mathsf{diag})} H_3$ since $\mathsf{ML}(\cdot,\mathsf{tr},\mathbb{1},\mathsf{diag})$-equivalence implies co-spectrality. On the other hand, G_5 () and H_5 () are co-spectral regular graphs [55], with co-spectral complements, which cannot be distinguished by $\mathsf{ML}(\cdot,\mathsf{tr},\mathbb{1},\mathsf{diag})$.

A close inspection of the proofs of Proposition 7.6 and Theorem 7.8, shows that $G \equiv_{\mathsf{ML}(\mathcal{L}^+)} H$ implies that for any expression $e(X)$ in $\mathsf{ML}(\mathcal{L}^+)$ such that $e(A_G)$ (and thus also $e(A_H)$) is an $n \times 1$-vector, $e(A_G)$ is a permutation of $e(A_H)$. Indeed, both can be written as linear combinations of indicator vectors, $e(A_G)$ in terms of $\mathbb{1}_{V_i}$'s and $e(A_H)$ in terms of $\mathbb{1}_{W_i}$'s, using the *same* coefficients. This implies that we can allow pointwise function applications *on vectors* and scalars, denoted by $\mathsf{apply}_\mathsf{v}[f]$, $f \in \Omega$, at no increase in expressiveness.

▶ **Corollary 7.10.** *Let G and H be two graphs of the same order. We have that $G \equiv_{\mathsf{ML}(\mathcal{L})} H$ if and only if $G \equiv_{\mathsf{ML}(\mathcal{L}^+\cup\{\mathsf{apply}_\mathsf{v}[f],f\in\Omega\})} H$.*

▶ Remark 7.11. An equitable partition can be defined *without* the $\mathsf{diag}(\cdot)$ operation, provided that function applications on *vectors* are allowed. Hence, the same story holds when first adding pointwise function applications on vectors to $\mathsf{ML}(\cdot,^*,\mathbb{1})$ and $\mathsf{ML}(\cdot,\mathsf{tr},\mathbb{1})$, rather than first adding $\mathsf{diag}(\cdot)$ like we did in this section.

8 The Impact of Pointwise Functions on Matrices

We conclude by considering pointwise function applications on *matrices*, the only operation from Table 1 that we did not consider yet. As we will see shortly, *pointwise multiplication of matrices*, also known as the Schur-Hadamard product, is what results in an increase in expressive power. We denote the Schur-Hadamard product by the binary operator $\circ$, i.e., $(A \circ B)_{ij} = A_{ij}B_{ij}$ for matrices A and B.

▶ **Example 8.1.** We recall that in expression $\#\mathsf{3degr}(X)$ in Example 7.1, products of diagonal matrices resulted in the ability to zoom in on *vertices* that carry specific degree information. When diagonal matrices are concerned, the product of matrices coincides with pointwise multiplication of the *vectors* on the diagonals. Allowing pointwise multiplication on matrices has the same effect, but now on *edges* in graphs. As an example, suppose that we want to count the number of "triangle paths" in G, i.e., paths $(v_0, \ldots, v_k)$ of length k in G such that each edge (v_{i-1}, v_i) on the path is part of a triangle. This can be done by expression

$$\#\Delta\mathsf{paths}_k(X) := \mathbb{1}(X)^* \cdot ((\mathsf{apply}[f_{>0}](X^2 \circ X))^k \cdot \mathbb{1}(X),$$

where $f_{>0}(x) = 1$ if $x \neq 0$ and $f_{>0}(x) = 0$ otherwise [3]. Indeed, when evaluated on adjacency matrix A_G, $A_G^2 \circ A_G$ extracts from A_G^2 only those entries corresponding to paths (u, v, w) of length 2 such that (u, w) is an edge as well, i.e., it identifies edges involved in triangles. Then, $\mathsf{apply}[f_{>0}](A_G^2 \circ A_G)$ sets all non-zero entries to 1. By considering the kth power of this matrix and summing up all its entries, the number of triangle paths is obtained. It can be verified that for graphs G_5 () and H_5 (), $\#\Delta\mathsf{paths}_2(A_{G_5}) = [160] \neq [132] = \#\Delta\mathsf{paths}_2(A_{H_5})$ and hence, they can be distinguished when the Schur-Hadamard product is available. Recall that all previous fragments could not distinguish between these two graphs.

In fact, in $\mathsf{ML}(\cdot, ^*, \mathsf{tr}, \mathbb{1}, \mathsf{diag}, +, \times, \circ)$ we can compute the *coarsest stable edge colouring* of a graph $G = (V, E)$ which arises as the result of applying the edge colouring algorithm by Weisfeiler-Lehman [6, 11, 47, 57]. Initially, an edge colouring $\chi_0 : V \times V \to \{0, 1, 2\}$ is defined such that $\chi_0(v, v) = 2$, $\chi_0(v, w) = 1$ if $\{v, w\} \in E$, and $\chi_0(v, w) = 0$ for $v \neq w$ and $\{v, w\} \notin E$. Such a colouring naturally induces a partitioning Π_{χ_0} of $V \times V$. A colouring $\chi : V \times V \to C$ for some set of colours C is called *stable* if and only if for any two pairs (v_1, v_2) and (v_1', v_2') in $V \times V$,

$$\chi(v_1, v_2) = \chi(v_1', v_2') \Leftrightarrow \mathsf{L}^2(v_1, v_2) = \mathsf{L}^2(v_1', v_2'),$$

where for a pair $(v, v') \in V \times V$ and pairs (c, d) of colours in C,

$$\mathsf{L}^2(v, v') := \{(c, d, p_{v,v'}^{c,d}) \mid p_{v,v'}^{c,d} \neq 0\} \text{ and } p_{v,v'}^{c,d} := |\{v'' \in V \mid \chi(v, v'') = c, \chi(v'', v') = d\}|.$$

In other words, $\mathsf{L}^2(v, v')$ lists the number of triangles (v, v', v'') in which (v, v'') has colour c and (v'', v) has colour d, for each pair of colours. Such a stable edge colouring χ is called *coarsest* when the corresponding edge partition Π_χ is the coarsest stable edge partition. That is, Π_χ refines Π_{χ_0}, χ is stable and any other colouring satisfying these conditions results in a finer partition than Π_χ.

Two graphs $G = (V, E)$ and $H = (W, F)$ are said to be *indistinguishable by edge colouring*, denoted by $G \equiv_{\mathsf{2WL}} H$, if the following holds. Let $\Pi_{\chi_G} = \{E_1, \ldots, E_\ell\}$ and $\Pi_{\chi_H} = \{F_1, \ldots, F_\ell\}$ be the edge partitions corresponding to stable edge colourings χ_G and χ_H of H. Then, $G \equiv_{\mathsf{2WL}} H$ if there is a bijection $\imath : \Pi_{\chi_G} \to \Pi_{\chi_H}$ such that E_i and $F_{\imath(i)}$ have the same colour and the same number of entries carrying value 1.

In the seminal paper by Cai, Fürer and Immerman [11], the following was shown.

▶ **Theorem 8.2.** *Let G and H be two graphs of the same order. Then, $G \equiv_{\mathsf{2WL}} H$ if and only if $G \equiv_{\mathsf{C}^3} H$.*

We have the following characterisation of $\mathsf{ML}(\cdot, ^*, \mathsf{tr}, \mathbb{1}, \mathsf{diag}, +, \times, \circ)$-equivalence.

▶ **Theorem 8.3.** *Let G and H be two graphs of the same order, then $G \equiv_{\mathsf{ML}(\cdot, ^*, \mathsf{tr}, \mathbb{1}, \mathsf{diag}, +, \times, \circ)} H$ if and only if $G \equiv_{\mathsf{C}^3} H$.*

Proof. We only have space here to sketch the proof. The proof is not that different from the one used in the context of equitable partitions. Let $G = (V, E)$ and $H = (W, F)$ be two graphs. First, we simulate algorithm 2-STAB(A_G) [6], that computes the coarsest stable edge colouring, by expressions $\mathsf{stabcol}_i(X)$, for $i = 1, \ldots, \ell$, in $\mathsf{ML}(\cdot, ^*, \mathsf{tr}, \mathbb{1}, \mathsf{diag}, +, \times, \circ)$. Each $\mathsf{stabcol}_i(A_G)$ is an *indicator matrix* representing the part of the partition Π of $V \times V$

[3] The use of $\mathsf{apply}[f_{>0}](\cdot)$ is just for convenience and can be simulated when evaluated on given instances using $\cdot$, $+$, $\times$ and $\circ$.

corresponding to a specific colour. Based on well-known properties of these indicator matrices (they form standard basis of the *cellular* or *coherent algebra* associated with G [33]), we show that $G \equiv_{\mathsf{ML}(\cdot,^*,\mathsf{tr},\mathbb{1},\mathsf{diag},+,\times,\circ)} H$ implies that $\{\mathsf{stabcol}_i(A_H)\}$ also represent a partition of $W \times W$ corresponding to the coarsest stable colouring of H. Finally, G and H are shown to be indistinguishable by edge colouring, based on the partitions $\{\mathsf{stabcol}_i(A_G)\}$ and $\{\mathsf{stabcol}_i(A_H)\}$. Hence, by Theorem 8.2, $G \equiv_{\mathsf{ML}(\cdot,^*,\mathsf{tr},\mathbb{1},\mathsf{diag},+,\times,\circ)} H$ implies $G \equiv_{\mathsf{C}^3} H$.

For the converse, we use that $G \equiv_{\mathsf{2WL}} H$ implies that there exists an orthogonal matrix O such that $A_G \cdot O = O \cdot A_H$ and furthermore, the mapping $Y \mapsto O \cdot Y \cdot O^{\mathsf{t}}$ is an isomorphism between the cellular algebras of G and H. In particular, it commutes with the Schur-Hadamard product [23]. This is crucial to show that $e(A_G) = e(A_H)$ for all sentences $e(X) \in \mathsf{ML}(\cdot,^*,\mathsf{tr},\mathbb{1},\mathsf{diag},+,\times,\circ)$. ◀

▶ Remark 8.4. We can do some simplification in $\mathsf{ML}(\cdot,^*,\mathsf{tr},\mathbb{1},\mathsf{diag},+,\times,\circ)$. Indeed, the trace operator can be simulated by $\mathsf{tr}(e(X)) = \mathbb{1}(X)^* \cdot (e(X) \circ \mathsf{diag}(\mathbb{1}(X))) \cdot \mathbb{1}(X)$ and can hence be omitted. Moreover, $\mathsf{diag}(\cdot)$ can be replaced by a simpler operator, denoted by Id, which returns the identity matrix of the same dimensions as the input. Indeed, $\mathsf{diag}(e(X)) = (e(X) \cdot \mathbb{1}(X)^*) \circ \mathsf{Id}(X)$. We can thus work with $\mathsf{ML}(\cdot,^*,\mathbb{1},\mathsf{Id},+,\times,\circ)$ instead.

▶ Remark 8.5. Similar to Corollary 7.10, we can allow *any* pointwise function application on matrices. This follows from the proof of Theorem 8.3 in which it is shown that for expressions $e_i(X)$, for $i = 1,\ldots,p$, such that each $e_i(A_G)$ (and thus also each $e_i(A_H)$) is an $n \times n$-matrix, $e_i(A_G) = \sum a_j^{(i)} \times \mathsf{stabcol}_j(A_G)$ and $e(A_H) = \sum a_j^{(i)} \times \mathsf{stabcol}_i(A_H)$, for scalars $a_j^{(i)} \in \mathbb{C}$. This implies that $\mathsf{apply}[f]\big(e_1(A_G),\ldots,e_p(A_G)\big) = \sum f(a_j^{(1)},\ldots,a_j^{(p)}) \times \mathsf{stabcol}_j(A_G)$, and similarly, $\mathsf{apply}[f]\big(e_1(A_H),\ldots,e_p(A_H)\big) = \sum f(a_j^{(1)},\ldots,a_j^{(p)}) \times \mathsf{stabcol}_j(A_H)$ As a consequence, $\mathsf{apply}[f]\big(e_1(A_G),\ldots,e_p(A_G)\big) \cdot O = O \cdot \mathsf{apply}[f]\big(e_1(A_H),\ldots,e_p(A_H)\big)$ for the orthogonal matrix O in the proof of Theorem 8.3. This suffices to show that $e(A_G) = e(A_H)$ for any sentence $e(X)$ in $\mathsf{ML}(\cdot,^*,\mathsf{tr},\mathbb{1},\mathsf{diag},+,\times,\mathsf{apply}[f], f \in \Omega)$, or in other words, for any sentence in MATLANG.

▶ Remark 8.6. The orthogonal matrix O in the proof of Theorem 8.3 can be taken to be compatible with the common equitable partitions of G and H, just as in Theorem 7.8. This follows from the fact that the diagonal indicator matrices $\mathsf{diag}(\mathsf{eqpart}_i(A_G))$ are part of the indicator matrices that constitute the basis of the cellular algebra of G [6].

▶ Remark 8.7. An almost direct consequence of Theorem 8.3 is that $G \equiv_{\mathsf{2WL}} H$ implies that G and H have the same number of (simple) cycles of length 7. This was known to hold for cycles of length $\ell = 1,2,\ldots,6$, and known not to hold for cycles of length greater than 7 [25]. The case $\ell = 7$ was left open in Fürer [25]. In view of Theorem 8.3 it suffices to show that we can count cycles of length $\ell = 1,2,\ldots,7$ in MATLANG. This, however, is a direct consequence of the formulas for counting cycles given in Noga et al. [1]. Indeed, a close inspection of these formulas reveals that only limited linear algebra functionality is required and hence they can be formulated in MATLANG. Although formulas exist for counting cycles of length greater than 7, they require to count the number of 4-cliques, which is not possible in MATLANG.

9 Concluding Remarks

We have characterised $\mathsf{ML}(\mathcal{L})$-equivalence for undirected graphs and clearly identified what additional distinguishing power each of the operations has. That natural characterisations can be obtained once more attests that MATLANG is an adequate matrix language. Although motivated by the increased interest in integrating linear algebra functionality in database

management systems, the presented results are primarily of theoretical interest. As such, they do not directly translate into effective procedures for evaluating or optimising linear algebra inside database systems.

We conclude with some avenues for further investigation. Although some of the results generalise to directed graphs (with asymmetric adjacency matrices), an extension to the case when queries can have multiple inputs seems challenging. The generalisation beyond graphs, i.e., for arbitrary matrices, is wide open. Of interest may also be to connect $\mathsf{ML}(\mathcal{L})$-equivalence to fragments of first-order logic (without counting). A possible line of attack could be to work over the boolean semiring instead of over the complex numbers (see Grohe and Otto [29] for a similar approach). More general semirings could open the way for modelling and querying labeled graphs using matrix query languages.

We also note that MATLANG was extended in Brijder et al. [9] with an operator inv that computes the inverse of a matrix, if it exists, and returns the zero matrix otherwise. The extension, $\mathsf{MATLANG} + \mathsf{inv}$, was shown to be more expressive than MATLANG. For example, connectedness of graphs can be checked by a single sentence in $\mathsf{MATLANG} + \mathsf{inv}$. Of course, we here consider *equivalence* of graphs. Even when considering a "classical" logic like FO^3, the three-variable fragment of first-order logic, $G \equiv_{\mathsf{FO}^3} H$ implies that G is connected if and only if H is connected. Translated to our setting, for any fragment $\mathsf{ML}(\mathcal{L})$ in which $G \equiv_{\mathsf{ML}(\mathcal{L})} H$ implies that the Laplacian $\mathsf{diag}(A_G \cdot \mathbb{1}) - A_G$ of G is co-spectral with the Laplacian of $\mathsf{diag}(A_H \cdot \mathbb{1}) - A_H$ of H, $G \equiv_{\mathsf{ML}(\mathcal{L})} H$ implies that G is connected if and only if H is connected. It even implies that G and H must have the same number of connected components, as this is determined by the multiplicity of the eigenvalue 0 of the Laplacian [10]. Nevertheless, we can also consider equivalence of graphs relative to $\mathsf{MATLANG} + \mathsf{inv}$. We observe, however, that the inverse of a matrix can be computed using $+$ and $\times$, by the Cayley-Hamilton Theorem [4], given the coefficients of the characteristic polynomial of the adjacency matrix. These coefficients can be computed using $+$, $\times$ and tr. For fragments supporting $\cdot$, $+$, $\times$ and tr, the operator inv thus does not add distinguishing power. It is unclear what the impact is of inv for smaller fragments such as $\mathsf{ML}(\cdot, {}^{\cdot}\mathbb{1})$ and $\mathsf{ML}(\cdot, {}^{*}, \mathbb{1}, \mathsf{diag})$.

To relate our notion of equivalence more closely to the expressiveness questions studied in Brijder et al. [9], it may be interesting to investigate notions of *locality* of $\mathsf{ML}(\mathcal{L})$ expressions, as this underlies the inexpressibility of connectivity of MATLANG [42]. It would be nice if this can be achieved in purely algebraic terms, without relying on locality notions in logic.

To conclude, MATLANG was also extended with an eigen operator which returns a matrix whose columns consist of eigenvectors spanning the eigenspaces [9]. Since the choice of eigenvectors is not unique, this results in a non-deterministic semantics. We leave it for future work to study the equivalence of graphs relative to *deterministic* fragments supporting the eigen operator, i.e., such that the result of expressions does not depend on the eigenvectors returned. As a starting point one could, for example, force determinism by considering a certain answer semantics. That is, if $e(X)$ is an expression using $\mathsf{eigen}(X)$, one can define $\mathsf{cert}(e(A_G)) := \bigcap_V e(A_G, V)$, where V ranges over all bases of the eigenspaces. Distinguishability with regards to such a certain answer semantics demands further investigation.

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. doi:10.1007/BF02523189.

2 Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan Reutter, and Domagoj Vrgoč. Foundations of Modern Query Languages for Graph Databases. *ACM Comput. Surv.*, 50(5):68:1–68:40, 2017. doi:10.1145/3104031.

3 Albert Atserias and Elitza N. Maneva. Sherali-Adams Relaxations and Indistinguishability in Counting Logics. *SIAM J. Comput.*, 42(1):112–137, 2013. doi:10.1137/120867834.

4 Sheldon Axler. *Linear Algebra Done Right*. Springer, third edition, 2015. doi:10.1007/978-3-319-11080-6.

5 Pablo Barceló. Querying Graph Databases. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS, pages 175–188, 2013. doi:10.1145/2463664.2465216.

6 Oliver Bastert. *Stabilization procedures and applications*. PhD thesis, Technical University Munich, Germany, 2001. URL: http://nbn-resolving.de/urn:nbn:de:bvb:91-diss2002070500045.

7 Matthias Boehm, Michael W. Dusenberry, Deron Eriksson, Alexandre V. Evfimievski, Faraz Makari Manshadi, Niketan Pansare, Berthold Reinwald, Frederick R. Reiss, Prithviraj Sen, Arvind C. Surve, and Shirsih Tatikonda. SystemML: Declarative machine learning on Spark. *Proceedings of the VLDB Endowment*, 9(13):1425–1436, 2016. doi:10.14778/3007263.3007279.

8 Matthias Boehm, Berthold Reinwald, Dylan Hutchison, Prithviraj Sen, Alexandre V. Evfimievski, and Niketan Pansare. On Optimizing Operator Fusion Plans for Large-scale Machine Learning in SystemML. *Proceedings of the VLDB Endowment*, 11(12):1755–1768, 2018. doi:10.14778/3229863.3229865.

9 Robert Brijder, Floris Geerts, Jan Van den Bussche, and Timmy Weerwag. On the Expressive Power of Query Languages for Matrices. In *21st International Conference on Database Theory*, ICDT, pages 10:1–10:17, 2018. doi:10.4230/LIPIcs.ICDT.2018.10.

10 Andries E. Brouwer and Willem H. Haemers. *Spectra of Graphs*. Universitext. Springer, 2012. doi:10.1007/978-1-4614-1939-6.

11 Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992. doi:10.1007/BF01305232.

12 Lingjiao Chen, Arun Kumar, Jeffrey Naughton, and Jignesh M. Patel. Towards Linear Algebra over Normalized Data. *Proceedings of the VLDB Endowment*, 10(11):1214–1225, 2017. doi:10.14778/3137628.3137633.

13 Dragoš M. Cvetković. GRAPHS AND THEIR SPECTRA. *Publikacije Elektrotehničkog fakulteta. Serija Matematika i fizika*, 354/356:1–50, 1971. URL: http://www.jstor.org/stable/43667526.

14 Dragoš M. Cvetković. The main part of the spectrum, divisors and switching of graphs. *Publ. Inst. Math. (Beograd) (N.S.)*, 23(37):31–38, 1978. URL: http://elib.mi.sanu.ac.rs/files/journals/publ/43/6.pdf.

15 Dragoš M. Cvetković, Peter Rowlinson, and Slobodan Simić. *Eigenspaces of Graphs*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1997. doi:10.1017/CBO9781139086547.

16 Dragoš M. Cvetković, Peter Rowlinson, and Slobodan Simić. *An Introduction to the Theory of Graph Spectra*. London Mathematical Society Student Texts. Cambridge University Press, 2009. doi:10.1017/CBO9780511801518.

17 Anuj Dawar. On the Descriptive Complexity of Linear Algebra. In *Proceedings of the 15th International Workshop on Logic, Language, Information and Computation*, WoLLIC, pages 17–25, 2008. doi:10.1007/978-3-540-69937-8_2.

18 Anuj Dawar, Martin Grohe, Bjarki Holm, and Bastian Laubner. Logics with Rank Operators. In *Proceedings of the 24th Annual IEEE Symposium on Logic In Computer Science*, LICS, pages 113–122, 2009. doi:10.1109/LICS.2009.24.

19 Anuj Dawar and Bjarki Holm. Pebble games with algebraic rules. *Fund. Inform.*, 150(3-4):281–316, 2017. doi:10.3233/FI-2017-1471.

20 Anuj Dawar, Simone Severini, and Octavio Zapata. Descriptive Complexity of Graph Spectra. In *Proceedings of the 23rd International Workshop on Logic, Language, Information and Computation*, WoLLIC, pages 183–199, 2016. doi:10.1007/978-3-662-52921-8_12.

21 Holger Dell, Martin Grohe, and Gaurav Rattan. Lovász Meets Weisfeiler and Lehman. In *45th International Colloquium on Automata, Languages, and Programming*, ICALP, pages 40:1–40:14, 2018. doi:10.4230/LIPIcs.ICALP.2018.40.

22 Ahmed Elgohary, Matthias Boehm, Peter J. Haas, Frederick R. Reiss, and Berthold Reinwald. Compressed linear algebra for large-scale machine learning. *The VLDB Journal*, pages 1–26, 2017. doi:10.1007/s00778-017-0478-1.

23 Shmuel Friedland. Coherent algebras and the graph isomorphism problem. *Discrete Applied Mathematics*, 25(1):73–98, 1989. doi:10.1016/0166-218X(89)90047-4.

24 Martin Fürer. On the power of combinatorial and spectral invariants. *Linear Algebra and its Applications*, 432(9):2373–2380, 2010. doi:10.1016/j.laa.2009.07.019.

25 Martin Fürer. On the Combinatorial Power of the Weisfeiler-Lehman Algorithm. In Dimitris Fotakis, Aris Pagourtzis, and Vangelis Th. Paschos, editors, *Algorithms and Complexity*, pages 260–271. Springer, 2017. doi:10.1007/978-3-319-57586-5_22.

26 Chris Godsil and Gordon F. Royle. *Algebraic Graph Theory*, volume 207 of *Graduate Texts in Mathematics*. Springer, 2001. doi:10.1007/978-1-4613-0163-9.

27 Erich Grädel and Wied Pakusa. Rank Logic is Dead, Long Live Rank Logic! In *24th EACSL Annual Conference on Computer Science Logic*, CSL, pages 390–404, 2015. doi:10.4230/LIPIcs.CSL.2015.390.

28 Martin Grohe, Kristian Kersting, Martin Mladenov, and Erkal Selman. Dimension Reduction via Colour Refinement. In *22th Annual European Symposium on Algorithms*, ESA, pages 505–516, 2014. doi:10.1007/978-3-662-44777-2_42.

29 Martin Grohe and Martin Otto. Pebble games and linear equations. *The Journal of Symbolic Logic*, 80(3):797–844, 2015. doi:10.1017/jsl.2015.28.

30 Martin Grohe and Wied Pakusa. Descriptive complexity of linear equation systems and applications to propositional proof complexity. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS, pages 1–12, 2017. doi:10.1109/LICS.2017.8005081.

31 Willem H. Haemers and Edward Spence. Enumeration of cospectral graphs. *European J. Combin.*, 25(2):199–211, 2004. doi:10.1016/S0195-6698(03)00100-8.

32 Frank Harary and Allen J. Schwenk. The spectral approach to determining the number of walks in a graph. *Pacific J. Math.*, 80(2):443–449, 1979. URL: https://projecteuclid.org:443/euclid.pjm/1102785717.

33 Donald G. Higman. Coherent algebras. *Linear Algebra and its Applications*, 93:209–239, 1987. doi:10.1016/S0024-3795(87)90326-0.

34 Bjarki Holm. *Descriptive Complexity of Linear Algebra*. PhD thesis, University of Cambridge, 2010.

35 Dylan Hutchison, Bill Howe, and Dan Suciu. LaraDB: A Minimalist Kernel for Linear and Relational Algebra Computation. In *Proceedings of the 4th ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond*, BeyondMR, pages 2:1–2:10, 2017. doi:10.1145/3070607.3070608.

36 Neil Immerman and Eric Lander. Describing Graphs: A First-Order Approach to Graph Canonization. In Alan L. Selman, editor, *Complexity Theory Retrospective: In Honor of Juris Hartmanis on the Occasion of His Sixtieth Birthday*, pages 59–81. Springer, 1990. doi:10.1007/978-1-4612-4478-3_5.

37 Naihuan Jing. Unitary and orthogonal equivalence of sets of matrices. *Linear Algebra and its Applications*, 481:235–242, 2015. doi:10.1016/j.laa.2015.04.036.

38 Charles R Johnson and Morris Newman. A note on cospectral graphs. *Journal of Combinatorial Theory, Series B*, 28(1):96–103, 1980. doi:10.1016/0095-8956(80)90058-1.

39 Kristian Kersting, Martin Mladenov, Roman Garnett, and Martin Grohe. Power Iterated Color Refinement. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, AAAI'14, pages 1904–1910. AAAI Press, 2014. URL: `https://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8377`.

40 Andreas Kunft, Alexander Alexandrov, Asterios Katsifodimos, and Volker Markl. Bridging the Gap: Towards Optimization Across Linear and Relational Algebra. In *Proceedings of the 3rd ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond*, BeyondMR, pages 1:1–1:4, 2016. `doi:10.1145/2926534.2926540`.

41 Andreas Kunft, Asterios Katsifodimos, Sebastian Schelter, Tilmann Rabl, and Volker Markl. BlockJoin: Efficient Matrix Partitioning Through Joins. *Proceedings of the VLDB Endowment*, 10(13):2061–2072, 2017. `doi:10.14778/3151106.3151110`.

42 Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. `doi:10.1007/978-3-662-07003-1`.

43 Shangyu Luo, Zekai J. Gao, Michael Gubanov, Luis L. Perez, and Christopher Jermaine. Scalable Linear Algebra on a Relational Database System. *SIGMOD Rec.*, 47(1):24–31, 2018. `doi:10.1145/3277006.3277013`.

44 Peter N. Malkin. Sherali–Adams relaxations of graph isomorphism polytopes. *Discrete Optimization*, 12:73–97, 2014. `doi:10.1016/j.disopt.2014.01.004`.

45 Albert W. Marshall, Ingram Olkin, and Barry C. Arnold. *Inequalities: Theory of Majorization and its Applications*. Springer, 2011. `doi:10.1007/978-0-387-68276-1`.

46 Hung Q. Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. In-Database Factorized Learning. In *Proceedings of the 11th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web*, AMW, 2017. URL: `http://ceur-ws.org/Vol-1912/paper21.pdf`.

47 Martin Otto. *Bounded Variable Logics and Counting: A Study in Finite Models*, volume 9 of *Lecture Notes in Logic*. Cambridge University Press, 2017. `doi:10.1017/9781316716878`.

48 Motakuri V. Ramana, Edward R. Scheinerman, and Daniel Ullman. Fractional isomorphism of graphs. *Discrete Mathematics*, 132(1-3):247–265, 1994. `doi:10.1016/0012-365X(94)90241-0`.

49 Peter Rowlinson. The main eigenvalues of a graph: A survey. *Applicable Analysis and Discrete Mathematics*, 1(2):455–471, 2007. URL: `http://www.jstor.org/stable/43666075`.

50 Edward R. Scheinerman and Daniel H. Ullman. *Fractional Graph Theory: a Rational Approach to the Theory of Graphs*. John Wiley & Sons, 1997. URL: `https://www.ams.jhu.edu/ers/wp-content/uploads/sites/2/2015/12/fgt.pdf`.

51 Maximilian Schleich, Dan Olteanu, and Radu Ciucanu. Learning Linear Regression Models over Factorized Joins. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD, pages 3–18, 2016. `doi:10.1145/2882903.2882939`.

52 Mario Thüne. *Eigenvalues of matrices and graphs*. PhD thesis, University of Leipzig, 2012.

53 Gottfried Tinhofer. Graph isomorphism and theorems of Birkhoff type. *Computing*, 36(4):285–300, 1986. `doi:10.1007/BF02240204`.

54 Gottfried Tinhofer. A note on compact graphs. *Discrete Applied Mathematics*, 30(2):253–264, 1991. `doi:10.1016/0166-218X(91)90049-3`.

55 Edwin R. van Dam and Willem H. Haemers. Which graphs are determined by their spectrum? *Linear Algebra and its Applications*, 373:241–272, 2003. `doi:10.1016/S0024-3795(03)00483-X`.

56 Erwin R. van Dam, Willem H. Haemers, and Jack H. Koolen. Cospectral graphs and the generalized adjacency matrix. *Linear Algebra and its Applications*, 423(1):33–41, 2007. `doi:10.1016/j.laa.2006.07.017`.

57 Boris Weisfeiler. *On Construction and Identification of Graphs*. Number 558 in Lecture Notes in Mathematics. Springer-Verlag Berlin Heidelberg, 1976. `doi:10.1007/BFb0089374`.

Fragments of Bag Relational Algebra: Expressiveness and Certain Answers

Marco Console
School of Informatics, University of Edinburgh, United Kingdom
console.marco@gmail.com

Paolo Guagliardo
School of Informatics, University of Edinburgh, United Kingdom
paolo.guagliardo@ed.ac.uk

Leonid Libkin
School of Informatics, University of Edinburgh, United Kingdom
libkin@inf.ed.ac.uk

Abstract

While all relational database systems are based on the bag data model, much of theoretical research still views relations as sets. Recent attempts to provide theoretical foundations for modern data management problems under the bag semantics concentrated on applications that need to deal with incomplete relations, i.e., relations populated by constants and nulls. Our goal is to provide a complete characterization of the complexity of query answering over such relations in fragments of bag relational algebra.

The main challenges that we face are twofold. First, bag relational algebra has more operations than its set analog (e.g., additive union, max-union, min-intersection, duplicate elimination) and the relationship between various fragments is not fully known. Thus we first fill this gap. Second, we look at query answering over incomplete data, which again is more complex than in the set case: rather than certainty and possibility of answers, we now have numerical information about occurrences of tuples. We then fully classify the complexity of finding this information in all the fragments of bag relational algebra.

2012 ACM Subject Classification Theory of computation → Database theory; Theory of computation → Database query languages (principles); Theory of computation → Incomplete, inconsistent, and uncertain databases; Information systems → Relational database query languages; Information systems → Structured Query Language

Keywords and phrases bag semantics, relational algebra, expressivity, certain answers, complexity

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.8

Funding Work supported by EPSRC grants M025268 and N023056.

Acknowledgements The authors thank Etienne Toussaint and the referees for their helpful comments.

1 Introduction

While all relational database management systems (DBMSs) use bags as the basis of their data model, much of relational database theory uses a model based on sets, thus disallowing repetitions of tuples. The presence of duplicates in real-life databases is a very important consideration that is reflected in practically all aspects of data management, such as querying, storing, and accessing data [15, 30]. Theoretical research has raised this issue several times. By the early 1990s there was agreement on what the standard collection of bag relational algebra operations is [4], and in the mid 1990s their expressiveness and complexity were thoroughly studied [18, 26], albeit in the context of the model of nested relations, or complex objects, which was the research focus back then [9, 10]. Around the same time it was noticed that the well developed theory of query optimization, especially for conjunctive queries, does

not apply to bag semantics [11], and despite many attempts and partial results [23, 12], the key problem of the decidability of such optimizations remains unsolved [24]. Other languages, in particular those with aggregates and fixpoints in the spirit of Datalog, have been studied under bag semantics as well [7, 13, 26].

More recently, bag semantics has been considered in modern data management applications that combine traditional databases and reasoning tasks. In [20], fundamental problems of data integration and data exchange are studied under bag semantics and are shown to differ rather drastically from their set semantics counterparts. In [28], a similar program is carried out for ontology based data access (OBDA), where an ontology supplements information provided by a relational database in which duplicates are allowed. What is common to these applications is that in both of them one needs to query incomplete data, that is, databases with null values. The standard approach to querying such databases, which is used in data integration, data exchange, and OBDA applications, is based on the classical notion of certain answers [22].

However, when it comes to bags, the notion of certain answers becomes more complex than under set semantics. In general, an incomplete database D represents a collection $[\![D]\!] = \{D_1, D_2, \ldots\}$ of complete databases, obtained by interpreting incomplete data in D. A tuple $\bar{a}$ is a certain answer to a query q if it is in $q(D')$ for every $D' \in [\![D]\!]$; see [1, 22]. Under bag semantics, we have more information: for each tuple, we know the number $\#\big(\bar{a}, q(D')\big)$ of occurrences of $\bar{a}$ in $q(D')$. Thus, as D' ranges over $[\![D]\!]$, we have a range of numbers that define an interval between

$$\min(\bar{a}, q, D) = \min_{D' \in [\![D]\!]} \#\big(\bar{a}, q(D')\big) \qquad \text{and} \qquad \max(\bar{a}, q, D) = \max_{D' \in [\![D]\!]} \#\big(\bar{a}, q(D')\big)$$

Under set semantics, $\min(D, q, \bar{a}) = 1$ means that $\bar{a}$ is a certain answer to q on D, and $\max(D, q, \bar{a}) = 1$ means that $\bar{a}$ is a possible answer. On sets, these can be easily checked for positive relational algebra, but it is hard (coNP-complete and NP-complete, respectively) for full relational algebra [2]. This tells us that, in terms of relational algebra operations, selection, projection, Cartesian product and union are easy, but difference makes things hard. Our goal is to paint a similar picture for bags. The problems that we face are:

1. there are more operations that are included in bag relational algebra, and we have less understanding of them;
2. even for basic operations, there is very little knowledge of the complexity of answering queries over incomplete data.

We now explain these points in more detail.

Bag algebra fragments

Under set semantics, we have well understood fragments of relational algebra: SPC (select-project-Cartesian product) queries, positive relational algebra RA^+ that adds union, and full relational algebra RA that adds difference. Moreover, intersection is expressible as the natural join of two relations over the same attributes. Under bag semantics, however, the situation is different:

- SPC queries follow their set-theoretic analogs except that they keep duplicates.
- For union, there are several options: either the additive union, that adds up multiplicities (and corresponds to `UNION ALL` in SQL), or the set theoretic union (SQL's `UNION`), or the *max-union* which takes the maximum number of occurrences of a tuple.
- There are two standard notions of intersection: the set-theoretic one (SQL's `INTERSECT`), or the one taking the minimum number of occurrences of a tuple (SQL's `INTERSECT ALL`). Neither of them is the join of two relations any more.

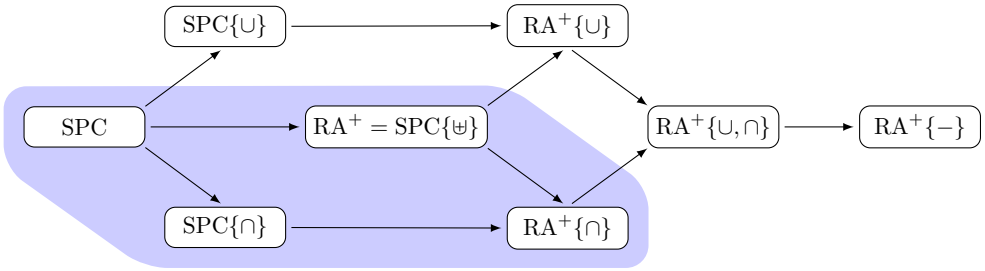


Figure 1 Summary of the results. An edge indicates a more expressive fragment. Adding duplicate elimination makes every fragment more expressive, and incomparable with $\mathrm{RA}^+\{-\}$. Extending $\mathrm{RA}^+\{-\}$ with duplicate elimination results in a fragment that has the expressive power of full RA. The shaded area includes the fragments for which computing the minimum occurrences of certain answers is tractable, while this is intractable whenever duplicate elimination is added. Computing the maximum number of occurrences is intractable for all fragments.

- There are also two versions of the difference operator, corresponding to SQL's `EXCEPT` and `EXCEPT ALL`: the former removes duplicates, and the latter keeps them.
- Finally, there is the duplicate elimination operation, that corresponds to SQL's `DISTINCT`.

The language RA of bag relational algebra then has the following operations [4, 18, 26]:

- multiplicity-preserving versions of *selection*, *projection* and *Cartesian product*, which form the class of SPC queries;
- *additive union* $\uplus$ that adds up multiplicities of tuples; together with SPC queries it forms the *positive relational algebra* RA^+;
- *max-union* $\cup$ that keeps the maximum number of occurrences of a tuple;
- *min-intersection* $\cap$ that keeps the minimum number of occurrences of a tuple;
- *difference* $-$ that subtracts the number of occurrences of a tuple up to zero, i.e., $\#(\bar{a}, R - R') = \max\big(\#(\bar{a}, R) - \#(\bar{a}, R'), 0\big)$;
- *duplicate elimination* ε that turns a bag into a set.

To understand how query answering behaves in these fragments, we first need to understand their relative expressiveness. It *might* appear that these questions have already been answered in [17, 26]. However, this was done in the context of nested relations, and the results used the power of nesting in an essential way. For the usual bag algebra with non-nested relations, as implemented in all DBMSs, these basic results are surprisingly lacking. Thus, as our first task, we shall produce a full picture of the expressiveness of bag relational algebra fragments (which will indeed be different from the known results in the nested case).

Incomplete information and bags

There is a much bigger variety of relational algebra fragments for bags, but little is known about finding $\min(\bar{a}, q, D)$ and $\max(\bar{a}, q, D)$ for queries in those fragments. We know that min is easy to compute for RA^+ queries and that for full RA the problem is computationally hard: checking whether $\min(\bar{a}, q, D) \geq n$ is NP-complete [2, 14]. Checking whether $\max(\bar{a}, q, D) \geq n$ is NP-complete even for SPC queries [14]. The complexity of actually computing min and max (or, in terms of a decision problem, checking whether $\min(D, q, D) = n$, and likewise for max) is still open.

Outline of the results

Our main results are summarized in Figure 1.

Expressiveness. We characterize the relative expressive power of RA fragments, as shown in the diagram. Furthermore, adding duplicate elimination to a fragment that does not have it results in a language that is strictly more expressive (than the original fragment), and incomparable with $\mathrm{RA}^+\{-\}$. The relative expressiveness of bag operations is indeed different from what was known in the nested relational case [17, 18, 26]. For example, over nested relations, adding min-intersection to the analog of RA^+ suffices to express max-union, but in the usual relational algebra over bags these two operations are incomparable in their expressiveness.

Complexity of min. For fragments in the shaded area, computing $\min(D, q, \bar{a})$ is tractable, and it can be done by evaluating the query naively on the incomplete database. For all fragments outside the shaded area, and all fragments with duplicate elimination (from $\mathrm{SPC}\{\varepsilon\}$ to the full RA), the complexity is intractable: checking whether $\min(\bar{a}, q, D)\ \theta\ n$ is NP-complete when θ is $\leq$, coNP-complete when θ is $\geq$, and DP-complete when θ is $=$. Recall that DP is the class of problems that are the intersection of an NP problem and a coNP problem [29].

Complexity of max. For all the fragments, inside and outside the shaded area, and with or without duplicate elimination, computing max is intractable: checking $\max(D, q, \bar{a})\ \theta\ n$ is NP-complete when θ is $\geq$, coNP-complete when θ is $\leq$, and DP-complete when θ is $=$.

Organization

Bag relational algebra is defined in Section 2, and the relative expressive power of its fragments is studied in Section 3. Query answering over bags with nulls is discussed in Section 4, and its complexity is classified in Section 5. Concluding remarks are given in Section 6.

2 Bag Relational Algebra

We now describe the standard operations of bag relational algebra and provide their semantics [4, 17, 18, 26]. A bag is a collection of elements with associated multiplicities (numbers of occurrences); if an element b occurs n times in a bag B, we write $\#(b, B) = n$. If $\#(b, B) = 0$, it means that b does not occur in B. Sets are just a special case when $\#(b, B) \in \{0, 1\}$.

In a database D, each k-ary relation name R of the schema is associated with a bag R^D of k-tuples; we will omit the superscript D whenever it is clear from the context. We assume that the attributes of a k-ary relation are $1, \ldots, k$, i.e., we adopt the unnamed perspective [1].

Syntax

The syntax of relational algebra (RA) expressions is defined as follows:

$$
\begin{array}{rcll}
e, e' & ::= & R & \text{(base relations)} \\
& | & \sigma_{i=j}(e) & \text{(selection)} \\
& | & \pi_{\alpha}(e) & \text{(projection)} \\
& | & e \times e' & \text{(Cartesian product)} \\
& | & e \uplus e' & \text{(additive union)} \\
& | & e \cup e' & \text{(max-union)} \\
& | & e \cap e' & \text{(intersection)} \\
& | & e - e' & \text{(difference)} \\
& | & \varepsilon(e) & \text{(duplicate elimination)}
\end{array}
$$

where i and j in $\sigma_{i=j}(e)$ are positive integers, and α in $\pi_\alpha(e)$ is a possibly empty tuple of positive integers.

The *arity* of RA expressions is defined as follows: for base relations, it is given by the schema; for $\sigma_{i=j}(e)$ and $\varepsilon(e)$, it is the arity of e; for $\pi_\alpha(e)$, it is the arity of α; for $e \times e'$, it is the sum of the arities of e and e'; for $e \star e'$ with $\star \in \{\cup, \uplus, \cap, -\}$, it is the arity of e.

We then say that an RA expression is well-formed w.r.t. a schema if: it mentions only relation names from the schema; i and j in $\sigma_{i=j}(e)$ are less than or equal to the arity of e; the elements of α in $\pi_\alpha(e)$ are less than or equal to the arity of e; the expressions e and e' in $e \star e'$, with $\star \in \{\cup, \uplus, \cap, -\}$, have the same arity. In the rest of the paper, we implicitly assume that we are always working with well-formed RA expressions.

Semantics

We give the semantics of (well-formed) RA expressions e by inductively defining the quantity $\#(\bar{a}, e, D)$, which is the number of occurrences of a tuple $\bar{a}$ (of appropriate arity) in the result of applying e to a database D. This is done as follows:

$$
\begin{aligned}
\#(\bar{a},\, R,\, D) &= \#\big(\bar{a}, R^D\big) \\
\#(\bar{a},\, \sigma_{i=j}(e),\, D) &= \begin{cases} \#(\bar{a}, e, D) & \text{if } \bar{a}.i = \bar{a}.j \\ 0 & \text{otherwise} \end{cases} \\
\#(\bar{a},\, \pi_\alpha(e),\, D) &= \sum_{\bar{a}' \colon \pi_\alpha(\bar{a}') = \bar{a}} \#(\bar{a}', e, D) \\
\#\big(\bar{a}\bar{a}',\, e \times e',\, D\big) &= \#\big(\bar{a}, e, D\big) \cdot \#\big(\bar{a}', e', D\big) \\
\#(\bar{a},\, e \uplus e',\, D) &= \#(\bar{a}, e, D) + \#(\bar{a}, e', D) \\
\#(\bar{a},\, e \cup e',\, D) &= \max\big\{\, \#(\bar{a}, e, D),\, \#(\bar{a}, e', D) \,\big\} \\
\#(\bar{a},\, e \cap e',\, D) &= \min\big\{\, \#(\bar{a}, e, D),\, \#(\bar{a}, e', D) \,\big\} \\
\#(\bar{a},\, e - e',\, D) &= \max\big\{\, \#(\bar{a}, e, D) - \#(\bar{a}, e', D),\, 0 \,\big\} \\
\#(\bar{a},\, \varepsilon(e),\, D) &= \min\big\{\, \#(\bar{a}, e, D),\, 1 \,\big\}
\end{aligned}
$$

where $\bar{a}.i$ denotes the i-th element of $\bar{a}$, $\pi_{i_1,\ldots,i_n}(\bar{a})$ is the tuple $(\bar{a}.i_1, \ldots, \bar{a}.i_n)$, and the tuples $\bar{a}$ and $\bar{a}'$ in the rule for $e \times e'$ have the same arity as e and e', respectively.

Then, for an expression e and a database D, we define $e(D)$ as the bag of tuples $\bar{a}$ of the same arity as e so that $\#\big(\bar{a}, e(D)\big) = \#(\bar{a}, e, D)$.

Fragments

The two main fragments of RA we consider are SPC, consisting of selection (σ), projection (π) and Cartesian product ($\times$), and RA$^+$, which is SPC extended with additive union ($\uplus$). Given a fragment $\mathcal{L}$ of RA, we write $\mathcal{L}\{\mathsf{op}_1, \ldots, \mathsf{op}_n\}$ to denote the fragment obtained by adding the RA operations $\mathsf{op}_1, \ldots, \mathsf{op}_n$ to $\mathcal{L}$. Thus, for instance, RA$^+$ is SPC$\{\uplus\}$.

A *query* is a mapping q from databases to bags of tuples. We always assume that queries are generic, that is, invariant under permutations of the domain [1]. A query q is *expressible* in a fragment $\mathcal{L}$ of RA if there is an expression e in that fragment so that $e(D) = q(D)$ for every database D.

Then, given two fragments $\mathcal{L}$ and $\mathcal{L}'$, we say that $\mathcal{L}'$ is *at least as expressive* as $\mathcal{L}$, and write $\mathcal{L} \subseteq \mathcal{L}'$, if every query expressible in $\mathcal{L}$ is also expressible in $\mathcal{L}'$. We say that $\mathcal{L}'$ is *more expressive* than $\mathcal{L}$, and write $\mathcal{L} \subsetneq \mathcal{L}'$, if $\mathcal{L}'$ is at least as expressive as $\mathcal{L}$ and there is a query that is expressible in $\mathcal{L}'$ but not in $\mathcal{L}$. Notice that if $\mathcal{L}'$ has all the operations of $\mathcal{L}$, then $\mathcal{L}'$ is at least as expressive as $\mathcal{L}$.

3 Expressive Power of Bag Relational Algebra Fragments

In this section, we study the relative expressiveness of RA fragments. We present the results that justify the edges in Figure 1, along with additional results that are not explicitly captured in that diagram.

We start by showing that extending positive relational algebra with max-union or intersection results in a more expressive fragment.

▶ **Proposition 1.** $\mathrm{RA}^+ \subsetneq \mathrm{RA}^+\{\star\}$ *for* $\star \in \{\cup, \cap\}$.

Proof sketch. We only need to show that there exists a query that is expressible in $\mathrm{RA}^+\{\star\}$ but not in RA^+. To this end, consider a schema consisting of two nullary (i.e., of arity 0) relation symbols R and S, and suppose that $R \star S$ is expressible in RA^+, i.e., there exists an RA^+ expression e equivalent to $R \star S$. For a database D, let $m = |R^D|$ and $n = |S^D|$; then, $|e(D)| = f_\star(m, n)$, where $f_\cup = \max$ and $f_\cap = \min$, and it is expressible by a polynomial $p_e \in \mathbb{N}[m, n]$, because $e \in \mathrm{RA}^+$ [16]. For $n_0 \in \mathbb{N}$, define the polynomial $p \in \mathbb{N}[m]$ such that $p(m) = p_e(m, n_0)$. When $\star = \cup$, we choose $n_0 = \deg(p_e)$ to derive a contradiction of the fact that $\tilde{p}(m) = d$ is the unique polynomial of degree at most d that interpolates the $d + 1$ data points $(0, d), \ldots, (d, d)$. When $\star = \cap$, we choose $n_0 > 0$ to derive a contradiction of the fact that p is strictly increasing in the interval $[0, +\infty)$, since its coefficients are in $\mathbb{N}$. ◀

The proof of Proposition 1 also applies to show the following.

▶ **Corollary 2.** $\mathrm{SPC} \subsetneq \mathrm{SPC}\{\star\}$ *for* $\star \in \{\cup, \cap\}$.

We now show that additive union increases the expressive power of $\mathrm{SPC}\{\cap\}$ and $\mathrm{SPC}\{\cup\}$.

▶ **Proposition 3.** $\mathrm{SPC}\{\star\} \subsetneq \mathrm{RA}^+\{\star\}$ *for* $\star \in \{\cup, \cap\}$.

Proof sketch. On databases with nullary relations of size 1, every $\mathrm{SPC}\{\star\}$ expression with $\star \in \{\cap, \cup\}$ can only yield a (nullary) relation of size 1, while there are RA^+ expressions (e.g., $R \uplus R$ where R is a base relation) whose results size on such databases is greater than 1. ◀

In particular, the proof of Proposition 3 also applies to show the following.

▶ **Corollary 4.** $\mathrm{SPC} \subsetneq \mathrm{RA}^+$.

Next, we show that $\cup$ increases the expressive power of $\mathrm{RA}^+\{\cap\}$, and $\cap$ increases the expressive power of $\mathrm{RA}^+\{\cup\}$. In turn, this implies that $\cup$ and $\cap$ are incomparable operations.

▶ **Proposition 5.** $\mathrm{RA}^+\{\star\} \subsetneq \mathrm{RA}^+\{\cup, \cap\}$ *for* $\star \in \{\cup, \cap\}$.

Proof sketch.

- $\mathrm{RA}^+\{\cup\} \subsetneq \mathrm{RA}^+\{\cap, \cup\}$

  We show that, on a schema consisting of two nullary relation symbols R and S, $e = R \cap S$ is not expressible in $\mathrm{RA}^+\{\cup\}$. To this end, let e' be an $\mathrm{RA}^+\{\cup\}$ expression over R and S. Then, $|e'(D)|$ is given by a function in the variables $m = |R^D|$ and $n = |S^D|$ consisting of sum, product and max. When e' mentions R, for all databases D where $m > n$ we have that $|e'(D)| \geq m > n = |e(D)|$. When e' mentions S, for all databases D where $m < n$ we have that $|e'(D)| \geq n > m = |e(D)|$. Therefore e' cannot be equivalent to e.

- $\mathrm{RA}^+\{\cap\} \subsetneq \mathrm{RA}^+\{\cap,\cup\}$

  We show that, on a schema consisting of two nullary relation symbols R and S, $e = R \cup S$ is not expressible in $\mathrm{RA}^+\{\cap\}$. To this end, let e' be an $\mathrm{RA}^+\{\cap\}$ expression over R and S. It can be shown that e' can be equivalently rewritten to $e_1 \cap \cdots \cap e_k$ where each e_i is an RA^+ expression. Note that this applies only for nullary relations, which suffices for our purposes, but it does not hold in general. Then, for every database D with $m = |R^D|$ and $n = |S^D|$, we have that $|e(D)| = \max(m,n) = \min(p_1,\ldots,p_k) = |e'(D)|$, where $p_i \in \mathbb{N}^+[m,n]$. Now, let n_0 be an integer greater than 1; then, $\max(m,n_0) = \min(p'_1,\ldots,p'_k)$ where each p'_i is a polynomial in $\mathbb{N}^+[m]$ such that $p'_i(m) = p_i(m,n_0)$. One of these polynomials must be the straight line $p(m) = m$, which leads to a contradiction for $m < n_0$. ◀

▶ **Corollary 6.** *$\mathcal{L}\{\cup\}$ and $\mathcal{L}\{\cap\}$ are incomparable, for $\mathcal{L} \in \{\mathrm{SPC}, \mathrm{RA}^+\}$.*

Finally, we show that with additive union and difference one can express both intersection and max-union, therefore $\mathrm{RA}^+\{-\}$ is the most expressive fragment of RA without duplicate elimination.

▶ **Proposition 7.** $\mathrm{RA}^+\{\cap,\cup\} \subsetneq \mathrm{RA}^+\{-\}$.

Proof sketch. To show that every $e \in \mathrm{RA}^+\{\cap,\cup\}$ can be expressed in $\mathrm{RA}^+\{-\}$, we proceed by induction: the base case is when e is an RA^+ expression, which is trivially in $\mathrm{RA}^+\{-\}$; in the inductive step, we use the fact that, for every $x, y \in \mathbb{N}$, $\min(x,y) = x \dot{-} (y \dot{-} x)$ and $\max(x,y) = x + (y \dot{-} x)$, where $x \dot{-} y = \max(x-y, 0)$ is the *monus* operation. That $\mathrm{RA}^+\{-\}$ is more expressive than $\mathrm{RA}^+\{\cap,\cup\}$ then follows from the fact that every query expressible in $\mathrm{RA}^+\{\cap,\cup\}$ is monotone, while in $\mathrm{RA}^+\{-\}$ one can express non-monotone queries. ◀

Then, obviously, we immediately get the following.

▶ **Corollary 8.** *$\mathrm{RA}^+\{-,\varepsilon\}$ and RA have the same expressive power.*

We conclude this section by showing that adding duplicate elimination to a fragment that does not already have it increases its expressive power.

▶ **Proposition 9.** *$\mathcal{L} \subsetneq \mathcal{L}\{\varepsilon\}$ for every fragment $\mathcal{L}$ of RA without duplicate elimination.*

Proof sketch. On databases with non-empty nullary relations of even size, every $\mathcal{L}$ expression can only yield a (nullary) relation of even size, whereas duplication elimination allows one to obtain, from such databases, relations of size 1. ◀

4 Certain Answers under Bag Semantics

Dealing with incomplete information is a recurring topic in many different areas of logic and computer science. In database theory, the main way to deal with the lack of information is via *incomplete databases*. Intuitively, an incomplete database D is a compact representation of a possibly infinite collection $[\![D]\!]$ of complete databases, which define the *semantics* of D.

In this paper, we use incomplete databases with *marked* (or *labeled*) *nulls*. This model of incompleteness is very common in the database literature [1, 22] and naturally occurs in many different scenarios, e.g., in data exchange and integration (cf. [6, 8, 25]). In this model databases are populated by *constants* and *nulls*, coming from two disjoint and countably infinite sets denoted by Const and Null, respectively. More formally, a k-ary relation is a finite bag of k-ary tuples over $\mathsf{Const} \cup \mathsf{Null}$. A database D then maps each k-ary relation symbol R in the schema to a k-ary bag relation R^D. Given a database $D = \{R_1^D, \ldots, R_n^D\}$, we write

$\mathsf{Const}(D)$ and $\mathsf{Null}(D)$ for the set of constants and nulls occurring in the R_i^Ds, respectively. The *active domain* of D is the set $\mathsf{Const}(D) \cup \mathsf{Null}(D)$, denoted by $\mathsf{adom}(D)$. We say that D is *complete* if $\mathsf{Null}(D) = \varnothing$.

The semantics $[\![D]\!]$ of a database D is defined by means of valuations. A valuation v is a map $v\colon \mathsf{Null}(D) \to \mathsf{Const}$, and the result of applying v to D is the complete database vD obtained by replacing each null $\bot \in \mathsf{Null}(D)$ with $v(\bot)$. Observe that applying v to each relation R^D in D preserves multiplicities, i.e., for each relation name R in the schema and each tuple $\bar{c} \in R^{vD}$ the following equality holds: $\#\big(\bar{c}, R^{vD}\big) = \sum_{\bar{a}\colon v\bar{a}=\bar{c}} \#\big(\bar{a}, R^D\big)$. The set $[\![D]\!]$ is defined as $[\![D]\!] = \{vD \mid v \text{ is a valuation}\}$.

When relations are sets, the standard way to answer a query q on an incomplete database D is to compute *certain answers*, i.e., tuples that are in $q(D)$ for every $D' \in [\![D]\!]$, and *possible answers*, i.e., tuples that are in $q(D)$ for some $D' \in [\![D]\!]$. When relations are bags, however, one must also take multiplicities into account. In what follows, this is done by computing the minimum and maximum number of occurrences of a tuple in the answers across all databases in $[\![D]\!]$ (cf. [14]). Let D be a database, let q be a relational algebra expression of arity n, and let $\bar{a} \in \mathsf{Const}(D)^n$ be a tuple of constants, we define $\min(\bar{a}, q, D)$ and $\max(\bar{a}, q, D)$ as follows:

$$\min(\bar{a}, q, D) = \min_{vD \in [\![D]\!]} \#\big(\bar{a}, q(vD)\big) \quad ; \quad \max(\bar{a}, q, D) = \max_{vD \in [\![D]\!]} \#\big(\bar{a}, q(vD)\big) \tag{1}$$

Intuitively, $\min(\bar{a}, q, D)$ and $\max(\bar{a}, q, D)$ are extensions of certain and possible answers to bag databases. Indeed, $\min(\bar{a}, q, D) \geq 1$ if and only if $\bar{a}$ is in $q(D')$ for every $D' \in [\![D]\!]$ (and thus it is a certain answer), and $\max(\bar{a}, q, D) \geq 1$ if $\bar{a}$ is in $q(D')$ for some $D' \in [\![D]\!]$ (and thus it is a possible answer).

Thus, from now on we assume $\min(\bar{a}, q, D)$ and $\max(\bar{a}, q, D)$ as our standard notion of query answers and study their complexity. More specifically, we will focus on *data complexity*, that is, computing $\min(\bar{a}, q, D)$ and $\max(\bar{a}, q, D)$ for a fixed query q. Depending on the type of comparison we use, several decision problems arise from the computation of min and max. These decision problems are defined as follows.

PROBLEM:	$\mathsf{MIN}^\theta[q]$, for $\theta \in \{>, =, <\}$ and a query q of arity n.
INPUTS:	an incomplete database D, a tuple $\bar{a} \in \mathsf{Const}(D)^n$, a non-negative integer k.
QUESTION:	is $\min(\bar{a}, q, D)\ \theta\ k$?

PROBLEM:	$\mathsf{MAX}^\theta[q]$, for $\theta \in \{>, =, <\}$ and a query q of arity n.
INPUTS:	an incomplete database D, a tuple $\bar{a} \in \mathsf{Const}(D)^n$, a non-negative integer k.
QUESTION:	is $\max(\bar{a}, q, D)\ \theta\ k$?

Whether the number k is represented in unary or binary form does not matter: the results will be the same regardless. All tractability results are shown assuming binary representation, and all matching hardness results will be proved for the case when k is represented in unary. The choice of inequalities, i.e, $<$ vs $\leq$ or $>$ vs $\geq$, is not important: since k is an integer, $\leq k$ is the same condition as $< k + 1$.

As for the case of certain and possible answers, the complexity of the above problems depends on the fragment of RA in which the query q is expressed. While for some of these fragments the problems can be proved to be intractable, there are fragments of RA for which computing $\min(\bar{a}, q, D)$ is tractable and can actually be done via *naive evaluation*. The naive evaluation of a query q of arity n on a bag database D is defined as the bag obtained by assuming that each null value in $\mathsf{Null}(D)$ is a distinct constant and evaluating q directly over D. In what follows, we will denote by $\mathsf{naive}(\bar{a}, q, D)$ the number of occurrences of a

tuple $\bar{a} \in \mathsf{Const}(D)^n$ in the result of the naive evaluation of an RA query q on an incomplete database D. It is well known that $\mathsf{naive}(\bar{a}, q, D)$ can be computed in DLogSpace in data complexity [18, 26].

5 Complexity of Certain Answers

We now turn our attention to the complexity of evaluating bag relational algebra expressions on incomplete databases, that is, solving the problems $\mathsf{MIN}^\theta[q]$ and $\mathsf{MAX}^\theta[q]$ for queries in various fragments of RA. As already explained, these problems are natural bag analogs of the notions of certainty and possibility over set databases.

In Section 5.1, we first provide upper and lower bounds for full RA. When the relation θ is $<$ or $>$, the results are easily derivable from the results for the set case in [2]; we complement them with the exact complexity for equality.

Then, in Section 5.2, we focus on the problem $\mathsf{MIN}^\theta[q]$ and prove the exact tractability boundary shown in Figure 1. We start by showing that in all of the fragments up to $\mathrm{RA}^+\{\cap\}$, the value of min can be computed by naive evaluation of queries, which extends a result in [14]. We then show that outside this fragment the problem is intractable, namely NP-complete for $<$, coNP-complete for $>$, and DP-complete for $=$. In particular, we show that the problem is intractable for all fragments containing $\mathrm{SPC}\{\cup\}$, and all fragments containing $\mathrm{SPC}\{\varepsilon\}$.

Next, in Section 5.3, we look at $\mathsf{MAX}^\theta[q]$. It was shown in [14] that for $>$ the problem is NP-complete, even for very simple queries. Here, we complete the picture and settle the case for $=$, even when q is a query that simply returns a relation from the database.

Finally, in Section 5.4, we discuss what happens with more complex selection conditions in queries.

5.1 Upper and lower bounds for full RA

Before delving into the complexity of the different fragments of RA, we briefly look at the complexity of evaluating general expressions. First, observe that RA queries are *generic*, i.e., invariant under permutations of the domain. In the bag case, this is stated as follows.

▶ **Proposition 10.** *Let D and D' be* complete *databases, let ρ be a bijection between $\mathsf{adom}(D)$ and $\mathsf{adom}(D')$ such that $D' = \rho(D)$, and let $q \in$ RA be a query of arity n. Then, $\#(\bar{a}, q, D) = \#(\rho\bar{a}, q, D')$ for every tuple $\bar{a} \in \mathsf{adom}(D)^n$.*

Intuitively, this tells us that we need to take into account only finitely many valuations in order to compute the values in (1). Upper bounds of NP, coNP, and DP follow straightforwardly.

▶ **Proposition 11.** *Let q be an expression in* RA*. Then:*
- $\mathsf{MIN}^<[q]$ *and* $\mathsf{MAX}^>[q]$ *are in* NP*;*
- $\mathsf{MIN}^>[q]$ *and* $\mathsf{MAX}^<[q]$ *are in* coNP*;*
- $\mathsf{MIN}^=[q]$ *and* $\mathsf{MAX}^=[q]$ *are in* DP*.*

Proof sketch. Due to Proposition 10, in order to compute min and max we need to take into account only a limited number of valuations. Like in the set case [2], for a given database D we need to take into account only those valuations whose range consists of $\mathsf{Const}(D)$ and a new distinct constant for each null in $\mathsf{Null}(D)$. Moreover, evaluating RA expressions over complete bag semantics databases is in DLogSpace in data complexity. ◀

For general RA expressions, all of these problems are complete in their respective classes.

▶ **Proposition 12.** *Let q be an expression in* RA*. Then:*

- $\mathsf{MIN}^{<}[q]$ *and* $\mathsf{MAX}^{>}[q]$ *are* NP*-hard;*
- $\mathsf{MIN}^{>}[q]$ *and* $\mathsf{MAX}^{<}[q]$ *are* coNP*-hard;*
- $\mathsf{MIN}^{=}[q]$ *and* $\mathsf{MAX}^{=}[q]$ *are* DP*-hard.*

Proof sketch. The results for $<$ and $>$ follows directly from the fact that set databases can be simulated by bag databases. Hence, the hardness results presented in [2] apply. For $=$, we can show a reductions from a very well known DP-complete problem (see, e.g., Theorem 20 for the case of $\mathsf{MIN}^{=}[q]$ and Theorem 25 for the case of $\mathsf{MAX}^{=}[q]$). ◀

Despite these high bounds, one may expect that some fragments of RA will behave better; this is what we investigate next.

5.2 Computing min

We now look at computing certain answers to bag queries, that is, values $\min(\bar{a}, q, D)$. The usual naive evaluation works for queries in the $\mathrm{RA}^{+}\{\cap\}$ fragment. Outside it, the decision version of the problem, $\mathsf{MIN}^{<}[q]$, is intractable for every fragment, and for each such fragment the complexity is exactly the same. We now prove these facts.

5.2.1 Naive evaluation for $\mathrm{RA}^{+}\{\cap\}$

While computing min is hard in the general case, there exists a large fragment of RA for which min can be computed via naive evaluation. In the set case this fragment is well known to be positive relational algebra [22] consisting of selection, projection, Cartesian product, and union. Notice that over sets the intersection of two relations is expressible by join, but over bags this is no longer the case. It turns out that the good behavior of join with respect to certain answers extends to bags, when we add intersection explicitly. Indeed, for $\mathrm{RA}^{+}\{\cap\}$, one can compute min simply by using naive evaluation.

▶ **Theorem 13.** *Let q be an* $\mathrm{RA}^{+}\{\cap\}$ *expression of arity n, let D be a database, and let* $\bar{a} \in \mathsf{Const}(D)^n$*. Then,* $\min(\bar{a}, q, D) = \mathsf{naive}(\bar{a}, q, D)$*.*

Proof sketch. One can show that for every valuation v of $\mathsf{Null}(D)$ we have

$$\sum_{\bar{b}\colon v\bar{b}=v\bar{a}} \mathsf{naive}(\bar{b}, q, D) \leq \#(v\bar{a}, q, vD)\,.$$

Since the number of occurrences of a tuple $\bar{a}$ in a relation is always a non-negative integer, we have $\mathsf{naive}(\bar{a}, q, D) \leq \#(\bar{a}, q, vD)$. The claim then follows from the fact that whenever $\mathsf{naive}(\bar{a}, q, D) = k$ there exists a valuation v of $\mathsf{Null}(D)$ such that $\#(\bar{a}, q, vD) = k$. ◀

From the fact that bag relational algebra queries are in DLogSpace with respect to data complexity [18, 26], we then immediately get the following.

▶ **Corollary 14.** *For expressions* $q \in \mathrm{RA}^{+}\{\cap\}$*, the problems* $\mathsf{MIN}^{<}[q]$*,* $\mathsf{MIN}^{>}[q]$ *and* $\mathsf{MIN}^{=}[q]$ *are all in* DLogSpace*.*

5.2.2 Hardness for SPC{∪} and fragments with duplicate elimination

We now show that $\mathrm{RA}^+\{\cap\}$ is the best fragment for which we can compute certain answers efficiently under bag semantics. From the diagram in Figure 1, it suffices to prove that the problem is intractable for SPC{∪}, as well as for all fragments with duplicate elimination. This is what we do here; in fact we shall see that in all of these fragments, the intractable complexity will be exactly the same, namely NP-complete, coNP-complete, or DP-complete, when θ is $<$, or $>$, or $=$.

Technical tools

To prove these results, we use reductions from two well-known decision problems: *satisfiability* (SAT) and *satisfiability-unsatisfiability* (SAT-UNSAT) of propositional formulae in conjunctive normal form (CNF). Recall that a k-CNF formula ϕ is the conjunction of clauses, where each clause is a disjunction of at most k literals, i.e., variables or their negations. In what follows, we assume k-CNF formulae with exactly k distinct literals in each clause. In our scenario, this can be done without loss of generality. We write $\mathsf{Var}(\phi)$ for the set of variables of ϕ, and $|\phi|$ for the number of clauses of ϕ. The formula ϕ is *satisfiable* if there exists an assignment for $\mathsf{Var}(\phi)$ that satisfies all the clauses of ϕ. We let k-SAT refer to the problem of checking whether a CNF formula is satisfiable; this problem is well known to be NP-complete for $k \geq 3$. Given two k-CNF formulae ϕ and ψ, SAT-UNSAT is the problem of checking whether ϕ is satisfiable while ψ is not. For two given formulae $\phi, \psi \in k$-CNF, for $k \geq 3$, checking whether ϕ is satisfiable and ψ is not satisfiable is DP-complete [29].

For our reductions, we will use two additional technical tools: an encoding of k-CNF formulae as relations and a CNF formula with specific properties. Let ϕ be a propositional k-CNF formula whose clauses are $c_1, \ldots, c_n$. Assume an injection ρ from $\mathsf{Var}(\phi)$ to Null, for each $x \in \mathsf{Var}(\phi)$, the tuples $\bar{u}^t_x$ and $\bar{u}^f_x$ are defined respectively as $(0, \rho(x))$ and $(\rho(x), 1)$.

We next associate a relation R_i with each clause c_i, for $i \in \{1, \ldots, n\}$. To this end, we assume a linear ordering over $\mathsf{Var}(\phi)$. Consider a truth assignment τ for the variables of c_i, i.e., a mapping from the k variables used in c_i to *true* and *false* that makes c_i true. For each such assignment τ, we add to the relation R_i a single occurrence of the tuple $(\bar{u}_1 \ldots \bar{u}_k)$, where each $\bar{u}_j$ is equal to $\bar{u}^t_{x_j}$ if $\tau(x_j) = \mathit{true}$ and to $\bar{u}^f_{x_j}$ if $\tau(x_j) = \mathit{false}$, and the variables are considered in the ordering we assumed on $\mathsf{Var}(\phi)$. Finally, we define R_ϕ as $\biguplus_{i=1}^n R_i$.

Observe that the number of tuples in each R_i is exactly 2^k, hence the total number of tuples in R_ϕ is at most $|\phi| \cdot 2^k$. When k is fixed, the size of R_ϕ is polynomial with respect to the size of ϕ. Relations R_ϕ enjoy the following property that will be central in our proofs.

▶ **Lemma 15.** *Let ϕ be a k-CNF formula. There exists a truth assignment of $\mathsf{Var}(\phi)$ that satisfies exactly m clauses of ϕ if and only if there exits a valuation v of $\mathsf{Null}(R_\phi)$ such that* $\mathrm{range}(v) \subseteq \{0,1\}$ *and* $\#\big((0,1)^k, vR_\phi\big) = m$.

Proof sketch. First, one can show that for each clause c_i of ϕ and for every valuation v of $\mathsf{Null}(R_i)$ there is at most one occurrence of $(0,1)^k$ in vR_i. Observe now that, for each $\bar{u} \in R_i$, $v\bar{u} = (0,1)^k$ means that either $v\big(\rho(x)\big) = 1$ for a positive literal x appearing in c_i, or $v\big(\rho(x)\big) = 0$ for a negative literal $\neg x$ appearing in c_i. If there exists a valuation v of $\mathsf{Null}(R_\phi)$ such that $\mathrm{range}(v) \subseteq \{0,1\}$ and $\#\big((0,1)^k, vR_\phi\big) = m$, the following truth assignment satisfies m clauses of ϕ: $\tau(x) = \mathit{true}$ if $v(\rho(x)) = 1$, and $\tau(x) = \mathit{false}$ otherwise. Suppose now that a truth assignment τ for $\mathsf{Var}(\phi)$ that satisfies m clauses of ϕ exists. The valuation v for $\mathsf{Null}(D)$ is such that $\#\big((0,1)^k, vR_\phi\big) = m$: $v\big(\rho(x)\big) = 1$ if $\tau(x) = \mathit{true}$, and $v\big(\rho(x)\big) = 0$ otherwise. ◀

Our second technical tool is a formula $h(f,g)$ derived from two k-CNF formulae

$$f = \bigwedge_{i=1}^{n} \left(f_i^1 \vee \cdots \vee f_i^k\right) \qquad \text{and} \qquad g = \bigwedge_{i=1}^{m} \left(g_i^1 \vee \cdots \vee g_i^k\right)$$

where each f_i^j and g_i^j is a literal. Let x and y be two propositional variables not appearing in $\mathsf{Var}(f) \cup \mathsf{Var}(g)$; then, $h(f,g)$ is the following $(k+1)$-CNF formula:

$$\bigwedge_{i=1}^{n} \left(f_i^1 \vee \cdots \vee f_i^k \vee x\right) \wedge \bigwedge_{i=1}^{m} \left(g_i^1 \vee \cdots \vee g_i^k \vee \neg x\right) \wedge \bigwedge_{i=1}^{m} \left(g_i^1 \vee \cdots \vee g_i^k \vee \neg y\right) \wedge (x) \wedge (\neg x \vee y)$$

The formula $h(f,g)$ can be used to check whether f is satisfiable while g is not, due to the following property.

▶ **Lemma 16.** *Let f and g be two k-CNF formulae. Then f is satisfiable and g is unsatisfiable if and only if the maximum number of clauses of $h(f,g)$ that can be satisfied by a single truth assignment is exactly $|h(f,g)| - 1$.*

The SPC{∪} case

We now look at the complexity of handling the max-union operator ∪. Somewhat unexpectedly, adding ∪ to SPC gives rise to a substantial increase in the complexity of computing min; recall that in the set case, adding union is harmless and preserves the property that certain answers can be found by naive evaluation. To prove this claim, we will use reductions from SAT and SAT-UNSAT.

Let ϕ be a k-CNF formula, and let D_ϕ denote the database $\{D^R, D^T\}$ where D^R is R_ϕ (i.e., the encoding of CNF formulae presented earlier) and D^T contains an occurrence of $(0,1)^k$ for each clause in ϕ, that is, $\#\big((0,1)^k, D^T\big) = |\phi|$.

For reductions, we use the query $q = \pi_\varnothing(R \cup T)$. Note that we project onto the empty set of attributes, so the only possible answers for q are either the empty bag, or bags containing one or more occurrences of the empty tuple $()$.

To prove our claims, we first need to prove the following result.

▶ **Proposition 17.** *For every k-CNF formula ϕ, the database D_ϕ has the following properties:*

1. *There exists a truth assignment for $\mathsf{Var}(\phi)$ that satisfies exactly m clauses of ϕ if and only if there exits a valuation v for $\mathsf{Null}(D_\phi)$ such that $\mathrm{range}(v) \subseteq \{0,1\}$ and $\#\big((), q, vD_\phi\big) = \big((2^k+1)\cdot|\phi|\big) - m$.*
2. *For every valuation v of $\mathsf{Null}(D_\phi)$ there exists a valuation v' for $\mathsf{Null}(D_\phi)$ such that $\mathrm{range}(v') \subseteq \{0,1\}$ and $\#\big((), q, vD_\phi\big) \geq \#\big((), q, v'D_\phi\big)$.*

Proof sketch. The first property can be proved using Lemma 15. For the second, observe that $\#\big((), q, vD_\phi\big)$ depends on the number of occurrences of $(0,1)^k$ in vR_ϕ. ◀

With these notions in place, we are now ready to present the main results of this section. We start by proving that $\mathsf{MIN}^<[q]$ is NP-hard.

▶ **Theorem 18.** *The problem $\mathsf{MIN}^<[q]$ is NP-hard for $q = \pi_\varnothing(R \cup T)$.*

Proof sketch. To prove the claim, we show a reduction from SAT. Let ϕ be a 3-CNF formula. From Proposition 17, we get $\min\big((), q, D_\phi\big) = 9|\phi| - m$, where m is the maximum number of clauses of ϕ that can be satisfied by the same truth assignment. To obtain the reduction, simply observe that satisfiability of ϕ means that all the $|\phi|$ clauses of ϕ can be satisfied with the same truth assignment. In light of this, we can conclude that ϕ is satisfiable if and only if $\min\big((), q, D_\phi\big) < 9|\phi| - |\phi| + 1 = 8|\phi| + 1$. In turn, this gives the desired reduction. ◀

The complexity of $\mathsf{MIN}^{>}[q]$ follows directly from Theorem 18, being $\mathsf{MIN}^{<}[q]$ the complement of $\mathsf{MIN}^{>}[q]$.

▶ **Corollary 19.** *There exists a query $q \in \mathrm{SPC}\{\cup\}$ such that $\mathsf{MIN}^{>}[q]$ is* coNP*-hard.*

We now turn our attention to $\mathsf{MIN}^{=}[q]$ and show that this problem is hard for $q \in \mathrm{SPC}\{\cup\}$.

▶ **Theorem 20.** *The problem $\mathsf{MIN}^{=}[q]$ is* DP*-hard for $q = \pi_{\varnothing}(R \cup T)$.*

Proof sketch. To prove the claim we discuss a reduction from SAT-UNSAT. Let f and g be two 3-CNF formulae, let $h(f,g)$ be the formula defined earlier, and let D_h be the encoding of $h(f,g)$ defined above. Using Proposition 17, one can prove that exactly $|h(f,g)| - 1$ clauses of $h(f,g)$ can be satisfied by the same truth assignment if and only if there exists a valuation v of $\mathsf{Null}(D_h)$ such that $\#\big((), q, vD_h\big) = 16|h(f,g)|+1$ and there exists no valuation v' of $\mathsf{Null}(D_h)$ such that $\#(v'D_h, q, ()) \leq 16|h(f,g)|$. By Lemma 16, this implies that f is satisfiable and g is unsatisfiable if and only if $\min\big((), q, D_h\big) = 16|h(f,g)| + 1$, proving the claim. ◀

Handling duplicate elimination

In terms of tractability, no fragment survives the addition of duplicate elimination. In this section, we will show that the decision problems for all fragments from $\mathrm{SPC}\{\varepsilon\}$ to the full relational algebra are NP-complete, coNP-complete, and DP-complete for $<$, $>$ and $=$, respectively. To prove this, we will use reductions from SAT and SAT-UNSAT.

Let ϕ be a k-CNF formula, and let $D_\phi = \{D^S\}$ be the database where the relation D^S is defined as follows. Let R_i be the relation encoding the i-th clause of ϕ, as described at the beginning of this section; then define the relation $R'_i = (\{i\} \times R_i) \uplus \{\big(i, (0,1)^k\big)\}$, where $\{i\}$ and $\{\big(i, (0,1)^k\big)\}$ are bags containing one occurrence of the tuples (i) and $(i, \underbrace{0,1,\ldots,0,1}_{2k})$, respectively. We define $D^S = \biguplus_i R'_i$. Notice that the size of D^S is at most $(2^k + 1) \cdot |\phi|$.

In the proofs we use the very simple $\mathrm{SPC}\{\varepsilon\}$ query $q = \pi_{\varnothing}\big(\varepsilon(S)\big)$, whose output is either the empty set or the set $\{()\}$ containing the empty tuple.

To prove our complexity results we need the following proposition.

▶ **Proposition 21.** *For every k-CNF formula ϕ, the database D_ϕ has the following properties:*

1. *There exists an assignment for $\mathsf{Var}(\phi)$ that satisfies m clauses of ϕ if and only if there exists a valuation v of $\mathsf{Null}(D_\phi)$ such that $\mathrm{range}(v) \subseteq \{0,1\}$ and $\#\big((), q, vD_\phi\big) = \big((2^k + 1) \cdot |\phi|\big) - m$.*
2. *For every valuation v of $\mathsf{Null}(D_\phi)$ there is a valuation v' of $\mathsf{Null}(D_\phi)$ such that $\mathrm{range}(v') \subseteq \{0,1\}$ and $\#\big((), q, vD_\phi\big) \geq \#\big((), q, v'D_\phi\big)$.*

Proof sketch. The first property can be proved using Lemma 15. For the second, observe that $\#\big((), q, vD_\phi\big)$ depends on the number of occurrences of $(0,1)^k$ in vR_ϕ. ◀

Using Proposition 21 we can prove the main results of this section. We start with the complexity of $\mathsf{MIN}^{<}[q]$.

▶ **Theorem 22.** *The problem $\mathsf{MIN}^{<}[q]$ is* NP*-hard for $q = \pi_{\varnothing}(\varepsilon(S))$.*

Proof sketch. To prove the claim we show a reduction from SAT. Let ϕ be a 3-CNF formula and let D_ϕ be the database encoding ϕ defined above. Using Proposition 21, we can prove that $\min\big((), q, D_\phi\big) = 9|\phi| - m$, where m is the maximum number of clauses of ϕ that can be satisfied by the same truth assignment. To obtain the reduction, observe that if ϕ is

satisfiable then $|\phi|$ of its clauses can be satisfied by the same truth assignment. In turn, this proves that ϕ is satisfiable if and only if $\min\big((), q, D_\phi\big) < 9|\phi| - |\phi| + 1 = 8|\phi| + 1$. The desired reduction follows. ◀

The complexity of $\mathsf{MIN}^{>}[q]$ follows directly from Theorem 22.

▶ **Corollary 23.** *There exists a query $q \in \mathrm{SPC}\{\varepsilon\}$ such that $\mathsf{MIN}^{>}[q]$ is* coNP*-hard.*

We now look at $\mathsf{MIN}^{=}[q]$ and prove the desired lower bound for $q \in \mathrm{SPC}\{\varepsilon\}$.

▶ **Theorem 24.** *The problem $\mathsf{MIN}^{=}[q]$ is* DP*-hard for $q = \pi_\varnothing(\varepsilon(S))$.*

Proof sketch. To prove the claim, we argue that there exists a reduction from SAT-UNSAT to $\mathsf{MIN}^{=}[q]$ when $q = \pi_\varnothing(\varepsilon(S))$. Let f and g be two 3-CNF formulae, let $h(f, g)$ be the formula defined in the previous sections, and let D_h be the encoding of $h(f, g)$ as relation defined above. Using Proposition 21, one can prove that $|h(f, g)| - 1$ clauses of $h(f, g)$ can be satisfied by the same truth assignment if and only if there exists a valuation v of $\mathsf{Null}(D_h)$ such that $\#\big((), q, vD_h\big) = 16|h(f, g)| + 1$ and there exists no valuation v' of $\mathsf{Null}(D_h)$ such that $\#\big((), q, v'D_h\big) \leq 16|h(f, g)|$. Due to Lemma 16, this implies that f is satisfiable and g is not satisfiable if and only if $\min((), q, D_h) = 16|h(f, g)| + 1$. In turn, this proves the desired reduction. ◀

5.3 Computing max: hardness for simple queries

Computing max is hard already for very simple queries: in fact, $\mathsf{MAX}^{>}[q]$ is NP-complete for queries that simply return a base relation in the database [14]. Here we complete the picture and prove that solving $\mathsf{MAX}^{=}[q]$ for the same queries is complete for DP. To this end, we will show a reduction from SAT-UNSAT to $\mathsf{MAX}^{=}[q]$.

Let ϕ be a k-CNF formula, and let D_ϕ be the database consisting of the single relation $D^R = R_\phi$, which is the encoding of ϕ defined earlier. Using this simple construction we can prove the following.

▶ **Theorem 25.** *The problem $\mathsf{MAX}^{=}[q]$ is* DP*-hard even for a query q that returns a base relation in the database.*

Proof sketch. To prove the claim, we discuss a reduction from SAT-UNSAT. Let f and g be two 3-CNF formulae, and let $h(f, g)$ be the 4-CNF formula defined earlier. Consider the database D_h defined above and assume that the query q returns R. First, one can prove that for every valuation v of $\mathsf{Null}(D_h)$ there exists a valuation v' such that $\mathrm{range}(v') \subseteq \{0, 1\}$ and $\#\big((0, 1)^k, R, vD_h\big) \leq \#\big((0, 1)^k, R, v'D_h\big)$. From this considerations and Lemma 15, one can prove that exactly $|h(f, g)| - 1$ clauses of $h(f, g)$ can be satisfied by the same truth assignment if and only if there exists a valuation v of $\mathsf{Null}(D_h)$ such that $\#\big((0, 1)^4, R, vD_h\big) = |h(f, g)| - 1$ and there exists no valuation v' of $\mathsf{Null}(D_h)$ such that $\#\big((0, 1)^4, R, v'D_h\big) \geq |h(f, g)|$. Due to Lemma 16, f is satisfiable and g is unsatisfiable if and only if $\max\big((0, 1)^k, R, D_h\big) = |h(f, g)| - 1$. In turn, this proves the desired reduction. ◀

5.4 Complex selection conditions

In our definition of relational algebra, we assumed that selection conditions are equalities $i = j$. More generally, one can define these conditions by the grammar

$$c \;:=\; (i = j) \mid c \wedge c \mid c \vee c \mid \neg c$$

allowing arbitrary Boolean combinations. We briefly describe how more complex conditions affect our results.

If we simply add conjunction, that is, selection conditions are conjunctions of equalities, nothing changes at all. This is because the cascade of selections rule, $\sigma_{c \wedge c'}(e) = \sigma_c\big(\sigma_{c'}(e)\big)$, applies under bag semantics as well.

If disjunction is added, it might appear at first that this is the same as adding max-union, because $\sigma_{c \vee c'}(e) = \sigma_c(e) \cup \sigma_{c'}(e)$. As far as finding certain answers is concerned, the fragment SPC$\{\cup\}$ is intractable, so it appears that adding disjunction to selection conditions, under bag semantics, might lead to intractability. However, this is not the case: adding disjunction to selections is weaker than adding max-union, and preserves tractability.

▶ **Proposition 26.** *Let q be a query in* RA^+ *or arity n where selection conditions are positive Boolean combinations of equalities. Then,* $\mathsf{MIN}^<[q]$ *and* $\mathsf{MIN}^>[q]$ *can be solved in* DLogSpace. *More precisely, for every database D and every tuple $\bar{a} \in \mathsf{Const}(D)^n$, the value* $\min(\bar{a}, q, D)$ *can be computed by naive evaluation:* $\min(D, q, \bar{a}) = \mathsf{naive}(D, q, \bar{a})$.

For conditions that are unrestricted Boolean combinations of equalities, the problem becomes intractable, even when these are added to the positive fragment.

▶ **Proposition 27.** *In the extension of* RA^+ *that allows arbitrary Boolean combinations of equalities as selection conditions, there exist queries q such that* $\mathsf{MIN}^<[q]$ *is* NP*-complete,* $\mathsf{MIN}^>[q]$ *is* coNP*-complete, and* $\mathsf{MIN}^=[q]$ *is* DP*-complete.*

6 Conclusions

We have provided two complete classifications: of the expressive power of fragments of bag relational algebra, and of the complexity of computing certain and possible answers in those fragments. For the complexity of certain answers, we have a dichotomy: either they can be computed efficiently by naive evaluation, or their complexity is intractable, which means NP-complete, or coNP-complete, or DP-complete (depending on how the problem is turned into a decision problem).

Directions for future work are motivated by the recent work on bag semantics in data management applications where incompleteness naturally occurs, such as data exchange [20] and OBDA [28]. Notice that we have primarily concentrated on the closed-world semantics, which as of late has been actively studied in those contexts; see, e.g., [3, 5, 19, 21, 27]. Thus we believe our results could be relevant to understanding the complexity of these applications under the closed-world assumption. As another direction for future work, we would like to study the complexity of finding certain and possible answers in the fragments of bag relational algebra under the open-world assumption. The general case is of course undecidable, but the picture for the fragments studied here is not clear. Finally, we would like to use our results as the starting point for the study of answering queries with grouping and aggregation over incomplete data, as such queries rely on bag semantics.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

2 Serge Abiteboul, Paris Kanellakis, and Gösta Grahne. On the representation and querying of sets of possible worlds. *Theoretical Computer Science*, 78(1):158–187, 1991.

3 Shqiponja Ahmetaj, Magdalena Ortiz, and Mantas Simkus. Polynomial Datalog Rewritings for Expressive Description Logics with Closed Predicates. In *IJCAI*, pages 878–885, 2016.

4 Joseph Albert. Algebraic Properties of Bag Data Types. In *VLDB*, pages 211–219, 1991.
5 Giovanni Amendola, Nicola Leone, Marco Manna, and Pierfrancesco Veltri. Enhancing Existential Rules by Closed-World Variables. In *IJCAI*, pages 1676–1682, 2018.
6 Marcelo Arenas, Pablo Barceló, Leonid Libkin, and Filip Murlak. *Foundations of Data Exchange*. Cambridge University Press, 2014.
7 Leopoldo E. Bertossi, Georg Gottlob, and Reinhard Pichler. Datalog: Bag Semantics via Set Semantics. *CoRR*, abs/1803.06445, 2018. `arXiv:1803.06445`.
8 Meghyn Bienvenu and Magdalena Ortiz. Ontology-Mediated Query Answering with Data-Tractable Description Logics. In *Reasoning Web*, pages 218–307, 2015.
9 Peter Buneman, Shamim A. Naqvi, Val Tannen, and Limsoon Wong. Principles of Programming with Complex Objects and Collection Types. *Theor. Comput. Sci.*, 149(1):3–48, 1995.
10 R. G. G. Cattell. *The Object Database Standard: ODMG-93*. Morgan Kaufmann, 1993.
11 Surajit Chaudhuri and Moshe Y. Vardi. Optimization of *Real* Conjunctive Queries. In *PODS*, pages 59–70, 1993.
12 Sara Cohen. Equivalence of queries combining set and bag-set semantics. In *PODS*, pages 70–79, 2006.
13 Latha S. Colby and Leonid Libkin. Tractable Iteration Mechanisms for Bag Languages. In *ICDT*, pages 461–475, 1997.
14 Marco Console, Paolo Guagliardo, and Leonid Libkin. On Querying Incomplete Information in Databases under Bag Semantics. In *IJCAI*, pages 993–999. ijcai.org, 2017.
15 C. J. Date and Hugh Darwen. *A Guide to the SQL Standard*. Addison-Wesley, 1996.
16 Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, pages 31–40. ACM, 2007.
17 Stéphane Grumbach, Leonid Libkin, Tova Milo, and Limsoon Wong. Query languages for bags: expressive power and complexity. *SIGACT News*, 27(2):30–44, 1996.
18 Stéphane Grumbach and Tova Milo. Towards Tractable Algebras for Bags. *J. Comput. Syst. Sci.*, 52(3):570–588, 1996.
19 André Hernich. Answering Non-Monotonic Queries in Relational Data Exchange. *Logical Methods in Computer Science*, 7(3), 2011.
20 André Hernich and Phokion G. Kolaitis. Foundations of information integration under bag semantics. In *LICS*, pages 1–12. IEEE Computer Society, 2017.
21 André Hernich, Leonid Libkin, and Nicole Schweikardt. Closed world data exchange. *ACM Trans. Database Syst.*, 36(2):14:1–14:40, 2011.
22 Tomasz Imielinski and Witold Lipski. Incomplete information in relational databases. *Journal of the ACM*, 31(4):761–791, 1984.
23 T. S. Jayram, Phokion G. Kolaitis, and Erik Vee. The containment problem for real conjunctive queries with inequalities. In *PODS*, pages 80–89, 2006.
24 Phokion G. Kolaitis. The Query Containment Problem: Set Semantics vs. Bag Semantics. In *AMW*, 2013.
25 Maurizio Lenzerini. Data integration: a theoretical perspective. In *PODS*, pages 233–246, 2002.
26 Leonid Libkin and Limsoon Wong. Query Languages for Bags and Aggregate Functions. *J. Comput. Syst. Sci.*, 55(2):241–272, 1997.
27 Carsten Lutz, Inanç Seylan, and Frank Wolter. Ontology-Mediated Queries with Closed Predicates. In *IJCAI*, pages 3120–3126, 2015.
28 Charalampos Nikolaou, Egor V. Kostylev, George Konstantinidis, Mark Kaminski, Bernardo Cuenca Grau, and Ian Horrocks. The Bag Semantics of Ontology-Based Data Access. In *IJCAI*, pages 1224–1230, 2017.
29 Christos H. Papadimitriou and Mihalis Yannakakis. The Complexity of Facets (and Some Facets of Complexity). *J. Comput. Syst. Sci.*, 28(2):244–259, 1984.
30 Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. McGraw-Hill, 2003.

Categorical Range Reporting with Frequencies

Arnab Ganguly
Dept. of Computer Science, University of Wisconsin, Whitewater, USA
gangulya@uww.edu

J. Ian Munro
Cheriton School of Computer Science, University of Waterloo, Canada
imunro@uwaterloo.ca

Yakov Nekrich
Cheriton School of Computer Science, University of Waterloo, Canada
yakov.nekrich@googlemail.com

Rahul Shah
Dept. of Computer Science, Baton Rouge, USA
rahul@csc.lsu.edu

Sharma V. Thankachan
Dept. of Computer Science, University of Central Florida
sharma.thankachan@ucf.edu

Abstract

In this paper, we consider a variant of the color range reporting problem called *color reporting with frequencies*. Our goal is to pre-process a set of colored points into a data structure, so that given a query range Q, we can report all colors that appear in Q, along with their respective frequencies. In other words, for each reported color, we also output the number of times it occurs in Q. We describe an external-memory data structure that uses $O(N(1+\log^2 D/\log N))$ words and answers one-dimensional queries in $O(1+K/B)$ I/Os, where N is the total number of points in the data structure, D is the total number of colors in the data structure, K is the number of reported colors, and B is the block size.

Next we turn to an approximate version of this problem: report all colors σ that appear in the query range; for every reported color, we provide a constant-factor approximation on its frequency. We consider color reporting with approximate frequencies in two dimensions. Our data structure uses $O(N)$ space and answers two-dimensional queries in $O(\log_B N + \log^* B + K/B)$ I/Os in the special case when the query range is bounded on two sides. As a corollary, we can also answer one-dimensional approximate queries within the same time and space bounds.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases Data Structures, Range Reporting, Range Counting, Categorical Range Reporting, Orthogonal Range Query

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.9

1 Introduction

Orthogonal range query is a fundamental problem in computational geometry and data structures, with a large number of applications in database theory, information retrieval, and text mining. In this problem, we want to pre-process a set of points so that later, given a query rectangle Q, we can report some desired information about the points contained in Q. Typical examples of this information are: the *maximum* (or *minimum*) value of a point in the range (in this case we associate a numeric value to each point), the most-frequent value in the range (or *mode*), the *median* value, the number of points in the range, etc.

In the *colored (or categorical) orthogonal range reporting problem*, every point is assigned a color (category). Given a query range Q, we must report distinct categories of points in the query range. Typically, we want to avoid reporting the same category many times. For this reason, colored variants of the range reporting problem are considered to be more difficult. In the *orthogonal range counting problem*, we keep the set of points in a data structure so that, given a query range Q, we can count the number of points in Q.

In this paper, we consider a variant of color range reporting queries that combines color reporting with counting: *given a query range Q, we report all colors that appear in Q; additionally, for each color within Q, we also report the number of its occurrences in Q.* Henceforth, such queries are called *color reporting with frequencies*. We also consider an approximate variant: *all colors in a query range Q are reported; for each color σ, which appears K_σ times in Q, we find a constant factor approximation for K_σ.*

We describe two data structures – one for color reporting with exact frequencies in one dimension and the other for color reporting with approximate frequencies in two dimensions, which are the first of their kind in the external memory model.

1.1 Applications to Databases

Situations where we need to report distinct categories of objects in a query range arise frequently in database applications. Suppose we have a database of employees, and we want to identify all employees whose salary lies in the range \$40,000 – \$80,000. This problem corresponds to one-dimensional point reporting queries. Now suppose that we want to report different job positions, such that at least one employee holding this position earns between \$40,000 and \$80,000. This problem corresponds to one-dimensional color reporting queries. In many cases, we may be interested in a more involved query: *group all employees who earn between \$40,000 and \$80,000 according to their job positions, then list all groups and the number of employees in each group.* This problem can be modeled by (one-dimensional) color reporting queries with frequencies. Furthermore we may be satisfied if we obtain approximate numbers of employees for every group. In this case, it is sufficient to answer the approximate variant of the color reporting query with frequencies.

Another prominent application of color range queries is in databases containing a collection of texts (documents). Given a query pattern Q, we want to report all documents that contain Q at least once; for each such document, we want to estimate how many occurrences of Q it contains. Such a query can be reduced to a color reporting query with frequencies.

1.2 Model of Computation

We study the color reporting with (approximate) frequencies problems in the external memory model. Here, the storage is divided into a main (or internal) memory of limited size and a large (potentially unbounded) external disk. We can perform arithmetic operations on data that is stored in the main memory. The data structure, however, does not fit into main memory and must be stored on the external disk. Using one I/O operation, we can read a contiguous block of B words from disk into the main memory or write a block of B words from main memory into the external disk. The main memory is much faster than the external memory; hence, all main memory operations are considered free and the algorithm complexity is measured in the number of I/O operations.

We remark that when we answer a reporting query in the external memory model, it is preferable to report $\Theta(B)$ objects (points, colors) with one I/O operation. Design of methods that fully exploit this property of the external memory model poses an additional challenge.

1.3 Notation

Throughout this paper, N is the total number of points, where each point is associated with a color. We denote by D the number of distinct colors. The number of colors in the answer to a query is denoted by K. The number of points with color σ in the answer to a query (or the frequency of σ) is denoted by K_σ. We assume that a word consists of $\Theta(\log N)$ bits. Unless specified otherwise, the space usage is measured in words of $\Theta(\log N)$ bits.

1.4 Previous and Related Work

Due to its importance, both in theory and in practice, the problem of reporting distinct colors of points in a range has received considerable attention. Both one-dimensional and multi-dimensional color (or categorical) range reporting were considered in the literature [13, 7, 11, 12, 8, 18, 19, 14, 23]. Other variants of this problem, such as top-k color reporting [14, 20], color maxima queries [25], color stabbing queries [10] were also studied.

Gupta et al. [11] present a main memory data structure that uses linear-space (i.e., $O(N)$-word) and answers one-dimensional colored range reporting queries in $O(\log N + K)$ time. Muthukrishnan [18] showed that the query time can be improved to $O(1 + K)$ if points are in the rank space, i.e., point coordinates are integers $1, 2, \ldots, N$. Later, Nekrich and Vitter [24] removed the rank space assumption, without any penalty in space or time.

In the external memory model, one-dimensional color reporting is usually solved by reducing it to three-sided point reporting (i.e., to a special case of two-dimensional point reporting when the query range is bounded on three sides). Arge et al. [3] described a linear-space solution that supports queries in $O(\log_B N + K/B)$ I/Os. Nekrich [21] described an $O(N)$-word data structure that supports queries in $O(\log\log_B U + K/B)$ I/Os when point coordinates are bounded by U. If point coordinates are bounded by N, another $O(N)$-space data structure from [21] supports queries in $O(\log_2^{(h)} N + K/B)$ I/Os, where h is a positive constant and $\log^{(h)} N$ denotes the logarithm function iterated h times.[1] Larsen and Pagh [15] showed that one-dimensional queries can be answered in $O(1 + K/B)$ I/Os using $O(N)$ words if point coordinates are bounded by N. They also studied the related problem of reporting the top-k colors, where each color is associated with a static weight and the k highest weighted colors within the query range have to be reported. Nekrich and Vitter [24] demonstrated that one-dimensional color reporting queries can be answered in $O(1 + K/B)$ I/Os even if point coordinates are arbitrary integers. Patil et al. [25] considered another variant of color reporting in one-dimension, called *colored range maxima queries*; here, one has to report the points corresponding to k distinct colors within the query range having the highest y-coordinate values. They presented multiple data structures with different space-and-time trade-offs, such as an $O(N)$-word data structure with $O(\log^* N + k/B)$ query time[2] and an $O(N \log^* N)$-word data structure with optimal $O(1 + k/B)$ query time.

Although this problem has been studied extensively and optimal results for many colored range reporting problems were achieved, limited progress has been made for the *color range reporting with frequencies* problem. Gupta, Janardan, and Smid [11] presented an internal memory data structure that answers a query in $O(\log N + K)$ time and uses $O(N \log N)$ space; see the type-2 counting problem in [11]. If we combine a result of Muthukrishnan [18] with an idea of Sadakane [29], we can obtain an $O(N)$-word data structure for reporting colored points and their frequencies in the rank space in $O(1 + K)$ time. However, this result

[1] $\log^{(h)} N = \log(\log^{(h-1)} N)$ for $h > 1$ and $\log^{(1)} N = \log N$

[2] $\log^* N$ is the smallest number i such that $\log^{(i)} N \leq 2$

Table 1 Our Results.

Query Type	Space	Time
1D/2D Two-sided with Constant Appx	$O(N)$ words	$O(\log^* B + \log_B N + K/B)$ I/Os
1D One-sided Exact	$O(N)$ words	$O(1 + K/B)$ I/Os
1D Two-sided Exact	$O(N(1 + \log^2 D/\log N))$ words	$O(1 + K/B)$ I/Os

is not efficient in the external memory model, where the desired dependency on the output size is $O(f(N) + K/B)$ for some small function $f(N)$. To the best of our knowledge, there is no previous solution that exploits the properties of the external memory model. In contrast, the standard color reporting (without frequencies) can be solved in optimal $O(1 + K/B)$ time and $O(N)$ space in external memory [15, 24]. In this paper, we bridge this gap.

In the related approximate range counting problem, we must approximate the number of point within the query range. Chan and Wilkinson [9] presented an $O(N \log\log N)$-word data structure that answers approximate two-dimensional queries in $O(\log\log N)$ time. The previous paper by Nekrich [22] achieved a better approximation factor, compared to [9], at the cost of higher space usage. In another paper, Nekrich [23] presented an $O(N)$-space data structure that supports approximate point counting queries on an $N \times N$ grid in $O(1)$ I/Os, when the query range is bounded on three sides.

The second problem that we study combines color reporting with approximate point counting problem: we report all colors within the query range and provide a constant factor approximation for the frequency of each color. We primarily study the problem from the perspective of dominance reporting in two-dimensions, i.e., the query range is bounded on two sides; see e.g., [1] for a catalog on dominance queries. Straightforward reductions from this two-dimensional dominance problem can be used to answer a (two-sided) color reporting query with approximate frequencies in one dimension.

1.5 Our Results

Unless specified otherwise, we assume that points are in the rank space, i.e., point coordinates are integers $1, 2, \ldots, N$. More general scenarios can be reduced to this special case at the cost of increasing the query time by a small additive factor: if points are integers $1, \ldots, U$ for a parameter U, then the query cost increases by $O(\log_2 \log_B U)$; if point coordinates can assume arbitrary values, the query cost increases by $O(\log_B N)$.

We first consider the problem of reporting colors and their frequencies for a two-sided query in 2D, which is commonly referred to as dominance query [1] in 2D. We present a linear-space data structure that answers a constant-factor 2-sided approximate-frequency query: given a query range $[-\infty, \alpha] \times [-\infty, \beta]$, we report all colors in the query range, and for every reported color σ, we also provide an estimate k_σ on the number of its occurrences in the range. Let $\delta > 1$ be a constant that is fixed at the construction time and can be arbitrarily close to 1. For every color σ in the range, we obtain an integer index i, such that $\delta^i \leq k_\sigma \leq \delta^{i+1}$. The total cost of a query is $O(\log^2(N/B) + K/B)$ I/Os. Our data structure is based on a reduction to the problem of identifying all two-dimensional rectangles that contain a query point [4, 27] (rectangle stabbing problem), and uses $O(N)$ space. Then, we improve the time to $O(\log_B N + \log^* B + K/B)$ I/Os, still using linear space. As a straightforward corollary, we answer one-dimensional 2-sided queries with the same space-time trade-offs.

Additionally, we present an $O(N\lceil\frac{\log^2 D}{\log N}\rceil)$-word data structure that answers 1-dimensional 2-sided color reporting queries with exact frequencies in optimal $O(1+K/B)$ I/Os. We remark that the space is $O(N)$ words if the number of colors is not too large, i.e., when $D \leq 2^{\sqrt{\log N}}$. This result is based on an $O(N)$-space data structure that answers 1-dimensional 1-sided color reporting queries with exact frequencies in optimal $O(1 + K/B)$ I/Os.

Our results are presented in Table 1.

1.6 Map

We begin in Section 2 by reducing 2-sided approximate-frequency queries in 2D to rectangle stabbing queries (also known as orthogonal point enclosure). This leads to a simple linear space data structure. In Section 3, we improve this result to achieve our claimed I/O complexity without increasing the space usage. Section 4 explains how to use this data structure for answering approximate-frequency queries in one dimension. In Section 5, we present our data structure for reporting colors with exact frequencies in one dimension. Finally, we conclude in Section 6 with a couple of unanswered problems.

2 2D Color Dominance with Approximate Frequencies

In this section, we focus on answering two-dimensional colored dominance queries with approximate frequencies. Specifically, we consider the following:

▶ **Definition 1.** *Given a collection of two-dimensional colored points, we define* $\mathsf{FREQ}(\alpha, \beta, \sigma)$ *to be the number of 2d points* $p_i = (x_i, y_i)$ *satisfying* $x_i \leq \alpha$, $y_i \leq \beta$, *and the color of* p_i *is* σ.

▶ **Problem 2.** *Given* N *colored points in 2d, answer an* $\mathsf{APPX2D}(\alpha, \beta)$ *query: report all distinct colors* $\sigma_1, \sigma_2, \ldots, \sigma_K$ *with values* $f_1, f_2, \ldots, f_K$ *such that for each reported color* σ_i,

- $\mathsf{FREQ}(\alpha, \beta, \sigma_i) \neq 0$, *and*
- $\mathsf{FREQ}(\alpha, \beta, \sigma_i) \leq f_i \leq \delta \cdot \mathsf{FREQ}(\alpha, \beta, \sigma_i)$

We prove the following theorem in this section.

▶ **Theorem 3.** *There exists an* $O(N)$*-word data structure that answers* $\mathsf{APPX2D}(\alpha, \beta)$ *queries in* $O(\log^2 \frac{N}{B} + K/B)$ *I/Os.*

2.1 Shallow Cutting

We heavily rely on the concept of shallow cutting that has been used extensively in range reporting problems [1, 2, 17, 27, 28]. Consider a color σ and let N_σ be the total number of points of color σ. We call a point a σ-point if it is of color σ.

An h-shallow cutting for a parameter h can be constructed as follows. We start sweeping a vertical line V from $x = -\infty$ until we reach a σ-point such that the total number of σ-points encountered so far is δ^h. Now start sweeping a horizontal line H from $y = \infty$ until we reach the highest σ-point among these δ^h σ-points. Let c_1 be the point of intersection of the sweeping lines at this point. Let $\mathbb{R}_1^h$ be the region bounded by the sweeping lines; note that $\mathbb{R}_1^h$ is two-sided and has δ^h number of σ-points.

Now again start sweeping V to the right, until the region, say $\mathbb{R}_1^{h'}$, bounded by V and H contains δ^{h+1} points. Let c_1' be the point of intersection of the sweeping lines at this point. Start sweeping H downwards until the region, say $\mathbb{R}_2^h$, bounded by V and H again contains

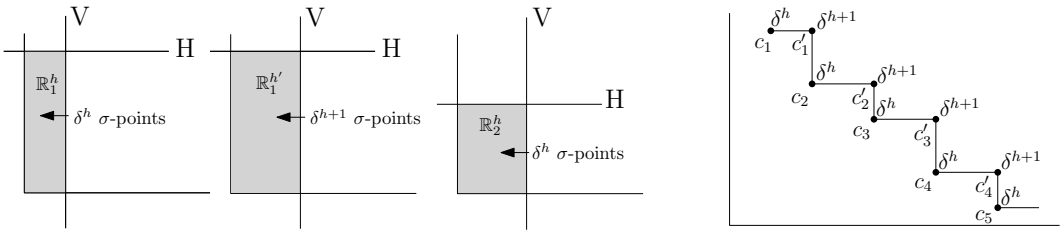


Figure 1 Illustration of Shallow Cutting and the resultant staircase structure.

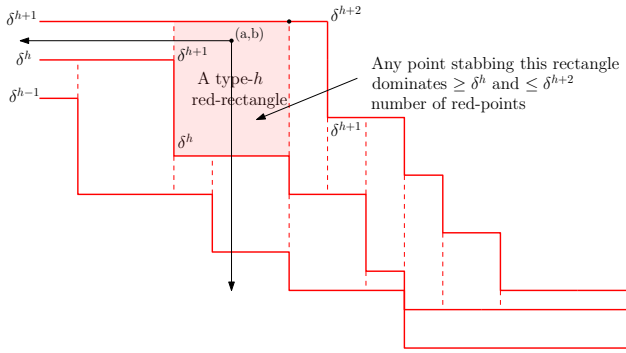


Figure 2 Illustration of Rectangular Tessellation.

δ^h points. Let c_2 be the new point of intersection of V and H. We repeat this process until V reaches the rightmost σ-point within the region bounded by V and H, following which H is swept down so that the last two-sided region contains δ^h many σ-points.

We denote by $\mathbb{R}_1^h, \mathbb{R}_2^h, \ldots,$ type-h σ-regions; likewise, type-h regions refers to the collection of type-h σ-regions over all possible colors σ. The points $c_1, c_1', c_2, c_2', \ldots$ are referred to as the corner points of this "*staircase structure*". We also refer to this staircase structure as the skyline of the regions. See Figure 1.

The following lemma will play a crucial role in proving our space bounds.

▶ **Lemma 4.** *The number of type-h regions is $O(N/\delta^h)$.*

Proof. Refer to Figure 1. It is easy to see that every time a new region $\mathbb{R}_i^h$ is created from $\mathbb{R}_{i-1}^h$, we read $\delta^{h+1} - \delta^h$ many new σ-points. Hence, we create at most $\frac{N_\sigma}{(\delta^{h+1}-\delta^h)} = \frac{N_\sigma}{(\delta-1)\delta^h}$ σ-regions, which is $O(N_\sigma/\delta^h)$. Thus, the number of type-h σ-regions is $O(N_\sigma/\delta^h)$. The lemma follows trivially. ◀

2.2 Tessellation Using Rectangles

Let us consider the shallow cutting outlined in the previous section. For each color σ, we carry out the shallow cutting for every $h \in \left[1, \lceil \log_\delta N \rceil\right]$, thereby breaking the xy-plane into various σ-regions – type-1 σ-regions, type-2 σ-regions, $\cdots$, type-$(\log_\delta N)$ σ-regions.

We now present the following important lemma (see Figure 3).

▶ **Lemma 5.** *A type-h σ-region is contained within a type-$(h+1)$ σ-region.*

Proof. Refer to Figure 3. It shows the situation when a type-h σ-region is not contained in a type-$(h+1)$ σ-region. Since the number of σ-points dominated by both B and D is δ^{h+1}, there are no σ-points either to the left of the segment AD, or below the segment CD, or within the rectangle $ABCD$. But then our construction is incorrect and the points D and C must have coincided with A and B respectively. The claim follows by contradiction. ◀

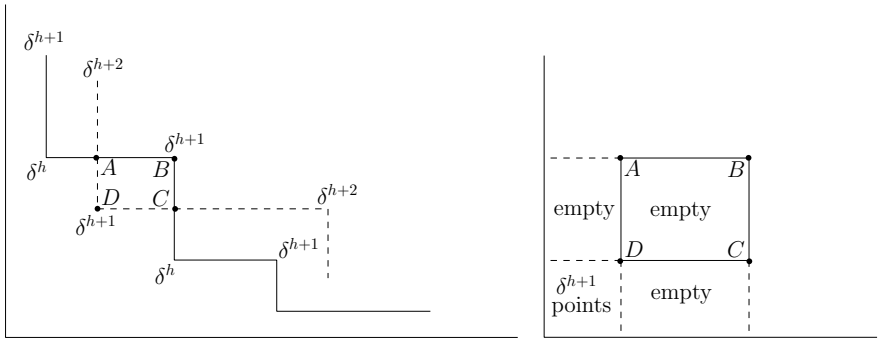


Figure 3 Illustration of Lemma 5. A type-h region is contained withing a type-$(h+1)$ region.

Based on the above lemma, the "*staircase structure*" of the $(h+1)$-shallow cutting lies to the right and above of the staircase structure for the h-shallow cutting.

Let us consider a point p in the plane such that it is contained in a type-$(h+1)$ σ-region but not contained in a type-h σ-region. This, as we shall see in more details in the next subsection, gives an estimate of the number of σ-points that are dominated by p. Our main idea is to associate such a point p with an approximate count of the σ-points that are dominated by it. Obviously, we cannot associate each point separately; hence, we break the xy-plane between a type-h and a type-$(h+1)$ region into several (yet appropriately bounded) rectangles. Specifically, consider the corner points of a type-h σ-region. We draw vertical lines from these points onto the skyline of the type-$(h-1)$ and type-$(h+1)$ σ-regions, thereby breaking the plane into several rectangles. See Figure 2.

We call each rectangle a type-h σ-rectangle if it is contained in a type-$(h+1)$ σ-region but not within a type-(h) σ-region. As before, type-h rectangles refers to the collection of type-h σ-rectangles over all possible colors σ. Using Lemma 4, we can show that the number of type-h rectangles for a fixed h is $O(N/\delta^h)$. The following lemma is obtained by summing over $h = 1, 2, \ldots, \lceil \log_\delta N \rceil$.

▶ **Lemma 6.** *The total number of rectangles is $O(N)$.*

2.3 Rectangle Stabbing

A rectangle is said to be stabbed by a point if it contains the point within its boundaries. The rectangle tessellation structure described in the previous section immediately leads to the following important lemma.

▶ **Lemma 7.** *If a type-h σ-rectangle is stabbed by (α, β), then the following inequality holds:*

$$\delta^h \leq \mathsf{FREQ}(\alpha, \beta, \sigma) \leq \delta^{h+2}$$

Proof. Consider the shallow cutting in Figure 2. Since the point (α, β) stabs a type-h σ-rectangle, it is contained within a type-$(h+1)$ σ-region. Thus, the number of σ-points dominated by any point on the skyline of a type-$(h+1)$ σ-region is at most δ^{h+2}. Hence, $\mathsf{FREQ}(\alpha, \beta, \sigma) \leq \delta^{h+2}$. To prove the other inequality, we observe in Figure 2 that any point within a type-$(h+1)$ σ-region dominates at most δ^h σ-points of a type-h σ-region. Hence, $\mathsf{FREQ}(\alpha, \beta, \sigma) \geq \delta^h$, as required. ◀

Intuitively, looking at the lemma above, it follows that in order to answer an $\mathsf{APPX2D}(\alpha, \beta)$ query, it suffices to report all the type-h σ-rectangles that are stabbed by (α, β) for all possible choices of σ and appropriate choices of h in each case. This is because if a type-h σ-rectangle is stabbed by a point (α, β), we have the following:

- there exists a σ-point dominated by (α, β), which follows simply from the way in which the rectangles are constructed, and
- $\delta^h \leq \mathsf{FREQ}(\alpha, \beta, \sigma) \leq \delta^{h+2}$, using Lemma 7

We notice that for each particular color there is at most one stabbed rectangle of that color. Therefore for each color, our task is to find the rectangle of that color (if any) that is stabbed by (α, β); for each stabbed rectangle, report its color and an appropriate real value which gives an approximation of $\mathsf{FREQ}(\alpha, \beta, \sigma)$.

To this end, we collect all the rectangles of all possible colors and store them in the following external-memory rectangle-stabbing data structure in [4].

▶ **Fact 8** (Theorem 4, [4]). *Given M rectangles, there exists an $O(M)$ space data structure that can report all rectangles stabbed by an input point in $O(\log^2(M/B) + occ/B)$ I/Os, where occ is the output size.*

With each type-h σ-rectangle, say R^h_σ, we store its corresponding color σ, and the value $\mathsf{COUNT}(R^h_\sigma) = \delta^{h+2}$.

Using Fact 8 and Lemma 6, the total space is $O(N)$ words.

2.4 Wrapping Up

To answer $\mathsf{APPX2D}(\alpha, \beta)$, we query the data structure of Fact 8 and report all the stabbed rectangles along with their color and $\mathsf{COUNT}(\cdot)$ values. The query time is $O(\log^2(N/B) + K/B)$, as desired.

Note that each color is reported exactly once, because only one rectangle of a particular color is stabbed; hence, we are left to show that for a reported type-h σ-rectangle R^h_σ, $\mathsf{COUNT}(R^h_\sigma)$ gives us a desired approximation of $\mathsf{FREQ}(\alpha, \beta, \sigma)$.

First we note that $\mathsf{COUNT}(R^h_\sigma) \geq \mathsf{FREQ}(\alpha, \beta, \sigma)$

Revisiting Lemma 7, we see that

$$\mathsf{FREQ}(\alpha, \beta, \sigma) \geq \delta^h = \frac{\mathsf{COUNT}(R^h_\sigma)}{\delta^2} \implies \mathsf{COUNT}(R^h_\sigma) \leq \delta^2 \cdot \mathsf{FREQ}(\alpha, \beta, \sigma)$$

By choosing $\sqrt{\delta}$ (instead of δ) during construction, we obtain

$$\mathsf{COUNT}(R^h_\sigma) \leq \delta \cdot \mathsf{FREQ}(\alpha, \beta, \sigma)$$

which leads to our desired approximation ratio. (Note that space and time bounds are only affected by a constant factor.) This completes the proof of Theorem 3.

3 Improvement via Bootstrapping

In this section, we set out to improve the query time of Theorem 3, still using linear space. More specifically, we prove the following theorem.

▶ **Theorem 9.** *There exists an $O(N)$-word data structure that answers $\mathsf{APPX2D}(\alpha, \beta)$ queries in $O(\log_B N + \log^* B + K/B)$ I/Os.*

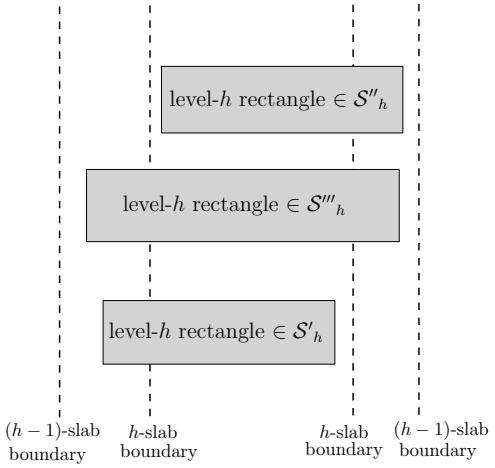


Figure 4 Illustration of type-h rectangles and the sets $\mathcal{S}'_h$, $\mathcal{S}''_h$, and S'''_h.

3.1 Framework and Overview

In order to prove Theorem 9, we need to improve the $\log^2(N/B)$ factor in the I/O complexity of Theorem 3. Unfortunately, as shown by Arge, Samoladas, and Yi [4], it is not possible to achieve $o(\log_2(N/B))$ query complexity for the rectangle stabbing problem in two dimensions if the data structure uses linear space. However, in our case, we are able to circumvent the lower bound of [4] because rectangles have some additional special properties. Our approach is based on a hierarchical subdivison of the plane into vertical slabs. The subdivision is based on two parameters:

$$\Delta_h = B(\log^{(h)}(N/B))^4 \text{ and } K_h = B(\log^{(h)}(N/B))^2$$

A slab boundary is a vertical line through $x = i + 0.5$ for some integer $i \in [1, N]$ and slab is the region between two slab-boundaries. The entire xy-plane is a level-0 slab. For $h = 1, 2, 3, \ldots, \log^* N$, we divide each level-$(h-1)$ slab into level h-slabs, so that

- the number of level-h slabs within any level-$(h-1)$ slab is $O(\Delta_{h-1}/\Delta_h)$
- the number of rectangles completely covered by any level-h slab is $O(\Delta_h)$.

A rectangle is a level-h rectangle if it crosses at least one boundary of a level-h slab, but is completely contained within a level-$(h-1)$ slab. See Figure 4.

Throughout this section we will denote by $\mathcal{S}_i$ the i-slab that contains the query point (α, β) for $i = 1, 2, \ldots, \log^* n$. Consider a particular h. Suppose that a point (α, β) stabs K rectangles, where $K \geq K_h$. All the level-h rectangles are contained in the $(h-1)$-slab $\mathcal{S}_{h-1}$. If we keep all level-h rectangles contained in $\mathcal{S}_{h-1}$ in the data structure of Fact 8, then all level-h rectangles stabbed by (α, β) are reported in $O(\log^2(\Delta_{h-1}/B) + K/B) = O(K_h/B + K/B) = O(K/B)$ I/Os. This case is considered in detail in Section 3.2.

The main difficulty is in reporting level-h rectangles for values of h, such that $K < K_h$. In this case, described in Section 3.3, we follow a different strategy. We keep all level-h rectangles that span the slab $\mathcal{S}_h$ (i.e., rectangles that intersect $\mathcal{S}_h$ but have all four corners outside of $\mathcal{S}_h$) in a separate data structure and use it to report all level-h rectangles that are stabbed by (α, β). Since (α, β) is within $\mathcal{S}_h$, we only need to look for rectangles whose y-projection contains β. In the general case, this approach would require too much space: a

level-h rectangle can span a large number of h-slabs and we would have to keep it in a data structure for every slab. Fortunately in our case, rectangles satisfy a so-called *monotonic property*, defined in Observation 15. Due to this property, we only need to store $o(\Delta_h)$ "lowest" spanning rectangles for every h-slab. Any point above those "lowest" rectangles is guaranteed to stab at least K_h rectangles. Additionally, we also need to classify rectangles into heavy and light ones, according to the frequencies of regions represented by these rectangles. A detailed description is provided in Section 3.3.2. We consider separately the case of the level-h rectangles that intersect but do not span $\mathcal{S}_h$. Queries on such rectangles can be reduced to three-dimensional dominance queries; see Section 3.3.1.

In the description above, we assumed that the value of K is known. Computing the value of K is rather straightforward – simply report the colors of all points dominated by (α, β). Using the result of Patil et al. [25], we can report all colors (and compute K) in $O(\log^* N + K/B)$ I/Os using an $O(N)$ space data structure.

3.2 Handling Case 1: $K \geq K_h$

For $h = 1, 2, 3, \ldots$ and for each level-$(h-1)$ slab $\mathbb{D}$, we create a rectangle stabbing structure of Fact 8 over all level-h rectangles that are completely within $\mathbb{D}$. By our choice of parameters, the number of such rectangles is $O(\Delta_{h-1})$. By definition of level-h rectangles, every rectangle is stored in one data structure; hence, the total space is $O(N)$ words.

To report all level-h rectangles stabbed by (α, β), we first identify the level-$(h-1)$ slab $\mathbb{D}_{h-1}$ that is stabbed by the point (α, β) using the following lemma.

▶ **Lemma 10.** *For any $h \in [1, \log^* N]$, we can identify the level-$(h-1)$ slab that is stabbed by a point in $O(1)$ I/Os by using an $O(N)$ space data structure.*

Proof. For each h, maintain a bit vector $B_h[1, N]$, such that $B_h[i] = 1$ iff $i + 0.5$ is a level-h slab boundary. Additionally, associate an $o(N)$-bit structure for constant rank/select support on B_h [26]. Clearly, the level-h slab stabbed by (α, β) is the one that starts at $t + 0.5$, where t is the rightmost position $\leq \beta$, such that $B_h[t] = 1$. The number of I/Os required is $O(1)$. The total space is $O(N \log^* N)$ bits, which is $O(N)$ words. ◀

Now, we consider a query over the rectangle stabbing structure associated with $\mathbb{D}_{h-1}$. Suppose the number of rectangles reported is occ_{h-1}. Then, the number of I/Os required is

$$O(\log^2(\Delta_{h-1}/B) + occ_{h-1}/B) \text{ i.e., } O((K_h + occ_{h-1})/B)$$

Therefore the total number of I/Os over all values of h, where $K \geq K_h$, is

$$O\left(\frac{1}{B} \sum_{h, K \geq K_h} (K_h + occ_{h-1})\right) \text{ i.e., } O(\log^* N + K/B)$$

In summary, we have the following lemma.

▶ **Lemma 11.** *If $K_h \leq K$, then all level-h rectangles that are stabbed by a point (α, β) can be retrieved in $O(\log^* N + K/B)$ I/Os by using an $O(N)$ space data structure.*

3.3 Handling Case 2: $K < K_h$

Let $\mathcal{S}_h$ be the set of all level-h rectangles intersecting with a level-h slab, say $\mathbb{D}_h$. Note that by definition, they all are completely within a level-$(h-1)$ slab, say $\mathbb{D}_{h-1}$, hence $|\mathcal{S}_h| = O(\Delta_{h-1})$. We divide $\mathcal{S}_h$ into three disjoint sets, $\mathcal{S}_h = \mathcal{S}'_h \cup \mathcal{S}''_h \cup \mathcal{S}'''_h$, defined as follows.

1. $\mathcal{S}'_h$: rectangles in $\mathcal{S}_h$ with both right corners inside $\mathbb{D}_h$ and both left corners outside $\mathbb{D}_h$.
2. $\mathcal{S}''_h$: rectangles in $\mathcal{S}_h$ with both left corners inside $\mathbb{D}_h$ and both right corners outside $\mathbb{D}_h$.
3. $\mathcal{S}'''_h$: rectangles in $\mathcal{S}_h$ with all 4 corners outside $\mathbb{D}_h$.

See Figure 4. We will answer stabbing queries for every subset of $\mathcal{S}_h$ separately.

3.3.1 Handling $\mathcal{S}'_h$ and $\mathcal{S}''_h$

We show how to find the rectangles that belong to $\mathcal{S}'_h$ and are stabbed by (α, β); the rectangles in $\mathcal{S}''_h$ can be found in a symmetric way.

Observe that since the right corners of any rectangle in $\mathcal{S}'_h$ are within the slab, a rectangle $[x_l, x_r] \times [y_b, y_t]$ is stabbed when $\alpha \leq x_r$ and $y_b \leq \beta \leq y_t$. Based on this intuition, we recall the following three-dimensional dominance reporting structure of Afshani [1].

▶ **Fact 12.** *Given M three-dimensional points, there exists an $O(M)$ space data structure that can report all the occ points dominated by a query point q in $O(\log_B M + occ/B)$ I/Os.*

Using the above result, we prove the following lemma.

▶ **Lemma 13.** *We can report all rectangles from sets $\mathcal{S}'_h$ and $\mathcal{S}''_h$, $1 \leq h \leq \log^* N$, stabbed by a query point (α, β) in $O(\log^* N + \log_B N + K/B)$ I/Os using an $O(N)$ space data structure.*

Proof. Based on the data structure in Fact 12, we represent a rectangle $R = [x_l, x_r] \times [y_b, y_t]$ in $\mathcal{S}'_h$ as a three-dimensional point $(-x_r, y_b, -y_t)$. The rectangle is stabbed by (α, β) iff $(-\alpha, \beta, -\beta)$ dominates $(-x_r, y_b, -y_t)$. For each $h \in [1, \log^* N]$, we maintain two dominance structures – one for the rectangles in $\mathcal{S}'_h$ and another for the rectangles in $\mathcal{S}''_h$. The total space over all levels is $O(N)$. Since $|\mathcal{S}'_h \cup \mathcal{S}''_h| = O(\Delta_h)$, if the number of rectangles reported at level h is occ_h, the number of I/Os is

$$O\left(\log_B(\Delta_h) + \frac{occ_h}{B}\right) \text{ i.e., } O\left(1 + \log_B(\log^{(h)} N) + \frac{occ_h}{B}\right)$$

The total number of I/Os for $h = 1, 2, \ldots, \log^* N$ is $O(\log^* N + \log_B N + \frac{K}{B})$. ◀

3.3.2 Handling $\mathcal{S}'''_h$

What remains is to show how to obtain the rectangles in $\mathcal{S}'''_h$ stabbed by the query. The idea is to divide $\mathcal{S}'''_h$ into two carefully selected subsets:

- a set $\mathcal{H}_h$ of heavy rectangles
- a set $\mathcal{L}_h$ of light rectangles

Heavy Rectangles. The set $\mathcal{H}_h$ contains all rectangles R in $\mathcal{S}'''_h$ with $\mathsf{COUNT}(R) \geq \frac{\Delta_{h-1}}{B}$, where $\mathsf{COUNT}(R)$ is the frequency associated with R. We have the following lemma.

▶ **Lemma 14.** *For any $h \in [1, \log^* N]$, the number of heavy rectangles in $\mathcal{H}_h$ is $O(B)$.*

Proof. Consider a level-h σ-rectangle $R = [x', x''] \times [y', y'']$. Let $\mathsf{COUNT}(R) = f$. Then, the number of σ-points in $[x', x'']$ is $\Theta(f)$, which follows from our shallow cutting construction. Therefore within a level-$(h-1)$ slab $\mathbb{D}$, the number of points is Δ_{h-1} and the number of level-h heavy rectangles completely within $\mathbb{D}$ is bounded by $\Delta_{h-1}/f = O(B)$. ◀

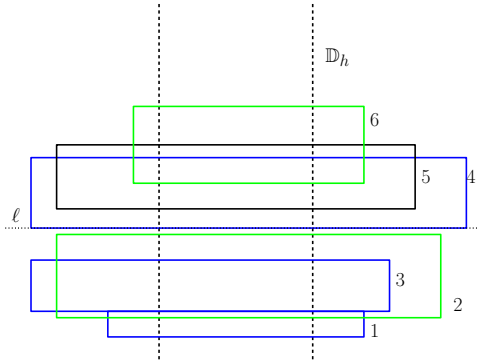


Figure 5 Example of $\mathcal{L}_h$ for $K_h = 3$. Rectangles spanning $\mathbb{D}_h$ with bottom segment below ℓ (i.e., 1, 2, 3, and 4) are included in $\mathcal{L}_h$. Every point within $\mathbb{D}_h$ and above ℓ stabs at least K_h many rectangles.

We can store these rectangles in space $\sum_h \left(\frac{N}{\Delta_h} \cdot B\right) = O(N)$. When a query is answered, we load all the rectangles in $\mathcal{H}_h$ into the internal memory using $O(1)$ I/Os, and then examine them in the internal memory to find the ones stabbed by (α, β). Hence, the total number of I/Os over all h is bounded by $O(\log^* N)$.

Light Rectangles. To create the set $\mathcal{L}_h \subseteq S_h'''$ of light rectangles, we sweep a horizontal line ℓ from $-\infty$ in the positive y-direction. Every time when ℓ crosses the bottom line of a rectangle from S_h''', we add its color to the set of colors, say $\mathcal{C}_h$. We stop when $\mathcal{C}_h$ contains K_h distinct colors (or when ℓ reaches the highest rectangle in S_h'''). See Figure 5. Then $\mathcal{L}_h$ is the set of all rectangles R in $\mathcal{S}_h'''$ with color $c \in \mathcal{C}_h$ and the $\mathsf{COUNT}(R) < \Delta_{h-1}/B$. Recall that $\mathsf{COUNT}(R)$ is the frequency associated with a rectangle R, and they are of the form $\delta, \delta^2, \ldots$. Therefore, the number of rectangles having a color from $\mathcal{C}_h$, which is associated with a particular $\mathsf{COUNT}(\cdot)$ is $\lceil \log_\delta(\Delta_{h-1}/B) \rceil$. Hence, the size of $\mathcal{L}_h$ is $|\mathcal{L}_h| = K_h \cdot \log_\delta(\Delta_{h-1}/B) = O(B(\log^{(h)} n)^3)$.

Since every rectangle in $\mathcal{L}_h$ spans $\mathbb{D}_h$, for our purposes they can be represented by their projections on the y-axis, i.e., x-coordinates can be ignored. To check whether a rectangle $[x_l, x_r] \times [y_b, y_t]$ in $\mathcal{L}_h$ is stabbed by (α, β), it suffices to check whether $y_b \le \beta \le y_t$. We can find all $[y_b, y_t]$ stabbed by β in $O(\log_B |\mathcal{L}_h|)$ I/Os [5].

Hence, all rectangles in $\mathcal{L}_h$ that are stabbed by (α, β) can be obtained in

$$O\left(\log_B |\mathcal{L}_h| + \frac{occ_h}{B}\right) = O\left(1 + \log_B(\log^{(h)} n) + \frac{occ_h}{B}\right)$$

I/Os, where occ_h is the number of such rectangles. Therefore the total number of I/Os summed over $h \in [1, \log^* N]$ is $O(\log^* N + \log_B N + K/B)$ as desired.

The total space needed to store all sets $\mathcal{L}_h$ is

$$O\left(\sum_h \frac{nK_h}{\Delta_h} \cdot \log \frac{\Delta_{h-1}}{B}\right) = O\left(\sum_h \frac{N}{\log^{(h)} \frac{N}{B}}\right) = O(N)$$

Now we will show that it is sufficient to examine rectangles in $\mathcal{L}_h$ if $K < K_h$.

▶ **Observation 15** (Monotonic Stabbing). *If a point (x, y) stabs a rectangle with color c, then any point (x, y'), where $y' > y$, stabs a rectangle with color c. Hence, the output (colors) of a stabbing query (x, y) is a subset of the output of a stabbing query (x, y') for $y' > y$.*

Let y_h denote the final y-coordinate of ℓ (i.e., the position of ℓ when $\mathcal{C}_h$ contains K_h colors, see Fig. 5). If a rectangle $R \in S_h'''$ stabs a point $(x, y) \in \mathbb{D}_h$ and $y \leq y_h$, then $R \in \mathcal{L}_h$. Consider an arbitrary point $(x, y) \in \mathbb{D}_h$ with $y \geq y_h$. For every color in $c \in \mathcal{C}_h$, there is a point $(x, y') \in \mathbb{D}_h$ such that $y' \leq y_h$ and (x, y') stabs a rectangle of color c. Hence, by Observation 15, (x, y) also stabs a rectangle of color c. Therefore any point $(x, y) \in \mathbb{D}_h$, such that $y \geq y_h$, stabs at least K_h different rectangles. We obtain the following lemma.

▶ **Lemma 16.** *We can report all rectangles from sets S_h''', $1 \leq h \leq \log^* N$, that are stabbed by a query point (α, β) in $O(\log^* N + \log_B N + K/B)$ I/Os using an $O(N)$ space data structure.*

3.4 Wrapping Up

Combining Lemmas 11, 13 and 16, the total space is $O(N)$ words and the total number of I/Os needed is $O(\log^* N + \log_B N + K/B)$, which is $O(\log^* B + \log_B N + K/B)$.[3] This completes the proof of Theorem 9.

4 1D Approximate Color Reporting

We now consider the following query: *reporting colors in one dimension with their approximate frequencies*, i.e., we report all the colors within the query range $[\alpha, \beta]$; however, instead of reporting the exact frequency f_{exact} for a color σ that lies in the query range, we report a real value f_σ such that $f_{exact} \leq f_\sigma \leq \delta \cdot f_{exact}$, where $\delta > 1$ is an arbitrary small constant. We reduce these queries to $\mathsf{APPX2D}(\alpha, \beta)$ queries in 2D discussed in Sections 2 and 3: represent a 1D point x as $(-x, x)$ in 2D; an approximate color frequency query in one dimension, denoted by $\mathsf{APPX1D}(\alpha, \beta)$, is answered using the query $\mathsf{APPX2D}(-\alpha, \beta)$. Hence, the following is an immediate consequence of Theorem 9.

▶ **Theorem 17.** *Given N one-dimensional colored points, we can answer an $\mathsf{APPX1D}(\alpha, \beta)$ query in $O(\log^* B + \log_B N + K/B)$ I/Os using an $O(N)$ space data structure.*

5 1D Exact Color Reporting

In this section, we answer one-dimensional color reporting queries with exact frequencies.

▶ **Problem 18.** *Given N one-dimensional colored points, answer an $\mathsf{EXACT1D}(\alpha, \beta)$ query defined as: report all distinct colors $\sigma_1, \sigma_2, \ldots, \sigma_K$, with associated values $\mathsf{FREQ}(\alpha, \beta, \sigma_1)$, $\mathsf{FREQ}(\alpha, \beta, \sigma_2)$, …, $\mathsf{FREQ}(\alpha, \beta, \sigma_K)$, such that for each σ_i, there exists a σ_i-point x_i satisfying $\alpha \leq x_i \leq \beta$. Here, $\mathsf{FREQ}(\alpha, \beta, \sigma_i)$ is the number of σ_i-points in $[\alpha, \beta]$.*

First, we describe a data structure for storing partially persistent lists, which will be heavily used for answering these queries. Then, we present a data structure that answers one-sided queries, i.e., either $\alpha = 1$ or $\beta = N$. This result is based on storing selected points and their colors in a persistent list; we achieve optimal space usage and query cost for this special case. Then, we show how the same data structure can be used to answer general queries; however the query cost is proportional to the total number of colors. Next, we show how the optimal query cost for two-sided queries can be achieved by a data structure that needs $O(N \log D)$ space. We then show how to reduce the space usage by presenting an $O(N\lceil \frac{\log^2 D}{\log N} \rceil)$-space data structure that answers an $\mathsf{EXACT1D}(\alpha, \beta)$ query in optimal $O(1 + K/B)$ I/Os; see Theorem 23.

[3] If $\log N > B$, then $\frac{\log N}{\log B} > \frac{\log N}{\log \log N} > \log^* N > \log^* B$. If $\log N < B$, then $\log^* N = O(\log^* B)$.

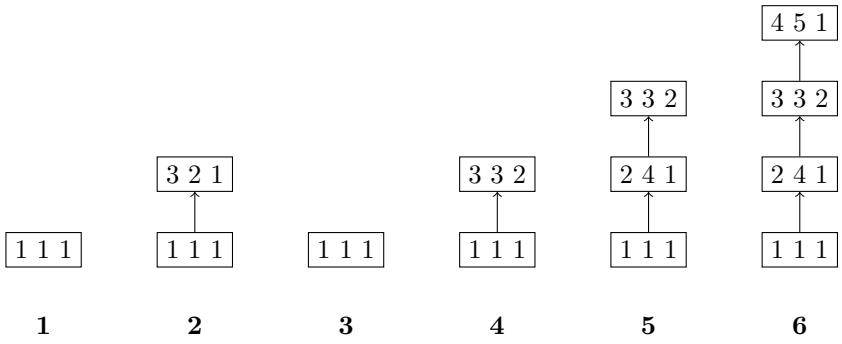


Figure 6 An example of a persistent list L for a set of five points such that $\mathsf{color}(1) = 1$, $\mathsf{color}(2) = 3$, $\mathsf{color}(3) = 3$, $\mathsf{color}(4) = 2$, and $\mathsf{color}(5) = 4$. Each version of the list is shown separately. Every entry of L contains a color σ, a point x, and its rank $\mathsf{rank}(x)$ (in this order). Versions $1, 2, 4, 5, 6$ correspond to points $1, 2, 3, 4, 5$. We remark that the total space used by L is $6 \cdot \text{const}$.

5.1 Partially Persistent Lists

Our solution makes use of the data structure for *offline partially persistent linked list* problem. In this problem, we maintain a linked list L under insertions and deletions: an arbitrary list entry can be deleted and a new entry can be inserted at any position of the list. We assume that L is originally empty and the sequence of updates is known in advance. The t-th update creates a new version of the list with timestamp t.

We can traverse the list (or a prefix of the list) for any timestamp t: given t, we can report all list entries that were in the list at time t. Also, given t and a value a, we can report all list entries that are smaller than a and that were in the list at time t. Such queries can be supported in $O(1 + K/B)$ I/Os, where K is the number of reported entries. The space needed to store a persistent list is $O(r)$ words or $O(r \log r)$ bits, where r is the total number of updates. The persistent list can be implemented using e.g., the partially persistent B-tree [6]; we refer to [16], Section 4, for a more detailed description.

5.2 1-sided Queries

A *one-sided* query is of the form $[1, \beta]$ or $[\alpha, N]$. Define the rank of a point p as the number of points that are smaller than or equal to p and have the same color:

$$\mathsf{rank}(p) = |\{p' \mid p' \leq p \text{ and } \mathsf{color}(p') = \mathsf{color}(p)\}|$$

We use a persistent list L in order to answer one-sided queries. (See Figure 6.) Every entry of L stores a color and entries are sorted by their colors in increasing order. An entry of L also contains some point p and its rank, $\mathsf{rank}(p)$. The list L is organized as follows. Originally L is empty. All points from a set X are traversed in the left-to-right order. When a point x is reached, we update the list L as follows. If L already contains an entry e for $\mathsf{color}(x)$, we remove e from L. Then we insert a new entry e' of color $\mathsf{color}(x)$ into an appropriate position (i.e., in sorted order) in L. The entry e' contains the point x and $\mathsf{rank}(x)$. Thus, the list L is updated one or two times for every point in X and the total number of entries in L is $O(N)$. In addition to L, we keep a table $V[1 \ldots N]$; its i-th entry $V[i]$ contains the index of the version of L after the point i was inserted.

Suppose that we want to report all colors that occur in a range $[1, \alpha]$ and output the number of occurrences for each color. We traverse the j-th version L_j of L where $j = V[\alpha]$. For every entry e of L_j, we report its color and the rank of the point p stored in e.

This brings us to the following result.

▶ **Theorem 19.** *There exists an $O(N)$ space data structure, using which we can answer* EXACT1D$(1, \beta)$ *in $O(1 + \frac{K}{B})$ I/Os.*

We can use the same persistent list L in order to answer two-sided queries. Unfortunately the query cost is significantly higher.

▶ **Lemma 20.** *There exists an $O(N)$-space data structure, using which we can answer* EXACT1D(α, β) *in $O(1 + D/B)$ I/Os.*

Proof. We use the persistent list L and the array V defined earlier. Suppose we want to answer a query on a range $[\alpha, \beta]$. We identify the versions f and l of L that correspond to $\alpha - 1$ and β, i.e., $f = V[\alpha - 1]$ and $l = V[\beta]$. By definition, both L_f and L_l contain exactly one entry for every color σ that occurs in $[1, \alpha - 1]$ and $[1, \beta]$ respectively. Let e_f and e_l denote entries of color σ in L_f and L_l respectively. If e_f and e_l contain the same point x, then $x < \alpha$ and σ does not occur in $[\alpha, \beta]$. If e_f and e_l contain two different points, p_f and p_l respectively, then σ occurs $\mathsf{rank}(p_l) - \mathsf{rank}(p_f)$ times in $[\alpha, \beta]$. Finally if there is no entry e_f in L_f, then the leftmost occurrence of σ lies in $[\alpha, \beta]$. In this case, σ occurs $\mathsf{rank}(p_l)$ times.

The values of $\mathsf{rank}(p_l)$ and $\mathsf{rank}(p_f)$ for all relevant colors can be computed as follows. We simultaneously generate versions L_f and L_l of L and merge them into a new (non-persistent) list T. Entries of T are sorted by color and T contains at most two entries of the same color. When T is available, we can retrieve values of p_l and p_f (resp. the value of p_l is e_f does not exist) for at least $B/2$ colors σ with one I/O operation. Hence, we can compute the number of occurrences for at least $B/2$ colors with $O(1)$ I/Os. The total cost of our procedure is $O(q/B)$, where q denotes the number of entries in T. Since $q \leq D$, our algorithm answers an EXACT1D(α, β) query in $O(1 + D/B)$ I/Os. ◀

5.3 Answering in Optimal I/Os

We use the data structure of Lemma 20 as a building block. For $i = 1, 2, \ldots, \log(D/B)$, we divide the set of points into i-chunks so that each chunk, except the last one, contains $B \cdot 2^i$ distinct colors. The last chunk contains at least $B \cdot 2^i$ and at most $B \cdot 2^{i+1}$ distinct colors. For any particular i, we use $C_{i,j}$ to denote the j^{th} i-chunk. For every i-chunk, we keep the data structure of Lemma 20 that answers queries in $O(2^i)$ I/Os. We also store the data structure of Lemma 20 for every two consecutive i-chunks $C_{i,j} \cup C_{i,j+1}$.

Consider a query $[\alpha, \beta]$. We find the smallest i such that there is no i-chunk $C_{i,j}$ that is entirely contained in $[\alpha, \beta]$. Then, the interval $[\alpha, \beta]$ intersects at most two i-chunks. Suppose that $[\alpha, \beta]$ is contained in one chunk $C_{i,j}$. Then we report all colors in $[\alpha, \beta]$ and the number of their occurrences in $O((B \cdot 2^i)/B)$ I/Os using the result of Lemma 20. If $i = 1$, the query cost is $O((B \cdot 2)/B) = O(1)$ I/Os. If $i > 1$, the interval $[\alpha, \beta]$ contains at least one $(i - 1)$-chunk and the number of distinct colors in $[\alpha, \beta]$ is $K \geq B \cdot 2^{i-1}$. Hence the query cost is $O(2^i) = O(K/B)$ I/Os. Now suppose that $[\alpha, \beta]$ intersects two chunks, $C_{i,j}$ and $C_{i,j+1}$. In this case, we answer a query using the data structure for the chunk pair $C_{i,j} \cup C_{i,j+1}$. By the same argument as above, the total query cost is $O(K/B)$ I/Os.

It remains to show how we can find the desired minimum i and the chunks $C_{i,j}$ and $C_{i,j+1}$. For $i = 1, 2, \ldots, \log^* n$, we keep boundaries of i-chunks in a bit vector of size $O(n)$. Using B_i and the data structure from [26], we can find the i-chunks that contain α and β in $O(1)$ time. In order to find, $C_{i,j}$ and $C_{i,j+1}$ we proceed as follows. Using B_r we find r-chunks C_{r,f_r} and C_{r,l_r} that contain α and β respectively for $r = 1, 2, \ldots, i$; we stop when $f_i = l_i$ or $f_i + 1 = l_i$. The total cost of finding the chunks is $O(i) = O(K/B + 1)$.

We get the following result.

▶ **Theorem 21.** *There exists an $O(N \log D)$-space data structure that answers a query* EXACT1D(α, β) *in optimal $O(1 + K/B)$ I/Os.*

5.4 Reducing Space Usage

▶ **Lemma 22.** *Consider a subset of the input points containing d distinct colors. If α and β lie within the boundaries of this subset, by using an $O\big(N(\log D + \frac{\log N}{D})\big)$-bit data structure, we can answer an* EXACT1D(α, β) *query in $O(1 + d/B)$ I/Os.*

Proof. Let P' be the subset of the set of points. We divide the set of points into sub-chunks C_i of size D^2 each (except for the last sub-chunk that can contain up to $2D^2$ points). If P' contain less than $2D^2$ points, all points are in the same sub-chunk. For every sub-chunk we keep the data structure of Lemma 20. Since each sub-chunk contains D^2 points and the color of every point can be specified with $\log D$ bits, the space usage of a sub-chunk data structure is $O(D^2 \log D)$ bits. If there is more than one sub-chunk, we keep a global data structure G that contains $O(N/D)$ elements. We implement G as a persistent list that is similar to the structure of Lemma 20. However our modified list contains at most D elements for each sub-chunk. We traverse sub-chunks of P' in the left-to-right order. For each sub-chunk C_i, we perform $O(D)$ updates on G. For every color σ that occurred in C_i, the following updates are performed. If G already contains an entry e_o of color σ we remove e_o from G. Then, we insert a new entry e of color σ into G; e contains the rightmost point x_σ of color σ in C_i and the rank of x_σ, $\mathsf{rank}(x_\sigma)$.

Consider a query $[\alpha', \beta']$ such that α' and β' are boundaries of sub-chunks C_f and C_l. Let $i_\alpha = V_g[f-1]$ and $i_\beta = V_g[l]$ denote the versions of G that correspond to sub-chunks C_{f-1} and C_l respectively. We can answer a query on $[\alpha', \beta']$ by essentially the same method that is used in Lemma 20. We generate versions G_{i_α} and G_{i_β} and merge them into a list T_G. Since G_{i_α} contain at most one entry of every color, T_G contains at most two entries of the same color. Let $p_f(\sigma)$ denote the rightmost occurrence of σ in $[1, \alpha' - 1]$ (i.e., $p_f(\sigma)$ is the rightmost occurrence of σ in the sub-chunk C_{f-1} or in some sub-chunk to the left of C_{f-1}) and $p_l(\sigma)$ denotes the rightmost occurrence of σ in $[1, \beta']$ (i.e., $p_l(\sigma)$ is the rightmost occurrence of σ in sub-chunks $C_1, C_2, \ldots, C_l$). The color σ occurs $\mathsf{rank}(p_l(\sigma)) - \mathsf{rank}(p_f(\sigma))$ times in $[\alpha', \beta']$ (or $\mathsf{rank}(p_l(\sigma))$ times if $p_f(\sigma)$ does not exist). We can retrieve at least $B/2$ values of $\mathsf{rank}(p_f(\sigma))$ and $\mathsf{rank}(p_l(\sigma))$ with one I/O operation. The total cost of a query is $O(1 + d/B)$ because the list T_G contains at most $2d$ entries.

Now we consider an EXACT1D(α, β) query restricted to the boundaries of P'. Suppose that $\alpha \in C_f$ and $\beta \in C_l$. If α and β are in the same sub-chunk, we use the data structure for that sub-chunk to answer the query. Otherwise we decompose $[\alpha, \beta]$ into at most three parts, $[\alpha, \beta] = [\alpha, \alpha' - 1] \cup [\alpha', \beta'] \cup [\beta' + 1, \beta]$ so that $[\alpha', \beta'] = \cup_{i=f+1}^{l-1} C_i$. The intervals $[\alpha, \alpha' - 1]$ and $[\beta' + 1, \beta]$ are in sub-chunks C_f and C_l respectively. We answer extended color reporting queries on $[\alpha, \alpha' - 1]$, $[\alpha', \beta']$, and $[\beta' + 1, \beta]$. An answer to each query is stored in a list of colors T_i, $1 \le i \le 3$. The entries of T_i are sorted by color in increasing order. For an entry of color σ in a list T_i, we store the frequency of σ in the corresponding interval. We merge lists T_i into a global list T in $O(1 + d/B)$ I/Os. Then we traverse the resulting list and obtain the frequencies of all colors in $[\alpha, \beta]$ in $O(1 + d/B)$ I/Os.

The persistent list G uses $O(N\lceil \frac{\log N}{D} \rceil)$ bits of space. All chunk data structures require $O(N \log D)$ bits. Hence the total space is $O\big(N(\log D + \frac{\log N}{D})\big)$ bits. ◀

▶ **Theorem 23.** *There exists an* $O(N\lceil\frac{\log^2 D}{\log N}\rceil)$*-space data structure that answers an* EXACT1D(α, β) *query in optimal* $O(1 + K/B)$ *I/Os.*

Proof. We divide the set of points into chunks as described in Theorem 21, and then use Lemma 22 for each chunk. Since each point is contained in $\log D$ chunks, the total space occupied is $O(N(\log D + \lceil\frac{\log N}{D}\rceil)\log D) = O(N(\log^2 D + \log N))$ bits or $O(N\lceil\frac{\log^2 D}{\log N}\rceil)$ words. A query is answered in $O(1 + K/B)$ I/Os as explained in the proof of Theorem 21. ◀

6 Conclusion

We leave the following questions unanswered. First, *can we obtain a data structure that answers one-dimensional color reporting queries with approximate frequencies in optimal* $O(1 + K/B)$ *I/Os?* Although such a data structure with $O(N \log^* N)$ space seems feasible, techniques do not seem to generalize to the more general case of two-dimensional dominance queries. Second, an interesting question is whether the approximation factor δ can be provided during query time, as opposed to the current approach of having to provide it during construction time. Additionally, we leave the question of reporting colors with exact frequency in case of 2D dominance unanswered. Finally, it would be interesting if one could generalize 2D dominance to the more general 3-sided/4-sided queries.

References

1. Peyman Afshani. On Dominance Reporting in 3D. In *ESA*, pages 41–51, 2008.
2. Pankaj K. Agarwal, Alon Efrat, and Micha Sharir. Vertical Decomposition of Shallow Levels in 3-Dimensional Arrangements and Its Applications. *SIAM J. Comput.*, 29(3):912–953, 1999. doi:10.1137/S0097539795295936.
3. Lars Arge, Vasilis Samoladas, and Jeffrey Scott Vitter. On Two-Dimensional Indexability and Optimal Range Search Indexing. In *Proc. 18th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 346–357, 1999. doi:10.1145/303976.304010.
4. Lars Arge, Vasilis Samoladas, and Ke Yi. Optimal External Memory Planar Point Enclosure. *Algorithmica*, 54(3):337–352, 2009. doi:10.1007/s00453-007-9126-2.
5. Lars Arge and Jeffrey Scott Vitter. Optimal External Memory Interval Management. *SIAM J. Comput.*, 32(6):1488–1508, 2003. doi:10.1137/S009753970240481X.
6. Bruno Becker, Stephan Gschwind, Thomas Ohler, Bernhard Seeger, and Peter Widmayer. An Asymptotically Optimal Multiversion B-Tree. *VLDB J.*, 5(4):264–275, 1996. doi:10.1007/s007780050028.
7. Panayiotis Bozanis, Nectarios Kitsios, Christos Makris, and Athanasios K. Tsakalidis. New Upper Bounds for Generalized Intersection Searching Problems. In *Proc. 22nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 464–474, 1995. doi:10.1007/3-540-60084-1_97.
8. Panayiotis Bozanis, Nectarios Kitsios, Christos Makris, and Athanasios K. Tsakalidis. New Results on Intersection Query Problems. *Comput. J.*, 40(1):22–29, 1997. doi:10.1093/comjnl/40.1.22.
9. Timothy M. Chan and Bryan T. Wilkinson. Adaptive and Approximate Orthogonal Range Counting. In *Proc. 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 241–251, 2013. doi:10.1137/1.9781611973105.18.
10. Arnab Ganguly, Wing-Kai Hon, and Rahul Shah. Stabbing Colors in One Dimension. In *2017 Data Compression Conference, DCC 2017, Snowbird, UT, USA, April 4-7, 2017*, pages 280–289, 2017. doi:10.1109/DCC.2017.44.

11 Prosenjit Gupta, Ravi Janardan, and Michiel H. M. Smid. Further Results on Generalized Intersection Searching Problems: counting, Reporting, and Dynamization. *Journal of Algorithms*, 19(2):282–317, 1995. doi:10.1006/jagm.1995.1038.

12 Prosenjit Gupta, Ravi Janardan, and Michiel H. M. Smid. Algorithms for Generalized Halfspace Range Searching and Other Intersection Searching Problems. *Comput. Geom.*, 6:1–19, 1996. doi:10.1016/0925-7721(95)00012-7.

13 Ravi Janardan and Mario A. Lopez. Generalized intersection searching problems. *International Journal of Computational Geometry and Applications*, 3(1):39–69, 1993.

14 Marek Karpinski and Yakov Nekrich. Top-K Color Queries for Document Retrieval. In *Proc. 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 401–411, 2011. URL: http://www.siam.org/proceedings/soda/2011/SODA11_032_karpinskim.pdf.

15 Kasper Green Larsen and Rasmus Pagh. I/O-efficient data structures for colored range and prefix reporting. In *Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 583–592, 2012. URL: http://portal.acm.org/citation.cfm?id=2095165{&}CFID=63838676{&}CFTOKEN=79617016.

16 Kasper Green Larsen and Freek van Walderveen. Near-Optimal Range Reporting Structures for Categorical Data. In *Proc. 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 256–276, 2013.

17 Jiří Matoušek. Reporting Points in Halfspaces. In *Proc. 32nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 207–215, 1991. doi:10.1109/SFCS.1991.185370.

18 S. Muthukrishnan. Efficient algorithms for document retrieval problems. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 657–666, 2002. doi:10.1145/545381.545469.

19 Alexandros Nanopoulos and Panayiotis Bozanis. Categorical Range Queires in Large Databases. In *Proc. 8th International Symposium on Advances in Spatial and Temporal Databases, (SSTD)*, pages 122–139, 2003. doi:10.1007/978-3-540-45072-6_8.

20 Gonzalo Navarro and Yakov Nekrich. Top-k document retrieval in optimal time and linear space. In *Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1066–1077, 2012. URL: http://portal.acm.org/citation.cfm?id=2095200{&}CFID=63838676{&}CFTOKEN=79617016.

21 Yakov Nekrich. External Memory Range Reporting on a Grid. In *Proc. 18th International Symposium on Algorithms and Computation (ISAAC)*, pages 525–535, 2007. doi:10.1007/978-3-540-77120-3_46.

22 Yakov Nekrich. Data Structures for Approximate Orthogonal Range Counting. In *Proc. 20th International Symposium on Algorithms and Computation (ISAAC)*, pages 183–192, 2009. doi:10.1007/978-3-642-10631-6_20.

23 Yakov Nekrich. Efficient range searching for categorical and plain data. *ACM Trans. Database Syst.*, 39(1):9, 2014. doi:10.1145/2543924.

24 Yakov Nekrich and Jeffrey Scott Vitter. Optimal Color Range Reporting in One Dimension. In *Proc. 21st Annual European Symposium Algorithms (ESA)*, pages 743–754, 2013. doi:10.1007/978-3-642-40450-4_63.

25 Manish Patil, Sharma V. Thankachan, Rahul Shah, Yakov Nekrich, and Jeffrey Scott Vitter. Categorical range maxima queries. In *Proc. 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 266–277, 2014. doi:10.1145/2594538.2594557.

26 Mihai Patrascu. Succincter. In *Proc. 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 305–313, 2008. doi:10.1109/FOCS.2008.83.

27 Saladi Rahul. Improved Bounds for Orthogonal Point Enclosure Query and Point Location in Orthogonal Subdivisions in $\mathbb{R}^3$. In *Proc.26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 200–211, 2015. doi:10.1137/1.9781611973730.15.

28 Saladi Rahul. Approximate Range Counting Revisited. In *Proc. 33rd International Symposium on Computational Geometry (SoCG)*, pages 55:1–55:15, 2017. doi:10.4230/LIPIcs.SoCG.2017.55.

29 Kunihiko Sadakane. Succinct data structures for flexible text retrieval systems. *J. Discrete Algorithms*, 5(1):12–22, 2007. doi:10.1016/j.jda.2006.03.011.

Approximating Distance Measures for the Skyline

Nirman Kumar
Department of Computer Science, University of Memphis, TN, USA
nkumar8@memphis.edu

Benjamin Raichel
Department of Computer Science, University of Texas at Dallas, TX, USA
benjamin.raichel@utdallas.edu

Stavros Sintos
Department of Computer Science, Duke University, Durham, NC, USA
ssintos@cs.duke.edu

Gregory Van Buskirk
Department of Computer Science, University of Texas at Dallas, TX, USA
greg.vanbuskirk@utdallas.edu

Abstract

In multi-parameter decision making, data is usually modeled as a set of points whose dimension is the number of parameters, and the *skyline* or *Pareto* points represent the possible optimal solutions for various optimization problems. The structure and computation of such points have been well studied, particularly in the database community. As the skyline can be quite large in high dimensions, one often seeks a compact summary. In particular, for a given integer parameter k, a subset of k points is desired which best approximates the skyline under some measure. Various measures have been proposed, but they mostly treat the skyline as a discrete object. By viewing the skyline as a continuous geometric hull, we propose a new measure that evaluates the quality of a subset by the Hausdorff distance of its hull to the full hull. We argue that in many ways our measure more naturally captures what it means to approximate the skyline.

For our new geometric skyline approximation measure, we provide a plethora of results. Specifically, we provide (1) a near linear time exact algorithm in two dimensions, (2) APX-hardness results for dimensions three and higher, (3) approximation algorithms for related variants of our problem, and (4) a practical and efficient heuristic which uses our geometric insights into the problem, as well as various experimental results to show the efficacy of our approach.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Skyline, Pareto optimal, Approximation, Hardness, Multi-criteria decision making

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.10

Related Version The full version of the paper is available online at [22], `http://utdallas.edu/~benjamin.raichel/stair.pdf`.

Funding B. Raichel and G. Van Burskirk are partially supported by NSF CRII Award 1566137 and CAREER Award 1750780. S. Sintos is supported by NSF under grants CCF-15-13816, CCF-15-46392, and IIS-14-08846, by an ARO grant W911NF-15-1-0408, and by BSF Grant 2012/229 from the U.S.-Israel Binational Science Foundation.

1 Introduction

When deciding which entry to return from a database where entries have multiple parameters, one must consider various criteria all at once. For example, given a collection of possible hotels, one seeks a hotel which costs less, is nearest to the desired location, and has a high rating. Typically one cannot optimize all these criteria simultaneously, though one would

never make a selection for which there is an alternative that is better in every way (i.e., cheaper, closer, and higher rated). This naturally leads to the definition of so called staircase or skyline points,[1] for which there are no strictly better alternatives, and each of which is a possible optimal solution depending on how the criteria are weighted.

Formally, we model our input as a set of n points P in $\mathbb{R}^d$. Given two points $p, q \in \mathbb{R}^d$, we say p *dominates* q, written $p \preceq q$, if $p_i \leq q_i$ for all i, where p_i (resp. q_i) denotes the ith coordinate of p (resp. q). Then the *staircase points* of P, denoted $\mathsf{S}(P)$, is the subset of points $p \in P$ such that p is *not* dominated by any point in $P \setminus \{p\}$.

While the staircase points model the set of possible solutions for various multi-criteria objectives, in many situations one desires a small subset of these points which well represent the tradeoffs in the solution space. For example, one may wish to present the user with a short list of search results to choose from. Moreover, the staircase might be large, particularly for high dimensional data sets, thus making a compact description highly desirable from a computational perspective. Here we propose a new measure to evaluate how well a subset of points approximates the staircase. Our work differs from previous attempts to define such a measure since rather than viewing $\mathsf{S}(P)$ as a discrete point set, instead we consider it as defining a continuous hull. Namely, we define the *staircase hull* of P, denoted $\mathsf{SH}(P)$, as the set of all points in $\mathbb{R}^d$ that are dominated by some point in P. Then for some integer parameter k, we seek the subset of at most k points $Q \subseteq P$ such that the distance between $\mathsf{SH}(Q)$ and $\mathsf{SH}(P)$ is minimized. To measure the distance between the two hulls we use the standard geometric notion of Hausdorff distance, which roughly speaking for two sets is defined as the maximum of all the distances from a point in one set to the closest point in the other set. For our problem we prove that this is equivalent to minimizing the maximum distance from any point in P to $\mathsf{SH}(Q)$. We argue that our geometric perspective in many ways better captures what it means to approximate the staircase. Moreover, our measure has additional advantages such as being translation invariant. In addition to defining this new measure, we provide a number of algorithmic, hardness, approximation, and experimental results as detailed below.

Previous work. Skyline/staircase points have been extensively used and studied, particularly in the database community (see for example [8]). In particular, it is known that for a set P of n points in $\mathbb{R}^d$, $\mathsf{S}(P)$ can be computed in $O(n(\log n + \log^{d-2} n))$ worst case time [24] for any d or in $O(n \log^{d-3} n \log\log n)$ for $d \geq 4$ [17]. In [12] they show a randomized algorithm for computing the $\mathsf{S}(P)$ in $O(n \log^{d-3})$ time for $d \geq 4$. There have also been a number of previous works on approximating the best subset of k input points to represent the skyline under various measures. Lin *et al.* [25] considered selecting k points so as to dominate the maximum possible number of points from P. This is an instance of the standard maximum coverage problem, for which the greedy algorithm achieves a $1 - 1/e$ approximation (to the number of dominated points). Later, Tao *et al.* [34] considered representing the skyline by a k-center clustering of the points in $\mathsf{S}(P)$, that is, minimize the maximum distance of a point in $\mathsf{S}(P)$ to the chosen subset, for which there are several standard 2-approximation algorithms to the optimal clustering radius. Koltun and Papadimitriou [21] consider what they call approximately dominating representatives (ADR), where a subset $Q \subseteq P$ is an ε-ADR if for every $p \in P$ there is some $q \in Q$ such that $(1 - \varepsilon)q$ dominates p (their actual definition

[1] In the title/abstract we use the term "skyline", more common in database papers, however in the rest of the paper we favor the term "staircase", common in computational geometry. In economics the term "Pareto" is often used.

differs slightly as they define dominating with the reverse inequality). Rather than focusing on minimizing the error radius as in our case and in [34], instead [21] keeps ε fixed and attempts to minimize k. As this variant is an instance of the well known Set-Cover problem, the greedy algorithm achieves an $O(\log n)$ approximation. In all three papers [25, 34, 21], among other things the authors give an exact polynomial time solution in 2 dimensions, and argue the problem is hard for dimensions 3 and higher. Several other measures have also been considered. Søholm *et al.* [33] select points so as to cover the maximum total area possible, giving an algorithm for the 2D case and some experimental results. Note they define dominance with the reverse inequality and so area is with respect to the coordinate axes, while in our case it would require defining some bounding box for the data. Magnani *et al.* [26] define a measure which combines two quantities which they call significance and diversity (where the user must specify the relative importance of each), and provide a hardness proof and experimental results.

More generally, there have been many other previously defined notions of compact approximate representations of the data, perhaps most notably the notion of clustering [36]. The so called k-regret minimizing set is a recent method of finding representative data points, which is used to approximate top-k queries [28, 13, 3, 11, 6, 23]. In the geometry community the notion of coresets, which are small subsets of the input approximately preserving some specified geometric property, has been widely studied [2]. Related to our notion of approximating the staircase hull, the case of approximating the convex hull [7] and the conic hull [9] was previously considered. These hull approximations more generally relate to a number of different types of matrix factorizations used in areas such as machine learning, including non-negative matrix factorization, CUR decomposition, and finding the top-k singular vectors. Note that points on the convex and conic hull boundaries are in general determined by a combination of multiple hull vertices. For the staircase hull we argue that each boundary point in some sense is determined by a single hull vertex, leading to stronger results as the problem can be modeled with asymmetric k-center.

Our results. As discussed above, here we introduce a new measure for approximating the staircase. Specifically, we seek the subset of k points $Q \subseteq P$ such that the Hausdorff distance between $\mathsf{SH}(Q)$ and $\mathsf{SH}(P)$ is minimized, which we refer to as the ***k-staircase problem***. In addition to introducing this new measure we have the following results:

1. In Section 3 we show that the k-staircase problem is APX-hard for any dimension $d \geq 3$. In particular, we give a simple argument to show it is hard to approximate within $\sqrt{2+\sqrt{3}} \approx 1.932$ for $d \geq 4$, and for $d = 3$ a more involved proof shows it is hard to approximate within a factor of $\sqrt{\frac{2+\sqrt{3}}{2+\sqrt{2}}} \approx 1.0455$.
2. Despite being hard to approximate for $d \geq 3$, in Section 4 we show that in two dimensions it can be solved optimally in $O(n \log^3 n)$ time.
3. In Section 2 we argue the k-staircase problem can actually be seen as an instance of the well known asymmetric k-center problem. In Section 5 this is used to give approximation algorithms for the k-staircase problem. In particular, in Section 5.1 we propose an $O(\log^*(n))$ approximation running in $O(n^2(d+\log^2 n))$ time, and an $O(\log^*(k))$ approximation running in polynomial time.
4. In Section 5.2, we show that if one is allowed to pick $2k$ rather than k points, then by making use of Well Separated Pair Decompositions (WSPD), for any fixed dimension d we can get a faster $O(kn \operatorname{polylog} n)$ time algorithm which still achieves an $O(\log^*(n))$ approximation for the k-staircase problem. This is the only result where we assume d is fixed. (The algorithm works for any d, though the approximation and time degrade.)

5. In Section 6 we propose a new heuristic for the k-staircase problem which is based on finding what we call relaxed CCV[2] points. Such points have nice symmetric geometric properties, and we argue that any point set always contains at least one such point. While certain contrived worst case examples make it difficult to provide worst case guarantees, we show experimentally in Section 6.1 that approximating the staircase hull using relaxed CCV points consistently gives a smaller amount of error than the guaranteed $O(\log^*(n))$-approximation algorithm of [30]. Moreover, our heuristic is simple, easy to implement, and fast, thus making it quite practical. Finally, in Section 6.2, we compare our measure to several others by using NBA player statistics to predict NBA All Star teams. Not only does our measure always select the most all star players, but also it is inversely proportional to the number of selected players, while for other measures the correspondence is less consistent.

Due to space limitations, some of the proofs can be found in the full version of the paper [22].

The Hausdorff distance uses the standard L_2 norm for distances between points, however, it can be generalized to any L_p norm.[3] Intuitively, the L_1 norm focuses on the average value over the dimensions, while L_∞ focuses on the dimension which is largest. We argue our algorithmic results hold for any L_p norm, thus allowing the user to tune the dial between these two extremes in order to best suit a given application.

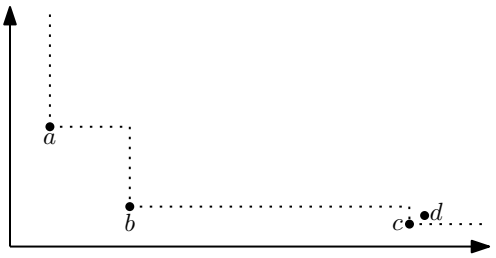


Figure 1.1 For $k = 2$, our solution is a and b, k-center a or b and c or d, max cover a or b and c.

Understanding and comparing measures. As mentioned above, one of the main ways in which our measure differs from [25] and [34] is that it considers the staircase as defining a continuous hull rather than a discrete point set. In particular, consider the simple four point example shown in Figure 1.1. For $k = 2$, in the maximum coverage measure of [25] the optimal solution contains either a or b along with the point c. For the k-center clustering measure of [34] the optimal solution contains exactly one of a or b and exactly one of c or d. On the other hand, for our measure the optimal solution contains points a and b. Arguably a and b represent real tradeoffs between the horizontal and vertical coordinate values, whereas c and d have a negligible advantage in the vertical coordinate over b but are far worse in the horizontal coordinate. In other words, when considering the discrete points, c and d are far from a and b. However, when considering the continuous hulls, c and d are very close to the hull $\mathsf{SH}(\{a, b\})$, yet b has a larger distance to the hull $\mathsf{SH}(\{a, c\})$.

There are several additional nice properties of our measure. First, it is natural from a geometric perspective as it is just the standard geometric notion of Hausdorff distance between two objects. Second, it is translation invariant, i.e., the optimal solution does not

[2] The notion of CCV points, which stands for "center capturing vertex", was introduced in [30] to solve the asymmetric k-center problem approximately.

[3] Here we use the standard L_p notation, though later we use L_α as p is often used to denote a point.

change if all points are translated by the same amount (where the amount in each coordinate can differ). This is also true of the measures in [25] and [34], however, it is not the case for [33] and [21], making it difficult to compare their measure to ours as theirs strongly depends on the choice of origin. A number of other properties, such as scale invariance or stability, have been used to characterize previous approaches for approximating the staircase. In the full version of the paper [22] we provide a table and description comparing our measure with others, showing the tradeoffs of these various properties. Rather than focusing on any one of these properties, here we advocate the benefit of taking into account the continuous hull, and give some partial justification of this approach with our experimental results in Section 6.

In order to maximize the benefit of our measure, there are several preprocessing steps that one may wish to take. First observe that depending on what the coordinate values represent, in some coordinates one might desire a larger value and in others a smaller one (e.g. hotel rating vs. price). To handle this one can negate all values in coordinates for which large values are better, and then translation invariance implies we can shift the points to remove negative values (if so desired). Translation invariance also implies one can translate the point set such that in each coordinate the minimum value of any data point is zero. Also, note that if one scales the data by the same factor in every coordinate then the optimal solution does not change, hence one can scale so the data points lie in the unit hypercube if so desired. On the other hand our measure is sensitive to scaling by different values in each coordinate, and this fact can be used to encode our preferences over the coordinates. Specifically, if one scales such that the maximum value is exactly 1 in each coordinate then this attempts to put each coordinate on equal footing. Alternatively, if one knows certain attributes are more important than others, one can weight the scaling differently in each coordinate to take this into account. Note that all this can easily be done in linear time.

2 Problem Statement and Connection to Asymmetric K-center

2.1 Definitions and problem statement

Given points $p, q \in \mathbb{R}^d$, we say p *dominates* q, written $p \preceq q$, if $p_i \leq q_i$ for all i, where p_i (resp. q_i) denotes the ith coordinate of p (resp. q). For a point set $P \subset \mathbb{R}^d$, let $\mathsf{S}(P)$ denote the set of *staircase points* of P, which is the set of points $p \in P$ such that p is *not* dominated by any point in $P \setminus \{p\}$. Finally, let $\mathsf{SH}(P)$ denote the *staircase hull* of P, which is the subset of points in $\mathbb{R}^d$ dominated by some point in P. Notice that $\mathsf{S}(P) \subset P \subset \mathsf{SH}(P)$ and $\mathsf{SH}(P) = \mathsf{SH}(\mathsf{S}(P))$ as dominance is transitive.

Let $\mathsf{d}_\alpha(x, y) = ||x - y||_\alpha$ denote the distance between $x, y \in \mathbb{R}^d$ with respect to the L_α norm[4], where $\alpha \geq 1$. For (closed) sets $X, Y \subset \mathbb{R}^d$, let $\mathsf{d}_\alpha(X, Y) = \min_{x \in X, y \in Y} ||x - y||_\alpha$ be the distance between point sets X and Y with respect to the L_α norm. If X is a singleton $\{x\}$, let $\mathsf{d}_\alpha(x, Y)$ denote $\mathsf{d}_\alpha(X, Y)$. For point sets X and Y, let $\mathsf{d}_{\mathsf{H}_\alpha}(X, Y) = \max\{\max_{x \in X} \mathsf{d}_\alpha(x, Y), \max_{y \in Y} \mathsf{d}_\alpha(y, X)\}$ denote the Hausdorff distance with respect to the L_α norm, between X and Y. Typically, we consider the case where $Y \subseteq X$; here $\mathsf{d}_{\mathsf{H}_\alpha}(X, Y) = \max_{x \in X} \mathsf{d}_\alpha(x, Y)$.

▶ **Problem 1** (k-staircase problem). *Given a set of n points $P \subset \mathbb{R}^d$ and a parameter k, find a subset $Q \subseteq P$ with $|Q| \leq k$ which minimizes $\mathsf{d}_{\mathsf{H}_\alpha}(\mathsf{SH}(P), \mathsf{SH}(Q))$.*

[4] The L_α norm is defined for $\alpha \geq 1$, as $||u||_\alpha = \left(\sum_{j=1}^{d} |u_j|^\alpha\right)^{1/\alpha}$.

For $Q \subseteq P$, the value $r = \mathsf{d}_{\mathsf{H}_\alpha}(\mathsf{SH}(P), \mathsf{SH}(Q))$ is *the radius of* Q. Throughout the paper, we say that Q *r-covers* P when $\mathsf{d}_{\mathsf{H}_\alpha}(\mathsf{SH}(P), \mathsf{SH}(Q)) \leq r$. For an optimal solution Q^* to Problem 1, we use r^* to denote its corresponding optimal radius. A subset $Q \subseteq P$ of size k is called a c-approximation for Problem 1 if the radius of Q is at most cr^*.

2.2 Properties and asymmetric k-center

Above we defined the notions of dominance, staircase points, and the staircase hull. In this section we provide further definitions and properties, to ultimately reduce Problem 1 to the well known asymmetric k-center problem, which we now define. Then, using the $O(\log^* n)$-approximation algorithm for the asymmetric k-center problem [30], in Section 5 we can show how to approximate the k-staircase problem.

▶ **Problem 2** (Asymmetric k-center)**.** *Given a set of n points P, with a corresponding asymmetric distance function $f : P \times P \to \mathbb{R}^{\geq 0}$, and a parameter k, find a subset $Q \subseteq P$ of at most k points which minimizes* $\max_{p \in P} f(Q, p) = \max_{p \in P} \min_{q \in Q} f(q, p)$.

In the above definition, by an *asymmetric* distance function we mean a function $f : P \times P \to \mathbb{R}^{\geq 0}$ which satisfies the *directed triangle inequality*: for any three points $a, b, c \in P$ we have $f(a, c) \leq f(a, b) + f(b, c)$. However, f may not be symmetric, i.e., in general $f(a, b) \neq f(b, a)$, and thus it is not a metric.

To reduce Problem 1 to Problem 2, we need to provide a number of definitions and structural properties. First, observe that the staircase hull is the union of a set of regions determined by individual points of P (unlike the convex hull for example). Using $\mathsf{SH}(p)$ to denote the subset of points in $\mathbb{R}^d$ dominated by p, we observe that $\mathsf{SH}(P) = \bigcup_{p \in P} \mathsf{SH}(p)$.

Next, we introduce some useful notation that we use throughout the paper. For a vector $u = (u_1, \ldots, u_d) \in \mathbb{R}^d$ define $||u||_\alpha^- = \left(\sum_{u_j < 0} |u_j|^\alpha\right)^{1/\alpha}$, and $||u||_\alpha^+ = \left(\sum_{u_j \geq 0} u_j^\alpha\right)^{1/\alpha}$. Observe that, $||u||_\alpha^- = ||-u||_\alpha^+$ and $||u||_\alpha^+ = ||-u||_\alpha^-$.

We first provide a simple expression for the distance from a point $p = (p_1, \ldots, p_d) \in \mathbb{R}^d$ to the region $\mathsf{SH}(q)$ of another point $q = (q_1, \ldots, q_d)$.

▶ **Lemma 3.** *For any points $p, q \in \mathbb{R}^d$, we have that,* $\mathsf{d}_\alpha(p, \mathsf{SH}(q)) = ||q - p||_\alpha^+$.

Proof. The region $\mathsf{SH}(q)$ is the set of points $(x_1, \ldots, x_d) \in \mathbb{R}^d$ such that $x_j \geq q_j$ for $j = 1, \ldots, d$. The L_α distance from p to x to the power of α is $(||x - p||_\alpha)^\alpha = \sum_{j=1}^d |x_j - p_j|^\alpha$. It is easy to see that to minimize this expression over $\mathsf{SH}(q)$ one should take the point $(x_1, \ldots, x_d)$ which has $x_j = p_j$ for all j such that $p_j > q_j$ and one should let $x_j = q_j$ otherwise. The result follows. ◀

The above implies that for any two points p and q, one can define a natural notion of *directed distance* (or equivalently asymmetric distance) from p to q, namely $\vec{\mathsf{d}}_\alpha(p, q) = \mathsf{d}_\alpha(q, \mathsf{SH}(p))$. Note the change in the order of the arguments. Alternatively one could define the directed distance from p to q as $\mathsf{d}_\alpha(p, \mathsf{SH}(q))$, which is natural when viewing p as traveling to $\mathsf{SH}(q)$. However, our definition is more natural when considering how well $\mathsf{SH}(p)$ covers q, ultimately helping relate our problem to the asymmetric k-center problem.

By Lemma 3, we have that,

$$\vec{\mathsf{d}}_\alpha(p, q) = \mathsf{d}_\alpha(q, \mathsf{SH}(p)) = ||p - q||_\alpha^+ = ||q - p||_\alpha^- = \left(\sum_{j : q_j < p_j} |q_j - p_j|^\alpha \right)^{1/\alpha}.$$

We say that p r-covers q when $\vec{\mathsf{d}}_\alpha(p, q) \leq r$.

A fundamental inequality obeyed by our directed distances is the following:

▶ **Lemma 4.** *Let x be a point dominated by y, i.e., $x \in \mathsf{SH}(y)$. Then, for any point p we have that, $\vec{\mathsf{d}}_\alpha(p,x) \leq \vec{\mathsf{d}}_\alpha(p,y)$, and $\vec{\mathsf{d}}_\alpha(y,p) \leq \vec{\mathsf{d}}_\alpha(x,p)$.*

Proof. We only prove the first inequality, as the proof of the second is similar. We have that $(||x-p||_\alpha^-)^\alpha = \sum_{j:x_j<p_j} |x_j-p_j|^\alpha$. Since $x \in \mathsf{SH}(y)$ we have that for all $j=1,\ldots,d$, $y_j \leq x_j$. For any j where $x_j < p_j$ clearly $y_j < p_j$ as well, and moreover, $|y_j-p_j|^\alpha \geq |x_j-p_j|^\alpha$. Thus, $(||x-p||_\alpha^-)^\alpha \leq (||y-p||_\alpha^-)^\alpha$. It follows that, $(\vec{\mathsf{d}}_\alpha(p,x))^\alpha = (||x-p||_\alpha^-)^\alpha \leq (||y-p||_\alpha^-)^\alpha = (\vec{\mathsf{d}}_\alpha(p,y))^\alpha \Rightarrow \vec{\mathsf{d}}_\alpha(p,x) \leq \vec{\mathsf{d}}_\alpha(p,y)$. ◀

The above lemma can now be used to show that our directed distances satisfy the directed triangle inequality, a fact not immediately apparent from our distance formula (Lemma 3), and which we require later in order to apply previous results for asymmetric k-center.

▶ **Lemma 5.** *The directed distance function $\vec{\mathsf{d}}_\alpha(\cdot,\cdot)$ obeys the directed triangle inequality. That is, for any three points $a,b,c \in \mathbb{R}^d$ we have $\vec{\mathsf{d}}_\alpha(a,c) \leq \vec{\mathsf{d}}_\alpha(a,b) + \vec{\mathsf{d}}_\alpha(b,c)$.*

Proof. Let z be the closest point in $\mathsf{SH}(b)$ to c, i.e., $z = \operatorname{argmin}_{x\in\mathsf{SH}(b)} \mathsf{d}_\alpha(c,x)$, and let z' be the closest point in $\mathsf{SH}(a)$ to z, i.e., $z' = \operatorname{argmin}_{x\in\mathsf{SH}(a)} \mathsf{d}_\alpha(z,x)$. Then,

$$\vec{\mathsf{d}}_\alpha(a,c) \leq \mathsf{d}_\alpha(c,z') \leq \mathsf{d}_\alpha(c,z) + \mathsf{d}_\alpha(z,z') = \vec{\mathsf{d}}_\alpha(b,c) + \vec{\mathsf{d}}_\alpha(a,z) \leq \vec{\mathsf{d}}_\alpha(b,c) + \vec{\mathsf{d}}_\alpha(a,b).$$

The first inequality holds because $z' \in \mathsf{SH}(a)$ and $\vec{\mathsf{d}}_\alpha(a,c) = \mathsf{d}_\alpha(c,\mathsf{SH}(a))$. The second inequality holds because the L_α norm satisfies the triangle inequality. The third equality follows from the definition of z and z'. Since z is dominated by b, by Lemma 4 we have that $\vec{\mathsf{d}}_\alpha(a,z) \leq \vec{\mathsf{d}}_\alpha(a,b)$ so the result follows. ◀

We now argue that for any subset $Q \subseteq P$ in Problem 1, minimizing $\mathsf{d}_{\mathsf{H}_\alpha}(\mathsf{SH}(P),\mathsf{SH}(Q))$ can instead be thought of as minimizing the distance from the finite point set P to the hull $\mathsf{SH}(Q)$. Thus as $\mathsf{SH}(Q) = \bigcup_{q\in Q} \mathsf{SH}(q)$, this in turn allows us to think of the problem as selecting Q so as to minimize directed distances from P to Q, i.e., a clustering problem.

For a point set Q and a point x, in the following we use the notation $\vec{\mathsf{d}}_\alpha(Q,x) = \min_{q\in Q} \vec{\mathsf{d}}_\alpha(q,x) = \min_{q\in Q} \mathsf{d}_\alpha(x,\mathsf{SH}(q))$. Observe that $\min_{q\in Q} \mathsf{d}_\alpha(x,\mathsf{SH}(q)) = \mathsf{d}_\alpha(x,\mathsf{SH}(Q))$, and hence $\vec{\mathsf{d}}_\alpha(Q,x) = \mathsf{d}_\alpha(x,\mathsf{SH}(Q))$.

▶ **Lemma 6.** *For $Q \subseteq P$, $\mathsf{d}_{\mathsf{H}_\alpha}(\mathsf{SH}(P),\mathsf{SH}(Q)) = \max_{p\in P} \mathsf{d}_\alpha(p,\mathsf{SH}(Q)) = \max_{p\in\mathsf{S}(P)} \mathsf{d}_\alpha(p,\mathsf{SH}(Q))$.*

Proof. Note that $\mathsf{S}(Q) \subseteq \mathsf{S}(P)$ since $Q \subseteq P$, and so

$$\mathsf{d}_{\mathsf{H}_\alpha}(\mathsf{SH}(P),\mathsf{SH}(Q)) = \max_{p\in\mathsf{SH}(P)} \mathsf{d}_\alpha(p,\mathsf{SH}(Q)).$$

As $\mathsf{S}(P) \subseteq P \subseteq \mathsf{SH}(P)$ we therefore have

$$\mathsf{d}_{\mathsf{H}_\alpha}(\mathsf{SH}(P),\mathsf{SH}(Q)) \geq \max_{p\in P} \mathsf{d}_\alpha(p,\mathsf{SH}(Q)) \geq \max_{p\in\mathsf{S}(P)} \mathsf{d}_\alpha(p,\mathsf{SH}(Q)).$$

For the other direction, for any point $p \in P$, let $f(p) \in \mathsf{S}(P)$ be any point that dominates p. For any point $q \in Q$, it follows by Lemma 4 that $\vec{\mathsf{d}}_\alpha(q,p) \leq \vec{\mathsf{d}}_\alpha(q,f(p))$ and as such taking the minimum over Q we get that $\vec{\mathsf{d}}_\alpha(Q,p) \leq \vec{\mathsf{d}}_\alpha(Q,f(p))$. Now taking the maximum over $p \in P$ we get that

$$\max_{p\in P} \mathsf{d}_\alpha(p,\mathsf{SH}(Q)) = \max_{p\in P} \vec{\mathsf{d}}_\alpha(Q,p) \leq \max_{p\in P} \vec{\mathsf{d}}_\alpha(Q,f(p)) \leq \max_{p\in\mathsf{S}(P)} \vec{\mathsf{d}}_\alpha(Q,p) = \max_{p\in\mathsf{S}(P)} \mathsf{d}_\alpha(p,\mathsf{SH}(Q)).$$

Similarly, because for each point $x \in \mathsf{SH}(P)$ there is some point $p \in P$ dominating x we can argue that $\mathsf{d}_{\mathsf{H}_\alpha}(\mathsf{SH}(P),\mathsf{SH}(Q)) \leq \max_{p\in P} \mathsf{d}_\alpha(p,\mathsf{SH}(Q))$. ◀

Note we can assume that any optimal solution Q to Problem 1 lies on the staircase $\mathsf{S}(P)$, since otherwise we can choose a dominating point on $\mathsf{S}(P)$ for each point of Q, defining a new set Q' such that $\mathsf{SH}(Q') \supseteq \mathsf{SH}(Q)$ and so $\max_{p\in\mathsf{S}(P)} \mathsf{d}_\alpha(p, \mathsf{SH}(Q')) \leq \max_{p\in\mathsf{S}(P)} \mathsf{d}_\alpha(p, \mathsf{SH}(Q))$. Thus the above implies that in Problem 1 we can consider the original input P as consisting entirely of staircase points (i.e., $\mathsf{S}(P) = P$), since otherwise as an initial step we can throw out all non-staircase points. Moreover, since $\mathsf{d}_{\mathsf{H}_\alpha}(\mathsf{SH}(P), \mathsf{SH}(Q)) = \max_{p\in P} \mathsf{d}_\alpha(p, \mathsf{SH}(Q))$ $= \max_{p\in P} \vec{\mathsf{d}}_\alpha(Q, p)$, the above implies Problem 1 can easily be seen as an instance of asymmetric k-center, for our asymmetric distance function $\vec{\mathsf{d}}_\alpha(\cdot,\cdot)$.

3 Hardness

In this section we mention APX-hardness results for the k-staircase problem for $d \geq 3$ using the L_2 norm. For the L_∞ norm, the hardness proof by Koltun and Papadimitriou [21] can be used to show that the problem is NP-hard, but there is no known APX-hardness proof.

Our main result for all $d \geq 4$ is,

▶ **Theorem 7.** *The k-staircase problem for the L_2 norm is* NP*-hard to approximate in $\mathbb{R}^d$, for any $d \geq 4$, within a factor better than $\sqrt{2+\sqrt{3}} \approx 1.932$.*

Proof. Since the results hold only for the L_2 norm, we skip the subscript 2 from the statements. We show the following claim which is required to establish the gap-preserving nature of a reduction from the k-center problem in $\mathbb{R}^2$ to the k-staircase problem in $\mathbb{R}^4$. We prove that there is a mapping $\phi : \mathbb{R}^2 \to \mathbb{R}^4$, such that for any two points $p, q \in \mathbb{R}^2$ we have that, $\|p - q\| = \vec{\mathsf{d}}(\phi(p), \phi(q)) = \vec{\mathsf{d}}(\phi(q), \phi(p))$.

The mapping ϕ is defined by $\phi((x, y)) = (x, -x, y, -y)$. Given $p = (p_x, p_y)$ and $q = (q_x, q_y)$, we have that $\|p - q\| = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$. Now, $\phi(p) - \phi(q) = (p_x - q_x, q_x - p_x, p_y - q_y, q_y - p_y)$. If $p_x \neq q_x$, exactly one of $p_x - q_x$ and $q_x - p_x$ is greater than 0. A similar statement is true for $p_y - q_y, q_y - p_y$. Thus, $||\phi(p) - \phi(q)||^+ = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$. If $p_x = q_x$, then neither of $(p_x - q_x)^2, (q_x - p_x)^2$ appear in the expression for $||\phi(p) - \phi(q)||^+$ but there is no contribution to $\|p - q\|$ from the x-coordinate as well. A similar statement is true for the y-coordinate. Thus, $\vec{\mathsf{d}}(\phi(p), \phi(q)) = ||\phi(p) - \phi(q)||^+ = \|p - q\|$. An analogous argument shows that $\|p - q\| = ||\phi(p) - \phi(q)||^- = ||\phi(q) - \phi(p)||^+ = \vec{\mathsf{d}}(\phi(q), \phi(p))$ as well. The claim follows.

The k-center problem in $\mathbb{R}^2$ is polynomial time reducible to the k-staircase problem in $\mathbb{R}^4$. Indeed, for a set P of n points in $\mathbb{R}^2$ we consider the set $P' = \{\phi(p) : p \in P\} \subset \mathbb{R}^4$. Clearly P' can be constructed in $O(n)$ time. Moreover, our claim implies that for any (at most) k points $p_1, \ldots, p_k \in P$ which give a clustering radius R, the corresponding points $\phi(p_1), \ldots, \phi(p_k)$ will give a value of R for the clustering radius in the k-staircase problem for P'. (Interestingly note that in fact all the mapped points lie on the staircase.) The minimum clustering radii thus are equal. Any APX-hardness result for the k-center problem in $\mathbb{R}^2$ can be combined with this polynomial time reduction to give the same hardness for the k-staircase problem in $\mathbb{R}^4$. Feder and Greene [16] proved that the k-center problem in $\mathbb{R}^2$ cannot be approximated to a factor better than $\frac{(1+\sqrt{7})}{2} \approx 1.823$. There is also such a result by Mentzer [27], although it is not that well known. Mentzer shows that the k-center problem in $\mathbb{R}^2$ is hard to approximate within a factor of $\sqrt{2+\sqrt{3}} \approx 1.932$. Using Mentzer's result [27] we conclude the result of the theorem. ◀

For $d = 3$, a non-trivial approximate embedding argument and Mentzer's result [27] implies,

▶ **Theorem 8.** *The k-staircase problem for the L_2 norm is* NP*-hard to approximate in $\mathbb{R}^3$ within a factor better than $\frac{\sqrt{2+\sqrt{3}}}{\sqrt{2+\sqrt{2}}} \approx 1.0455$.*

The proof of the last theorem can be found in [22].

Finally, we would like to point out the paper by Chuzhoy et al. [14], who showed that the asymmetric k-center problem is hard to approximate up to a factor $\log^* n - \Theta(1)$ unless NP $\subseteq$ DTIME$(n^{\log\log n})$. An interesting open question is if their reduction can be used to show that our k-staircase problem is hard to approximate by a factor better than $\log^* n$. The authors construct a complicated graph for their reduction. In order to use their construction we would have to embed their graph to the Euclidean space, such that i) the nodes are represented by points on the staircase, and ii) capture their (directed) graph distances with our geometric directed distances. Because of the complexity of their graph we could not find such a geometric embedding, and hence we leave it as an open problem. For details see the full version of the paper [22].

4 Exact Algorithm in 2D

In this section we present a polynomial time exact algorithm for the k-staircase problem in 2 dimensions for any L_α norm with $\alpha \geq 1$.

▶ **Theorem 9.** *Given a set of n points $P \subset \mathbb{R}^2$, an integer parameter $k \leq n$, and a real parameter $\alpha \geq 1$, a set $Q \subseteq P$ of size at most k can be computed in $O(n \log^3 n)$ time, such that $\mathsf{d}_{\mathsf{H}_\alpha}(\mathsf{SH}(P), \mathsf{SH}(Q))$ is minimized.*

Intuitively, the staircase problem is easier in 2D because of the ordering of the points along the x and the y axis: If the x-coordinate of $p \in \mathsf{S}(P)$ is smaller than the x-coordinate of $q \in \mathsf{S}(P)$, then the y-coordinate of p is larger than the y-coordinate of q. Let p_1, p_2 denote the x, y-coordinates of a point p and let $p, q \in \mathsf{S}(P)$ be two points in the staircase such that $p_1 \geq q_1$ and $p_2 \leq q_2$. Then, for any $\alpha, \beta \geq 1$ we have that $\vec{\mathsf{d}}_\alpha(p,q) = ||p-q||_\alpha^+ = \left((p_1 - q_1)^\alpha\right)^{1/\alpha} = p_1 - q_1 = \left((p_1 - q_1)^\beta\right)^{1/\beta} = \vec{\mathsf{d}}_\beta(p,q)$. Thus, all the norms are equivalent, and so without loss of generality, in what follows, $\alpha = 1$. We now describe our algorithm. Assume that we have a procedure Cover(P, r), which given a radius $r \geq 0$ returns a set $Q_r \subseteq \mathsf{S}(P)$ of minimum size satisfying $\mathsf{d}_{\mathsf{H}_\alpha}(\mathsf{SH}(P), \mathsf{SH}(Q_r)) \leq r$. Notice that by definition r^* is the minimum value of r such that $|Q_r| \leq k$. Since r^* is one of the $\Theta(n^2)$ distances $\vec{\mathsf{d}}(p_i, p_j)$, we could run a binary search over all possible distances to find r^*. For a given distance r we would need to decide if $r < r^*$ or $r \geq r^*$. To do this, we use Cover(P, r) and check if the set Q_r it returns has $|Q_r| \leq k$. A naive implementation of this algorithm would enumerate all the $\Theta(n^2)$ distances, sort them, and do the binary search over them, but this would clearly take at least quadratic time. We now show how to implement the algorithm in near-linear time.

Imagine an array containing all the $\Theta(n^2)$ distances in sorted order. As remarked above, we cannot directly compute this array in near-linear time. However we do not need access to the entire array. A typical query during binary search is : "return the m-th element of this array". Assume that finding the m-th smallest distance among the points in $\mathsf{S}(P)$, and the procedure Cover$(\cdot)$, both take $O(n \operatorname{polylog}(n))$ time. Then, all the $O(\log n)$ distance queries for the binary search and all the calls to Cover$(\cdot)$ would take $O(n \operatorname{polylog}(n))$ time. Following this scheme, we now present an algorithm with overall $O(n \log^3 n)$ running time.

Algorithm 1: Cover(P, r).

Input : P, r
Output : $Q_r \subseteq P$ with $|Q_r| \leq k_r$ and $\mathsf{d}_{\mathsf{H}_\alpha}(\mathsf{SH}(P), \mathsf{SH}(Q_r)) \leq r$

```
V = ∅
for p ∈ S(P) do
    V_p = {q | d⃗(p, q) ≤ r,  q ∈ S(P)}
    V = V ∪ {V_p}
F = Min_Set_Cover(S(P), V)
Q = {p | V_p ∈ F}
Return Q
```

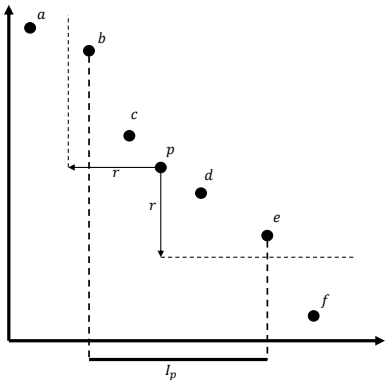


Figure 4.1 The interval I_p is defined by the x-coordinate of point b and of point e.

The procedure Cover(P, r). Let k_r be the smallest size of a set of centers Q_r such that $\mathsf{d}_{\mathsf{H}_\alpha}(\mathsf{SH}(P), \mathsf{SH}(Q_r)) \leq r$. The pseudocode of Cover(P, r) can be seen in Algorithm 1.

Correctness. For each point $p \in \mathsf{S}(P)$, the set V_p contains the points that are covered by p within radius r. Then we solve the min-set-cover instance where the elements are the points in $\mathsf{S}(P)$ and V is the family of sets. If the solution to the min-set-cover is optimal, the correctness of the algorithm follows.

Even though min-set-cover is an NP-hard problem, some special instances can be solved optimally in polynomial time. We show that the set system $(\mathsf{S}(P), V)$ can be mapped to a set system where the elements are points on a line and the sets are intervals intersected with the point set. This special case of the min-set-cover problem is the well known unweighted interval point cover problem (IPCP) and for n intervals and n points it can be solved optimally in $O(n \log n)$ time [4, 35, 20]. The mapping of points is simple. For a point $p = (p_1, p_2)$ it is just the projection of p to p_1 on the x-axis. Let P_1 denote this projected set of points. The following lemma shows that the set of points q for which $\vec{\mathsf{d}}_\alpha(p, q) \leq r$ corresponds to an interval on the x-axis.

▶ **Lemma 10.** *Let $p, q, z \in \mathsf{S}(P)$. If q_1 is between p_1 and z_1 and $\vec{\mathsf{d}}_\alpha(p, z) \leq r$ then $\vec{\mathsf{d}}_\alpha(p, q) \leq r$.*

Proof. We show it for the case where $p_1 \leq q_1 \leq z_1$, since the case of $z_1 \leq q_1 \leq p_1$ is similar. Since $p, q, z \in \mathsf{S}(P)$ and $p_1 \leq q_1 \leq z_1$, it also holds that $p_2 \geq q_2 \geq z_2$. We have that $\vec{\mathsf{d}}_\alpha(p, z) = p_2 - z_2 \leq r$. We conclude that $\vec{\mathsf{d}}_\alpha(p, q) = p_2 - q_2 \leq p_2 - z_2 \leq r$. ◀

The interval I_p for a point p can be computed as follows. The left end-point is just $p_1 - r$ but we can clip it to a point in P_1 by doing a successor search for $p_1 - r$ among the set of points P_1. The right end-point corresponds to the x-coordinate q_1 of a point $q = (q_1, q_2)$ such that $q_1 \geq p_1$ and is the largest value such that $p_2 - q_2 \leq r$. To compute this we can do a predecessor search for $p_2 + r$ in the projection of points P_2 on the y-axis. Once we find this point q_2, the left end-point of I_p is determined by q_1. In Figure 4.1, we show an example of the interval I_p for a point p. Using the above construction and Lemma 10 we can argue that each set V_p corresponds to an interval I_p on the x-axis and hence the set cover instance defined by $(\mathsf{S}(P), V)$ is an instance of the IPCP.

Complexity of Cover(P, r). We can compute the staircase $\mathsf{S}(P)$ of P in $O(n \log n)$ time. The (sorted) projected point sets P_1, P_2 as above can be constructed in additional $O(n)$ time. For each point p the set V_p is specified by the interval I_p which needs two predecessor

searches. Thus each such interval can be computed in $O(\log n)$ time. The min-set-cover for the IPCP instance can be computed in $O(n \log n)$ time. Thus, overall Cover(P, r) takes $O(n \log n)$ time.

Search over the possible radii. Let $Z = n(n-1)$ denote the total number of distances $\vec{\mathsf{d}}_\alpha(p, q)$ between points $p, q \in P$. As remarked above, given m with $1 \leq m \leq Z$, we need a method to compute the m-th smallest distance in this set. We now show how to do this in near-linear time, more specifically, $O(n \log^2 n)$ time. We first discuss another problem which will be used. Let A, B be two sorted arrays of real numbers with length M each. A classic exercise is to compute the m-th smallest element in the union of the two arrays in sublinear time and it is well known that it can be computed in $O(\log M)$ time. The C implementation of such an algorithm can be found in [22]. This algorithm makes at most $O(\log M)$ comparisons between the elements of A, B. The set R of all the Z distances $\vec{\mathsf{d}}_\alpha(p, q)$ can be written as the union of two sets A_x and A_y defined as, $A_x = \{p_1 - q_1 \mid p_1 \geq q_1\}$, and $A_y = \{p_2 - q_2 \mid p_2 \geq q_2\}$. Notice that $|A_x| = |A_y| = Z/2$, so let $M = Z/2$. So, our goal is to compute the m-th smallest element of $A_x \cup A_y$. To use the algorithm just mentioned, we need access to the i-th smallest element of A_x (or a similar query for A_y). Fortunately, we do not need to explicitly compute A_x, A_y. In [31, 32] the authors show that the i-th smallest L_1 distance of n points on a line can be computed in $O(n \log n)$ time. The distances in A_x, A_y can be seen as the L_1 distances among the projections of $\mathsf{S}(P)$ on the two axes, namely, among the points of P_1 and P_2. Since the algorithm of [31, 32] needs to run at most $O(\log M)$ times, the m-th smallest distance in R can be computed in $O(n \log n \log M) = O(n \log^2 n)$ time. The pseudocode of the overall procedure described above is given in [22].

Overall complexity. The $O(\log n)$ iterations of the procedure Cover(P, r) take $O(n \log^2 n)$ time overall, and the $O(\log n)$ queries for the m-th smallest distance overall take $O(n \log^3 n)$ time. This is the dominating term in the running time.

5 Approximate K-Staircase

In this section we give approximation algorithms for the k-staircase problem. First, we show a general algorithm that runs in quadratic time, and then we give a faster approximation algorithm for fixed d.

5.1 An approximation algorithm

In Section 2 we showed that the k-staircase problem is a special case of the asymmetric k-center problem. Previously, [30] gave an $O(\log^* n)$-approximation for the asymmetric k-center problem, so this immediately implies an $O(\log^* n)$ approximation for Problem 1 for any L_α norm. The proof of the next theorem can be found in [22].

▶ **Theorem 11.** *There is an $O(\log^*(n))$-approximation algorithm for the k-staircase problem with $O(n^2(d + \log^2 n))$ running time.*

One can also parameterize on the value of k, in which case [5] provided an $O(\log^* k)$-approximation. Their algorithm requires solving $O(\log n)$ linear programs, so while their algorithm runs in polynomial time, the precise asymptotic time was not stated.

▶ **Theorem 12.** *There exists an $O(\log^*(k))$-approximation algorithm for the k-staircase problem that runs in polynomial time.*

The $O(\log^* n)$-approximation algorithm of [30] can be implemented by performing a binary search over all the $\Theta(n^2)$ inter-point distances. For a value r, deciding if $r < r^*$ or $r \geq r^*$ requires solving a set cover problem. (This is similar to what we do for the exact algorithm in $2D$ in Section 4, except that the set cover instance in general is not solved exactly but only approximately.) In general, even constructing the set cover instance can take quadratic time, and thus approximately solving the entire problem in near-linear time seems difficult. We next use several geometric ideas to come up with a faster bi-criteria approximation (approximation on both k and radius) for any L_α norm, if d is fixed.

5.2 A faster approximation algorithm for fixed d

In this subsection we show a faster approximation algorithm for the k-staircase problem if the dimension d is fixed. Because of the space limit, here we only give a brief overview of our approach. In [22] we provide all the details and the proofs of our method. The main result of this section is the following theorem.

▶ **Theorem 13.** *Given a set of n points $P \subset \mathbb{R}^d$, for constant d, and an integer parameter $k \leq n$, a set $Q \subseteq P$ of size at most $2k$ can be computed in $O(kn\,\mathrm{polylog}(n))$ time, such that $\mathsf{d}_{H_\alpha}(\mathsf{SH}(P), \mathsf{SH}(Q)) = O(r^*_\alpha \log^* n)$, where $r^*_\alpha = \min_{Q \subseteq P, |Q| \leq k} \mathsf{d}_{H_\alpha}(\mathsf{SH}(P), \mathsf{SH}(Q))$.*

Our approach is to first get a fast approximation algorithm for the L_∞ norm and then argue that the same solution is a good approximation for any L_α. Intuitively, this will work due to the known inequality, $\|p-q\|_\infty \leq \|p-q\|_\alpha \leq d^{1/\alpha} \|p-q\|_\infty$, for any $p, q \in \mathbb{R}^d$ and any $\alpha \geq 1$, i.e., the L_∞ and any L_α norm differ by a factor that depends only on d and α. Next, we only focus on the L_∞ norm and we denote $r^* = \min_{Q \subseteq P, |Q| \leq k} \mathsf{d}_{H_\infty}(\mathsf{SH}(P), \mathsf{SH}(Q))$ (unlike in the other sections where we use r^* as the optimum distance for any L_α norm).

Our algorithm follows the structure of the algorithm in [30] for the asymmetric k center problem, which has two parts[5]: 1) A decider, such that given a distance r it either returns r is less than r^* or it returns a set $Q \subseteq P$ which is an $O(\log^* n)$-approximation. Their algorithm recursively solves many instances of the set cover problem using the greedy approximation algorithm [15]. Notice that a center in the asymmetric k-center problem covers a set of points within distance r so a set cover instance is indeed an intuitive way to model their problem. 2) A search over all $O(n^2)$ pair distances to find the smallest one where the decider returns an $O(\log^* n)$-approximation.

Both 1) and 2) can be made faster because we are focusing on the L_∞ norm for our k-staircase problem. 1) The sets defined by the set cover instances in [30] correspond to half-open boxes in $\mathbb{R}^d$ for the L_∞ norm. Using geometric data structures, like range trees [1], we can thus run the greedy algorithm for the set cover problem without explicitly constructing the set cover instances, i.e., without finding the points in P contained in each half-open box. 2) Instead of computing the quadratic number of all pairwise directed distances, we use the notion of Well-Separated Pair Decomposition (WSPD) [10, 18, 19, 29]. For a set of n points in $\mathbb{R}^d$, for fixed d, a WSPD can approximate all pairwise Euclidean distances with a near-linear number of distances. Unfortunately, our distance function $\vec{\mathsf{d}}_\infty(\cdot,\cdot)$ is not a metric and as such the known WSPD constructions are not useful. The main idea we use is to project the points to all d-axes separately, and construct the WSPD for each of the projected point sets. The key observation allowing us to construct such WSPDs is that for L_∞ the r^* distance is one of the pairwise distances among the projected 1-dimensional points.

[5] The authors in [30] do not present it this way, however it makes our algorithm easier to explain.

We note that our algorithm finds a set Q with at most $2k$ points. In order to guarantee at most k points with an approximation factor of $O(\log^* n)$ in [30] they make use of the notion of so called CCV points. These are defined and extended in the next section to get a heuristic that works very well in practice. Finally, note that our algorithm works for any dimension d. If d is not a constant, however, then we have an $O(d^{1/\alpha} r^*_\alpha \log^* n)$-approximation algorithm that runs in $O(\text{poly}(d)nk \log^{\text{poly}(d)} n)$ time, where $\text{poly}(d)$ is a linear polynomial of d.

6 A practical heuristic

Here we present a fast heuristic for the k-staircase problem, and provide experimental results showing its efficacy on both real and synthetic data. For simplicity, we present the result only for the L_2 norm and in the full version of the paper [22] we describe how to extend it to any L_α. Missing proofs of all lemmas in this section can be found in [22].

We first provide some intuition for our heuristic. As in the previous sections we look for r^* and the set of k centers using binary search. For a query radius r, we need to be able to decide if $r < r^*$, or $r \geq r^*$. The heuristic attempts to test this, by adapting an approach to solve the standard symmetric k-center problem. The algorithm for the symmetric k-center problem is iterative, and in each iteration it picks some arbitrary center p and removes all points in a ball of radius $2r$ around p. If the procedure stops within k iterations then clearly $r^* \leq 2r$. Otherwise $r^* > r$, because if $r^* \leq r$, then one can argue in each iteration this procedure entirely removes at least one optimal cluster which has not been completely covered yet. To see this, let p_i be the center that the algorithm selected in the ith round. Let o_i be the center from the optimal solution covering p_i. (Since p_i still exists, o_i's cluster has not been fully covered yet.) All remaining points in o_i's cluster are within distance r^* from o_i, which is within r^* from p_i, and hence all remaining points are covered by the $2r$ ball around p_i if $r \geq r^*$. If distances are asymmetric, however, this argument breaks since while p_i is within distance r^* from o_i, this does not imply o_i is within r^* from p_i. To deal with asymmetry, [30] defined the notion of *center capturing vertices* (CCV), where a point p is CCV(r) if whenever the distance from another point to p is less than r, then the distance from p to that point is also at most r. That is, at the resolution of the radius r, the distances involving that point look symmetric. If we could always find a CCV(r) point among the remaining ones, then by the directed triangle inequality the argument would still be valid. However, for general asymmetric distance functions such CCV points may not exist, so we define a new relaxed notion of CCV points allowing imbalance in the directed distances.

▶ **Definition 14** (λ-CCV(r))**.** *Given a point set $P \subset \mathbb{R}^d$, and real numbers $r \geq 0, \lambda > 0$, $p \in P$ is a λ-CCV(r) point, if for all $q \in P$ with $\vec{\mathsf{d}}_2(q,p) \leq r$ it is also true that $\vec{\mathsf{d}}_2(p,q) \leq \lambda r$.*

Note that a 1-CCV(r) point is the same as a CCV(r) point in [30]. Also note that if we can always find a λ-CCV(r) point rather than a CCV(r) point as desired above we will end up with a $(1+\lambda)$ rather than 2 approximation. If r is clear from the context, a CCV(r) (resp. λ-CCV(r)) point is denoted as a CCV (resp. λ-CCV) point. Ultimately our new heuristic works by selecting appropriate λ-CCV points. It is not clear whether such points even exist, however, the following lemma confirms that there is a point that is a $\sqrt{d-1}$-CCV(r) point for *any radius* r, and further, such a point can be found easily (in $O(dn)$ time). For any $u \in \mathbb{R}^d$ define its *weight* as $\mathsf{w}(u) = u_1 + u_2 + \ldots + u_d$.

▶ **Lemma 15.** *For any point set $P \subset \mathbb{R}^d$ with n points, the point $p \in P$ with minimum weight $\mathsf{w}(p)$ is a $\sqrt{d-1}$-CCV(r) point for any distance $r \geq 0$.*

Algorithm 2: **MinWeight**(P, k, r).

Input : P, k, r
Output : Either "$r < r^*$" or a set $Q \subseteq P$ with $|Q| \leq k$ such that $\mathsf{d}_{H_2}(\mathsf{SH}(P), \mathsf{SH}(Q)) \leq (1+\sqrt{d-1})r$

```
X ← P, Q ← ∅
while |Q| ≤ k AND X ≠ ∅ do
    p = argmin_{p∈X} w(p)
    Q ← Q ∪ {p}
    X ← X \ {q ∈ X | d⃗_2(p, q) ≤ (1+√(d−1))r}
if |Q| > k then
    Return "r < r*"
else
    Return Q
```

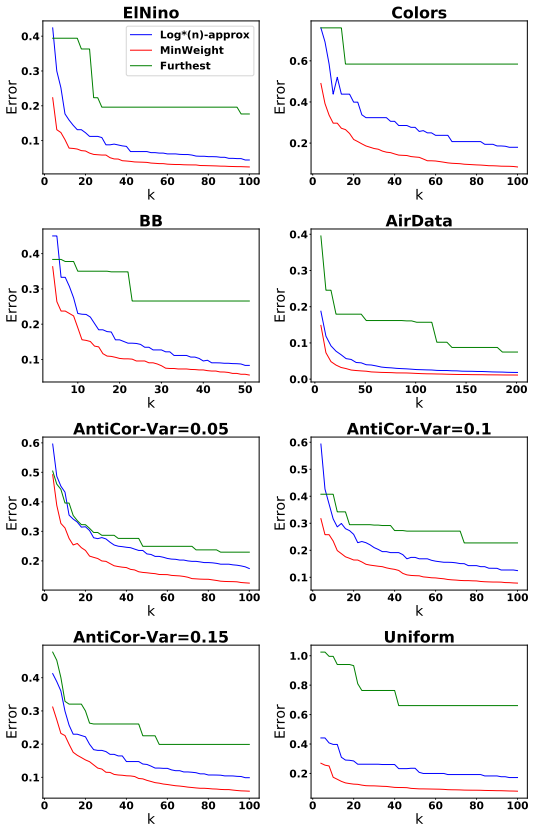


Figure 6.1 Approximation error for L_2 norm.

The decision procedure. To recollect, let r^* be the optimum radius for Problem 1 on a given set P of n points. Our aim is to build a decision procedure which we later use to do a binary search for r^*. Specifically, Algorithm 2 shows a fast procedure **MinWeight**(P, k, r), which either outputs "$r < r^*$" or returns a set Q with $|Q| \leq k$. We were not able to argue that the procedure is always correct when "$r < r^*$" is output, hence the term "heuristic", though the experiments below show that in practice this does not at all seem to be an issue. On the other hand if a set Q is output, we have the following.

▶ **Lemma 16.** *If* ***MinWeight***(P, k, r) *returns a set* $Q \subseteq P$*, then* $|Q| \leq k$ *and* $\mathsf{d}_{H_2}(\mathsf{SH}(P), \mathsf{SH}(Q)) \leq (1+\sqrt{d-1})r$*. The algorithm runs in* $O(dnk)$ *time.*

To see the obstacle in arguing that the algorithm is correct when "$r < r^*$" is output, let Z be an optimal r^* radius solution to Problem 1. We would like to argue that every point $z \in Z$ is at most $\sqrt{d-1}r^*$ from one of the chosen points, as this would imply a $(1+\sqrt{d-1})r^*$ radius covering. However, the issue is that the points we choose in each iteration are $\sqrt{d-1}$-CCV with respect to only the *remaining points* (Lemma 15), for any r. Specifically, consider a point p which is r^*-covered by a point o_i in the optimum solution. It is possible that o_i may cause p to *not* be a $\sqrt{d-1}$-CCV. However, if o_i was removed in some earlier iteration, then later p may become a $\sqrt{d-1}$-CCV point. This could be a problem, as in the worst case it means it is possible that o_i is not $\sqrt{d-1}r^*$-covered by p, and hence all points r^*-covered by o_i are not $(1+\sqrt{d-1})r^*$-covered by p, as we would like to argue.

6.1 Experiments

This section compares the performance of three algorithms: the heuristic MinWeight, the $\log^*(n)$ asymmetric k-center algorithm ([30]) which we call $\text{Log}^*(n)$-approx, and another greedy algorithm Furthest described below. We test on several real and synthetic datasets and use the approximation measure defined in Problem 2 with the asymmetric distance function $\vec{\mathsf{d}}_2(\cdot,\cdot)$. Additionally, in [22], we compare MinWeight and $\text{Log}^*(n)$-approx to the optimal solution (found by brute force) for small synthetic datasets.

All algorithms are implemented in Python on 64-bit machine with 4 3500 MHz cores and 16GB of RAM with Ubuntu 17.10.

Although we do not report running times, the simplicity of MinWeight (Algorithm 2) should be noted. There are k iterations that consist of two steps: greedily choose a point to add to the solution; remove all points within some radius of that point. Not only do both steps take $O(dn)$ time, but they are easily parallelized. Though we don't claim optimal implementations, we mention MinWeight always ran faster than $\text{Log}^*(n)$-approx.

Implementation details.

Log*(n): The authors in [30] simply try all n^2 distance pairs to find the optimal radius r^*. As this is impractical for large datasets, both algorithms (MinWeight and $\text{Log}^*(n)$-approx) instead do a binary search over all distance pairs of the staircase points. Specifically, if the returned set of either algorithm for a given radius is larger than k, we drop the smaller distances; otherwise, we drop the larger distances.

MinWeight: As MinWeight can possibly find a solution of size k on a radius $r < r^*$, the binary search might choose a path where future guesses of r yield no solutions. We store the most recent solution so that if this happens, we can use the stored solution.

Furthest: This is a simple greedy algorithm that iteratively adds the point furthest from the current solution set. That is, given an input set P and a solution set Q_i in round i, it adds $\arg\max_{p\in P}\vec{\mathsf{d}}_2(Q_i,p)$ to Q_i to obtain Q_{i+1} (and repeats until $i=k$). This algorithm does not require searching for an optimal radius r^*.

Datasets. For the experiments we use five of the most common datasets that have been used in other papers related to finding representative points or staircase queries ([3, 6, 8, 13, 28, 34]). More specifically, we use the following datasets.

BB: This is a commonly used ([3, 6, 13]) basketball dataset where each point is a player and the attributes are five statistics: points, rebounds, blocks, assists and fouls. There are 21,961 points in 5-d with 200 points on the staircase.

ElNino: Oceanographic data from the Pacific Ocean, with attributes such as surface temperature, water temperature, and wind speed. There are 178,080 points in 5-d, with 1,183 points on the staircase. This dataset was used in [3, 13] for finding representative points.

AirData: On-time flight data published by the US Department of Transportation, contained in the AirData dataset [6]. Information gathered from 14 carriers flying in January 2015 includes departure delay, taxi in, taxi out, air time, distance, actual elapsed time, and arrival delay. This set has 458,311 points in 7-d, with 6,439 points on the staircase.

Colors: The Colors data set is also commonly used for evaluating staircase and regret sets [28]. The data derives from the HSV color space of a color image, and includes the standard deviation, skewness, and mean of each H, S, and V in the space.

AntiCor: This is a synthetic dataset of anti-correlated points. There are 100,000 points in 5-d that are generated as described in [8]. We test on different values (0.05, 0.1, 0.15) of the variance (Var) that determines the position of the plane. For Var=0.05, there are

7,941 staircase points; for Var=0.1, 1,275 staircase points; and for Var=0.15, 529 staircase points.

Uniform: This synthetic dataset has points sampled uniformly from the unit hypercube which is described in [8]. There are 200,000 points in 7-d with 6,585 points on the staircase.

Results. In each experiment (see Figure 6.1), MinWeight outperforms the other algorithms by at least 50% which suggests that, at least in practice, MinWeight does not erroneously return "$r < r^*$". Indeed, when the input set does not have the structure mentioned after Lemma 16 (where multiple points of an optimal solution are removed in one iteration), MinWeight achieves a $(1+\sqrt{d-1})$-approximation. Even with a $(1+\sqrt{d-1})$-approximation, MinWeight seems to exceed expectations compared against the $\text{Log}^*(n)$-approx. We observed that after the binary search, MinWeight would consistently reach a final guess of r that was 2-4 times smaller than $\text{Log}^*(n)$-approx. One difference between the two algorithms that could explain this is the fact that MinWeight updates its set of $\sqrt{d-1}$-CCV points with respect to the remaining points in each iteration. Specifically, a point which is not $\sqrt{d-1}$-CCV(r) can become $\sqrt{d-1}$-CCV(r) in a later iteration. This indicates that, compared against the 'static' CCV point set in $\text{Log}^*(n)$-approx, there are better points to be chosen conditioned on some set of the previous $\sqrt{d-1}$-CCV(r) points.

6.2 Comparison of algorithms on real data

There are several well known algorithms which attempt to output a set of representative points on the skyline. Comparing the outputs of these algorithms is tricky, as each tries to optimize a different metric. In an attempt at an objective method of comparison, we look at these algorithms and measures on a real data set. Specifically, we look at NBA player statistics over several seasons and use the various staircase algorithms to predict potential all star players in each season (a selection of outstanding players made by the US National Basketball Association). As discussed below, our algorithm consistently selects the most potential all stars, and moreover our error measure appears to most closely track how many potential all stars are selected. Specifically, we compare our Hausdorff measure and our algorithm MinWeight with the measures and the algorithms of the three most cited papers for staircase representation, namely, KolPap [21], k-center [34], and Max-cover [25].

The dataset. For this experiment, we used the Basketball season data set that contains the statistics of each NBA player for each regular season. We use the statistics of each player for five seasons, 2004-2005, ..., 2008-2009. For each player we keep six statistics: total points, number of rebounds, number of assists, number of steals, number of blocks, and the ratio of field goals made over the number of attempted field goals.

The experiment. For each of the five seasons, we run the algorithms MinWeight, KolPap, k-center, and Max-cover and record the $k = 10$ representative players returned, respectively.

The comparison method. Notice that each algorithm returns a set of 10 players S. Each algorithm is attempting to optimize a different measure of error, thus in order to evaluate the quality of the returned set S, we need some neutral method which is different from all the metrics. Towards this end, we define a function $B(S)$ attempting to capture how "good" the set S is, where $B(S) = |T \cap S|$, and T is the set of 50 players selected by NBA experts as

the best in a season that fans could vote to participate in the NBA All Star game.[6] Namely, $B(S)$ returns the number of players in S that were selected as potential players for the NBA All Star game. Such a set of players T always contains the "best" players as selected by NBA experts so we feel it is a fair way to compare the results of the algorithms, so we consider T as the ground truth. Since the notion of skyline approximation is more natural and better suited for users, another possible or even better evaluation would have been by running a real user study.

Assessment of the metric itself. We also obtain some evidence that our metric is somewhat more desirable than the other metrics. Ideally, we would like to have an error function $F(S)$ (to be optimized by an algorithm) such that if $B(S_1) \geq B(S_2)$ then $F(S_1) \leq F(S_2)$, i.e., the better the set is, the smaller the error becomes, or in other words the error has an *inverse relationship* with the quality of the solution.

Format of Table 6.1. To evaluate an algorithm which returns a set S, we show in the table the intersection $B(S) = |T \cap S|$ of S with the ground truth, as well as the error of S for all measures under consideration. So each entry of the table for algorithm X and season Y presents data in the format $B(S)[a, b, c, d]$ where S is the set returned by X and a is the error of S in the Hausdorff distance (our measure), b is the error as per the metric of KolPap, c is the error as per the metric of k-center, and d is the error as per the metric of Max-Cover (the number of uncovered items in P by S).

Table 6.1 Statistics and errors by using different measures and algorithms for $k = 10$.

	2004-2005	2005-2006	2006-2007	2007-2008	2008-2009
MinWeight	8[0.1, 0.25, 0.9, 15]	9[0.18, 0.52, 0.76, 32]	8[0.17, 0.25, 0.79, 31]	8[0.12, 0.67, 0.72, 16]	8[0.09, 0.33, 0.63, 22]
KolPap	7[0.16, 0.18, 0.88, 17]	7[0.22, 0.33, 0.62, 16]	7[0.17, 0.25, 0.79, 30]	6[0.3, 0.34, 0.74, 66]	8[0.14, 0.23, 0.8, 9]
k-center	7[0.34, 0.51, 0.48, 27]	5[0.31, 1.2, 0.50, 57]	5[0.43, 0.75, 0.55, 68]	6[0.34, 0.93, 0.51, 19]	6[0.22, 0.71, 0.49, 49]
Max-cover	7[0.32, 0.44, 0.77, 2]	5[0.39, 0.69, 0.69, 4]	6[0.44, 0.75, 0.67, 1]	6[0.38, 0.59, 0.52, 1]	5[0.37, 0.48, 0.66, 3]

Analysis of the results. From Table 6.1 we can derive the following conclusions: (I) The MinWeight algorithm always returns better results (based on the function B), and (II) the Hausdorff error has an inverse relationship with the quality of the result $B(S)$. We obtain several data points to test this relationship. For a fixed season, the various algorithms return some set S.We plot the points $(B(S), \text{error})$ for each algorithm using the set S it returns and for the errors under the different metrics. If we look at the errors for a fixed measure, the Hausdorff distance error measure satisfies the inverse relationship in almost all the cases. The rest of the measures defined by the previous works do not seem to have this desirable inverse relationship. In the full version of the paper [22] we show the results for $k = 15$ and we provide more figures showing the relationship of the quality $B(S)$ with the error measures.

7 Conclusions

Our work suggests several directions for future research. As shown, the k-staircase problem is an instance of the asymmetric k-center problem and this implies an $O(\log^* n)$-approximation. However, it is an open question whether we can exploit the geometric properties of the

[6] We use `https://www.basketball-reference.com/allstar/` to get the players selected by the experts.

problem to improve the approximation. In particular, can we get an $f(d)$-approximation for the k-staircase problem, where $f(d)$ is a function only depending on the dimension? The practical success of our heuristic and the properties discussed in Section 6 gives some indication this may be possible. Even if this is not possible, one could consider whether relaxing the constraint on having exactly k points to $O(k)$ points helps, i.e., whether a bi-criteria approximation is possible. Alternatively, one can consider if there is an $O(\log^* k)$-approximation for the k-staircase problem that does not require solving multiple LP's [5].

References

1 Pankaj K. Agarwal. Range Searching. In *Handbook of Discrete and Computational Geometry, Second Edition.*, pages 809–837. CRC, 2004. doi:10.1201/9781420035315.ch36.

2 Pankaj K. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan. Geometric approximation via coresets. *Combinatorial and computational geometry*, 52:1–30, 2005.

3 Pankaj K. Agarwal, Nirman Kumar, Stavros Sintos, and Subhash Suri. Efficient Algorithms for k-Regret Minimizing Sets. In *16th International Symposium on Experimental Algorithms, SEA 2017, June 21-23, 2017, London, UK*, pages 7:1–7:23, 2017. doi:10.4230/LIPIcs.SEA.2017.7.

4 Mugurel Ionut Andreica, Eliana-Dina Tirsa, Cristina Teodora Andreica, Romulus Andreica, and Mihai Aristotel Ungureanu. Optimal geometric partitions, covers and K-centers. *arXiv preprint arXiv:0908.3652*, 2009.

5 Aaron Archer. Two O ($\log^*$ k)-Approximation Algorithms for the Asymmetric k-Center Problem. In *Integer Programming and Combinatorial Optimization, 8th International IPCO Conference, Utrecht, The Netherlands, June 13-15, 2001, Proceedings*, pages 1–14, 2001. doi:10.1007/3-540-45535-3_1.

6 Abolfazl Asudeh, Azade Nazi, Nan Zhang, and Gautam Das. Efficient Computation of Regret-ratio Minimizing Set: A Compact Maxima Representative. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 821–834, 2017. doi:10.1145/3035918.3035932.

7 Avrim Blum, Sariel Har-Peled, and Benjamin Raichel. Sparse Approximation via Generating Point Sets. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 548–557, 2016. doi:10.1137/1.9781611974331.ch40.

8 Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The Skyline Operator. In *Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany*, pages 421–430, 2001. doi:10.1109/ICDE.2001.914855.

9 Greg Van Buskirk, Benjamin Raichel, and Nicholas Ruozzi. Sparse Approximate Conic Hulls. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 2531–2541, 2017. URL: http://papers.nips.cc/paper/6847-sparse-approximate-conic-hulls.

10 Paul B. Callahan and S. Rao Kosaraju. A Decomposition of Multidimensional Point Sets with Applications to k-Nearest-Neighbors and n-Body Potential Fields. *J. ACM*, 42(1):67–90, 1995. doi:10.1145/200836.200853.

11 Wei Cao, Jian Li, Haitao Wang, Kangning Wang, Ruosong Wang, Raymond Chi-Wing Wong, and Wei Zhan. k-Regret Minimizing Set: Efficient Algorithms and Hardness. In *20th International Conference on Database Theory, ICDT 2017, March 21-24, 2017, Venice, Italy*, pages 11:1–11:19, 2017. doi:10.4230/LIPIcs.ICDT.2017.11.

12 Timothy M. Chan, Kasper Green Larsen, and Mihai Patrascu. Orthogonal range searching on the RAM, revisited. In *Proceedings of the 27th ACM Symposium on Computational Geometry, Paris, France, June 13-15, 2011*, pages 1–10, 2011. doi:10.1145/1998196.1998198.

13 Sean Chester, Alex Thomo, S. Venkatesh, and Sue Whitesides. Computing k-Regret Minimizing Sets. *PVLDB*, 7(5):389–400, 2014. doi:10.14778/2732269.2732275.

14 Julia Chuzhoy, Sudipto Guha, Eran Halperin, Sanjeev Khanna, Guy Kortsarz, Robert Krauthgamer, and Joseph Naor. Asymmetric k-center is $\log^* n$-hard to approximate. *J. ACM*, 52(4):538–551, 2005. doi:10.1145/1082036.1082038.

15 Vasek Chvátal. A Greedy Heuristic for the Set-Covering Problem. *Math. Oper. Res.*, 4(3):233–235, 1979. doi:10.1287/moor.4.3.233.

16 Tomás Feder and Daniel H. Greene. Optimal Algorithms for Approximate Clustering. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 434–444, 1988. doi:10.1145/62212.62255.

17 Harold N. Gabow, Jon Louis Bentley, and Robert Endre Tarjan. Scaling and Related Techniques for Geometry Problems. In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1984, Washington, DC, USA*, pages 135–143, 1984. doi:10.1145/800057.808675.

18 Sariel Har-Peled. *Geometric approximation algorithms*, volume 173. American mathematical society Boston, 2011.

19 Sariel Har-Peled and Manor Mendel. Fast Construction of Nets in Low-Dimensional Metrics and Their Applications. *SIAM J. Comput.*, 35(5):1148–1184, 2006. doi:10.1137/S0097539704446281.

20 Refael Hassin and Arie Tamir. Improved complexity bounds for location problems on the real line. *Operations Research Letters*, 10(7):395–402, 1991.

21 Vladlen Koltun and Christos H. Papadimitriou. Approximately Dominating Representatives. In *Database Theory - ICDT 2005, 10th International Conference, Edinburgh, UK, January 5-7, 2005, Proceedings*, pages 204–214, 2005. doi:10.1007/978-3-540-30570-5_14.

22 Nirman Kumar, Benjamin Raichel, Stavros Sintos, and Gregory Van Buskirk. Approximating Distance Measures for the Skyline. http://utdallas.edu/~benjamin.raichel/stair.pdf.

23 Nirman Kumar and Stavros Sintos. Faster Approximation Algorithm for the k-Regret Minimizing Set and Related Problems. In *Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments, ALENEX 2018, New Orleans, LA, USA, January 7-8, 2018.*, pages 62–74, 2018. doi:10.1137/1.9781611975055.6.

24 H. T. Kung, Fabrizio Luccio, and Franco P. Preparata. On Finding the Maxima of a Set of Vectors. *J. ACM*, 22(4):469–476, 1975. doi:10.1145/321906.321910.

25 Xuemin Lin, Yidong Yuan, Qing Zhang, and Ying Zhang. Selecting Stars: The k Most Representative Skyline Operator. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, pages 86–95, 2007. doi:10.1109/ICDE.2007.367854.

26 Matteo Magnani, Ira Assent, and Michael L. Mortensen. Taking the Big Picture: representative skylines based on significance and diversity. *VLDB J.*, 23(5):795–815, 2014. doi:10.1007/s00778-014-0352-3.

27 S. Mentzer. Approximability of Metric Clustering Problems. preprint on researchgate.net, 2016. URL: https://www.researchgate.net/publication/242489373_Approximability_of_Metric_Clustering_Problems.

28 Danupon Nanongkai, Atish Das Sarma, Ashwin Lall, Richard J. Lipton, and Jun (Jim) Xu. Regret-Minimizing Representative Databases. *PVLDB*, 3(1):1114–1124, 2010. doi:10.14778/1920841.1920980.

29 Giri Narasimhan and Michiel H. M. Smid. *Geometric spanner networks*. Cambridge University Press, 2007.

30 Rina Panigrahy and Sundar Vishwanathan. An O ($\log^* n$) Approximation Algorithm for the Asymmetric p-Center Problem. *J. Algo.*, 27(2):259–268, 1998. doi:10.1006/jagm.1997.0921.

31 J. Salowe. *Selection Problems in Computational Geometry*. PhD thesis, Rutgers University, 1987.

32 Jeffrey S. Salowe. L-Infinity Interdistance Selection by Parametric Search. *Inf. Process. Lett.*, 30(1):9–14, 1989. doi:10.1016/0020-0190(89)90166-X.

33 Malene Søholm, Sean Chester, and Ira Assent. Maximum Coverage Representative Skyline. In *Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016.*, pages 702–703, 2016. doi:10.5441/002/edbt.2016.95.

34 Yufei Tao, Ling Ding, Xuemin Lin, and Jian Pei. Distance-Based Representative Skyline. In *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*, pages 892–903, 2009. doi:10.1109/ICDE.2009.84.

35 Stan PM Van Hoesel and Albert PM Wagelmans. On the p-coverage problem on the real line. *Statistica Neerlandica*, 61(1):16–34, 2007.

36 Rui Xu and Donald C. Wunsch II. Survey of clustering algorithms. *IEEE Trans. Neural Networks*, 16(3):645–678, 2005. doi:10.1109/TNN.2005.845141.

Index-Based, High-Dimensional, Cosine Threshold Querying with Optimality Guarantees

Yuliang Li
Megagon Labs, Mountain View, California, USA
UC San Diego, San Diego, California, USA

Jianguo Wang
UC San Diego, San Diego, California, USA

Benjamin Pullman
UC San Diego, San Diego, California, USA

Nuno Bandeira
UC San Diego, San Diego, California, USA

Yannis Papakonstantinou
UC San Diego, San Diego, California, USA

Abstract

Given a database of vectors, a cosine threshold query returns all vectors in the database having cosine similarity to a query vector above a given threshold. These queries arise naturally in many applications, such as document retrieval, image search, and mass spectrometry. The present paper considers the efficient evaluation of such queries, providing novel optimality guarantees and exhibiting good performance on real datasets. We take as a starting point Fagin's well-known Threshold Algorithm (TA), which can be used to answer cosine threshold queries as follows: an inverted index is first built from the database vectors during pre-processing; at query time, the algorithm traverses the index partially to gather a set of candidate vectors to be later verified against the similarity threshold. However, directly applying TA in its raw form misses significant optimization opportunities. Indeed, we first show that one can take advantage of the fact that the vectors can be assumed to be normalized, to obtain an improved, tight stopping condition for index traversal and to efficiently compute it incrementally. Then we show that one can take advantage of data skewness to obtain better traversal strategies. In particular, we show a novel traversal strategy that exploits a common data skewness condition which holds in multiple domains including mass spectrometry, documents, and image databases. We show that under the skewness assumption, the new traversal strategy has a strong, near-optimal performance guarantee. The techniques developed in the paper are quite general since they can be applied to a large class of similarity functions beyond cosine.

2012 ACM Subject Classification Theory of computation → Data structures and algorithms for data management; Theory of computation → Database query processing and optimization (theory); Information systems → Nearest-neighbor search

Keywords and phrases Vector databases, Similarity search, Cosine, Threshold Algorithm

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.11

Related Version A full version of the paper is available at `https://arxiv.org/abs/1812.07695`.

Acknowledgements We are very grateful to Victor Vianu who helped us significantly improve the presentation of the paper. We also thank the anonymous reviewers for the very constructive and helpful comments. This work was supported in part by the National Science Foundation (NSF) under awards BIGDATA 1447943 and ABI 1759980, and by the National Institutes of Health (NIH) under awards P41GM103484 and R24GM127667.

1 Introduction

Given a database of vectors, a cosine threshold query asks for all database vectors with cosine similarity to a query vector above a given threshold.

This problem arises in many applications including document retrieval [11], image search [24], recommender systems [26] and mass spectrometry. For example, in mass spectrometry, billions of spectra are generated for the purpose of protein analysis [1, 25, 33]. Each spectrum is a collection of key-value pairs where the key is the mass-to-charge ratio of an ion contained in the protein and the value is the intensity of the ion. Essentially, each spectrum is a high-dimensional, non-negative and sparse vector with ∼2000 dimensions where ∼100 coordinates are non-zero.

Cosine threshold queries play an important role in analyzing such spectra repositories. Example questions include "is the given spectrum similar to any spectrum in the database?", spectrum identification (matching query spectra against reference spectra), or clustering (matching pairs of unidentified spectra) or metadata queries (searching for public datasets containing matching spectra, even if obtained from different types of samples). For such applications with a large vector database, it is critically important to process cosine threshold queries efficiently – this is the fundamental topic addressed in this paper.

▶ **Definition 1** (Cosine Threshold Query). *Let $\mathcal{D}$ be a collection of high-dimensional, non-negative vectors; $\mathbf{q}$ be a query vector; θ be a threshold $0 < \theta \leq 1$. Then the cosine threshold query returns the vector set $\mathcal{R} = \{\mathbf{s} | \mathbf{s} \in \mathcal{D}, \texttt{cos}(\mathbf{q}, \mathbf{s}) \geq \theta\}$. A vector $\mathbf{s}$ is called* θ-similar *to the query $\mathbf{q}$ if $\texttt{cos}(\mathbf{q}, \mathbf{s}) \geq \theta$ and the* score *of $\mathbf{s}$ is the value $\texttt{cos}(\mathbf{q}, \mathbf{s})$ when $\mathbf{q}$ is understood from the context.*

Observe that cosine similarity is insensitive to vector normalization. We will therefore assume without loss of generality that the database as well as query consist of unit vectors (otherwise, all vectors can be normalized in a pre-processing step).

In the literature, cosine threshold querying is a special case of Cosine Similarity Search (CSS) [31, 3, 26], where other aspects like approximate answers, top-k queries and similarity join are considered. Our work considers specifically CSS with exact, threshold and single-vector queries, which is the case of interest to many applications.

Because of the unit-vector assumption, the scoring function cos computes the dot product $\mathbf{q} \cdot \mathbf{s}$. Without the unit-vector assumption, the problem is equivalent to *inner product threshold querying*, which is of interest in its own right. Related work on cosine and inner product similarity search is summarized in Section 5.

In this paper we develop novel techniques for the efficient evaluation of cosine threshold queries. We take as a starting point the well-known Threshold Algorithm (TA), by Fagin et al. [16], because of its simplicity, wide applicability, and optimality guarantees. A review of the classic TA is provided in the full version of the paper.

A TA-like baseline index and algorithm and its shortcomings. The TA algorithm can be easily adapted to our setting, yielding a first-cut approach to processing cosine threshold queries. We describe how this is done and refer to the resulting index and algorithm as the *TA-like baseline*. Note first that cosine threshold queries use $\texttt{cos}(\mathbf{q}, \mathbf{s})$, which can be viewed as a particular family of functions $F(\mathbf{s}) = \mathbf{s} \cdot \mathbf{q}$ parameterized by $\mathbf{q}$, that are monotonic in $\mathbf{s}$ for unit vectors. However, TA produces the vectors with the top-k scores according to $F(\mathbf{s})$, whereas cosine threshold queries return all $\mathbf{s}$ whose score exceeds the threshold θ. We will show how this difference can be overcome straightforwardly.

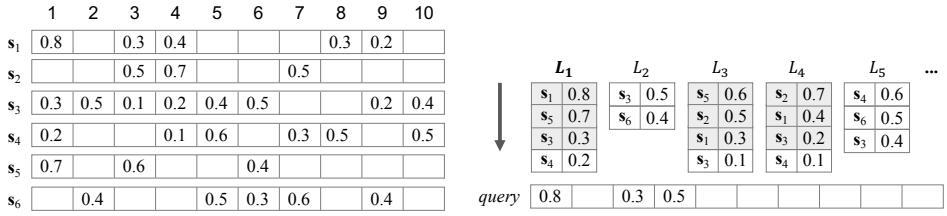


Figure 1 An example of cosine threshold query with six 10-dimensional vectors. The missing values are 0's. We only need to scan the lists L_1, L_3, and L_4 since the query vector has non-zero values in dimension 1, 3 and 4. For $\theta = 0.6$, the gathering phase terminates after each list has examined three entries (highlighted) because the score for any unseen vector is at most $0.8 \times 0.3 + 0.3 \times 0.3 + 0.5 \times 0.2 = 0.43 < 0.6$. The verification phase only needs to retrieve from the database those vectors obtained during the gathering phase, i.e., $\mathbf{s}_1$, $\mathbf{s}_2$, $\mathbf{s}_3$ and $\mathbf{s}_5$, compute the cosines and produce the final result.

A *baseline* index and algorithm inspired by TA can answer cosine threshold queries exactly without a full scan of the vector database for each query. In addition, the baseline algorithm enjoys the same instance optimality guarantee as the original TA. This baseline is created as follows. First, identically to the TA, the baseline index consists of one sorted list for each of the d dimensions. In particular, the i-th sorted list has pairs $(\mathsf{ref}(\mathbf{s}), \mathbf{s}[i])$, where $\mathsf{ref}(\mathbf{s})$ is a reference to the vector $\mathbf{s}$ and $\mathbf{s}[i]$ is its value on the i-th dimension. The list is sorted in descending order of $\mathbf{s}[i]$.[1]

Next, the baseline, like the TA, proceeds into a *gathering phase* during which it collects a complete set of references to candidate result vectors. The TA shows that gathering can be achieved by reading the d sorted lists from top to bottom and terminating early when a *stopping condition* is finally satisfied. The condition guarantees that any vector that has not been seen yet has no chance of being in the query result. The baseline makes a straightforward change to the TA's stopping condition to adjust for the difference between the TA's top-k requirement and the threshold requirement of the cosine threshold queries. In particular, in each round the baseline algorithm has read the first b entries of each index. (Initially it is $b = 0$.) If it is the case that $\mathsf{cos}(\mathbf{q}, [L_1[b], \ldots, L_d[b]]) < \theta$ then it is guaranteed that the algorithm has already read (the references to) all the possible candidates and thus it is safe to terminate the gathering phase, see Figure 1 for an example. Every vector $\mathbf{s}$ that appears in the j-th entry of a list for $j < b$ is a candidate.

In the next phase, called the *verification* phase, the baseline algorithm (again like TA) retrieves the candidate vectors from the database and checks which ones actually score above the threshold.

For inner product queries, the baseline algorithm's gathering phase benefits from the same $d \cdot \mathsf{OPT}$ instance optimality guarantee as the TA. Namely, the gathering phase will access at most $d \cdot \mathsf{OPT}$ entries, where OPT is the optimal index access cost. More specifically, the notion of OPT is the *minimal number of sequential accesses* of the sorted inverted index during the gathering phase for any TA-like algorithm applied to the specific query and index instance.

There is an obvious optimization: Only the k dimensions that have non-zero values in the query vector $\mathbf{q}$ should participate in query processing – this leads to a $k \cdot \mathsf{OPT}$ guarantee for inner product queries.[2] But even this guarantee loses its practical value when k is a large

[1] There is no need to include pairs with zero values in the list.
[2] This optimization is equally applicable to the TA's problem: Scan only the lists that correspond to

Table 1 Summary of theoretical results for the near-convex case.

	Stopping Condition		Traversal Strategy	
	Baseline	*This work*	*Baseline*	*This work*
Inner Product	Tight		$m \cdot \mathsf{OPT}$	$\mathsf{OPT} + c$
Cosine	Not tight	Tight	NA	$\mathsf{OPT}(\theta - \epsilon) + c$

number. In the mass spectrometry scenario k is $\sim$100. In document similarity and image similarity cases it is even higher.

For cosine threshold queries, the $k \cdot \mathsf{OPT}$ guarantee no longer holds. The baseline fails to utilize the unit vector constraint to reach the stopping condition faster, resulting in an unbounded gap from OPT because of the unnecessary accesses (see Appendix C of the full version).[3] Furthermore, the baseline fails to utilize the skewing of the values in the vector's coordinates (both of the database's vectors and of the query vector) and the linearity of the similarity function. Intuitively, if the query's weight is concentrated on a few coordinates, the query processing should overweight the respective lists and may, thus, reach the stopping condition much faster than reading all relevant lists in tandem.

We retain the baseline's index and the gathering-verification structure which characterizes the family of TA-like algorithms. The decision to keep the gathering and verification stages separate is discussed in Section 2. We argue that this algorithmic structure is appropriate for cosine threshold queries, because further optimizations that would require merging the two phases are only likely to yield marginal benefits. Within this framework, we reconsider

1. *Traversal strategy optimization*: A *traversal strategy* determines the order in which the gathering phase proceeds in the lists. In particular, we allow the gathering phase to move deeper in some lists and less deep in others. For example, the gathering phase may have read at some point $b_1 = 106$ entries from the first list, $b_2 = 523$ entries from the second list, etc. Multiple traversal strategies are possible and, generally, each traversal strategy will reach the stopping condition with a different configuration of $[b_1, b_2, \ldots, b_n]$. The traversal strategy optimization problem asks that we efficiently identify a traversal path that minimizes the access cost $\sum_{i=1}^{d} b_i$. To enable such optimization, we will allow lightweight additions to the baseline index.
2. *Stopping condition optimization*: We reconsider the stopping condition so that it takes into account (a) the specifics of the `cos` function and (b) the unit vector constraint. Moreover, since the stopping condition is tested frequently during the gathering phase, it has to to be evaluated very efficiently. Notice that optimizing the stopping condition is independent of the traversal strategy or skewness assumptions about the data.

Contributions and summary of results.

- We present a stopping condition for early termination of the index traversal (Section 3). We show that the stopping condition is *complete* and *tight*, informally meaning that (1) for any traversal strategy, the gathering phase will produce a candidate set containing all the vectors θ-similar to the query, and (2) the gathering terminates as soon as no more θ-similar vectors can be found (Theorem 7). In contrast, the stopping condition of the (TA-inspired) baseline is complete but not tight (Theorem 27 in the full version). The proposed stopping condition takes into account that all database vectors

dimensions that actually affect the function F.

[3] Notice, the unit vector constraint enables inference about the collective weight of the unseen coordinates of a vector.

are normalized and reduces the problem to solving a special quadratic program (Equation 1) that guarantees both completeness and tightness. While the new stopping condition prunes further the set of candidates, it can also be efficiently computed in $\mathcal{O}(\log d)$ time using incremental maintenance techniques.

- We introduce the *hull-based* traversal strategies that exploit the skewness of the data (Section 4). In particular, skewness implies that each sorted list L_i is "mostly convex", meaning that the shape of L_i is approximately the *lower convex hull* constructed from the set of points of L_i. This technique is quite general, as it can be extended to the class of *decomposable functions* which have the form $F(\mathbf{s}) = f_1(\mathbf{s}[1]) + \ldots + f_d(\mathbf{s}[d])$ where each f_i is non-decreasing.[4] Consequently, we provide the following optimality guarantee for inner product threshold queries: The number of accesses executed by the gathering phase (i.e., $\sum_{i=1}^{d} b_i$) is at most $\mathsf{OPT} + c$ (Theorem 16 and Corollary 18), where OPT is the number of accesses by the optimal strategy and c is the max distance between two vertices in the lower convex hull. Experiments show that in multiple real-world cases, c is a very small fraction of OPT.
- Despite the fact that cosine and its tight stopping condition are not decomposable, we show that the hull-based strategy can be adapted to cosine threshold queries by approximating the tight stopping condition with a carefully chosen decomposable function. We show that when the approximation is at most ϵ-away from the actual value, the access cost is at most $\mathsf{OPT}(\theta - \epsilon) + c$ (Theorem 20) where $\mathsf{OPT}(\theta - \epsilon)$ is the optimal access cost on the same query $\mathbf{q}$ with the threshold lowered by ϵ and c is a constant similar to the above decomposable cases. Experiments show that the adjustment ϵ is very small in practice, e.g., 0.1. We summarize these new results in Table 1.

The paper is organized as follows. We introduce the algorithmic framework and basic definitions in Section 2. Section 3 and 4 discuss the technical developments as we mentioned above. Finally, we discuss related work in Section 5 and conclude in Section 6.

2 Algorithmic Framework

In this section, we present a Gathering-Verification algorithmic framework to facilitate optimizations in different components of an algorithm with a TA-like structure. We start with notations summarized in Table 2.

To support fast query processing, we build an index for the database vectors similar to the original TA. The basic index structure consists of a set of 1-dimensional sorted lists (a.k.a inverted lists in web search [11]) where each list corresponds to a vector dimension and contains vectors having non-zero values on that dimension, as mentioned earlier in Section 1. Formally, for each dimension i, L_i is a list of pairs $\{(\mathsf{ref}(\mathbf{s}), \mathbf{s}[i]) \mid \mathbf{s} \in \mathcal{D} \wedge \mathbf{s}[i] > 0\}$ sorted in descending order of $\mathbf{s}[i]$ where $\mathsf{ref}(\mathbf{s})$ is a reference to the vector $\mathbf{s}$ and $\mathbf{s}[i]$ is its value on the i-th dimension. In the interest of brevity, we will often write $(\mathbf{s}, \mathbf{s}[i])$ instead of $(\mathsf{ref}(\mathbf{s}), \mathbf{s}[i])$. As an example in Figure 1, the list L_1 is built for the first dimension and it includes 4 entries: $(\mathbf{s}_1, 0.8)$, $(\mathbf{s}_5, 0.7)$, $(\mathbf{s}_3, 0.3)$, $(\mathbf{s}_4, 0.2)$ because $\mathbf{s}_1$, $\mathbf{s}_5$, $\mathbf{s}_3$ and $\mathbf{s}_4$ have non-zero values on the first dimension.

Next, we show the Gathering-Verification framework (Algorithm 1) that operates on the index structure. The framework has two phases: gathering and verification.

[4] The inner product threshold problem is the special case where $f_i(\mathbf{s}[i]) = q_i \cdot \mathbf{s}[i]$.

Table 2 Notation.

$\mathcal{D}$	the vector database
d	the number of dimensions
$\mathbf{s}$ (bold font)	a data vector
$\mathbf{q}$ (bold font)	a query vector
$\mathbf{s}[i]$ or s_i	the i-th dimensional value of $\mathbf{s}$
$\lvert\mathbf{s}\rvert$	the L1 norm of $\mathbf{s}$
$\lVert\mathbf{s}\rVert$	the L2 norm of $\mathbf{s}$
θ	the similarity threshold
$\mathtt{cos}(\mathbf{p},\mathbf{q})$	the cosine of $\mathbf{p}$ and $\mathbf{q}$
L_i	the inverted list of the i-th dimension
$\mathbf{b}=(b_1,\ldots,b_d)$	a position vector
$L_i[b_i]$	the b_i-th value of L_i
$L[\mathbf{b}]$	the vector $(L_1[b_1],\ldots,L_d[b_d])$

Algorithm 1: Gathering-Verification Framework.

```
   input  : (D, {L_i}_{1≤i≤d}, q, θ)
   output : R the set of θ-similar vectors
   /* Gathering phase                                   */
1  Initialize b = (b_1, ..., b_d) = (0, ..., 0);
   // φ(·) is the stopping condition
2  while φ(b) = false do
       // T(·) is the traversal strategy to
          determine which list to access
          next
3      i ← T(b);
4      b_i ← b_i + 1;
5      Put the vector s in L_i[b_i] to the candidate
         pool C;
   /* Verification phase                                */
6  R ← {s | s ∈ C ∧ cos(q, s) ≥ θ};
7  return R;
```

Gathering phase (line 1 to line 5)**.** The goal of the gathering phase is to collect a complete set of candidate vectors while minimizing the number of accesses to the sorted lists. The algorithm maintains a *position vector* $\mathbf{b}=(b_1,\ldots,b_d)$ where each b_i indicates the current position in the inverted list L_i. Initially, the position vector $\mathbf{b}$ is $(0,\ldots,0)$. Then it traverses the lists according to a *traversal strategy* that determines the list (say L_i) to be accessed next (line 3). Then it advances the pointer b_i by 1 (line 4) and adds the vector $\mathbf{s}$ referenced in the entry $L_i[b_i]$ to a candidate pool $\mathcal{C}$ (line 5). The traversal strategy is usually stateful, which means that its decision is made based on information that has been observed up to position $\mathbf{b}$ and its past decisions. For example, a strategy may decide that it will make the next 20 moves along dimension 6 and thus it needs state in order to remember that it has already committed to 20 moves on dimension 6.

The gathering phase terminates once a *stopping condition* is met. Intuitively, based on the information that has been observed in the index, the stopping condition checks if a complete set of candidates has already been found.

Next, we formally define stopping conditions and traversal strategies. As mentioned above, the input of the stopping condition and the traversal strategy is the information that has been observed up to position $\mathbf{b}$, which is formally defined as follows.

▶ **Definition 2.** *Let $\mathbf{b}$ be a position vector on the inverted index $\{L_i\}_{1\le i\le d}$ of a database $\mathcal{D}$. The partial observation at $\mathbf{b}$, denoted as $\mathcal{L}(\mathbf{b})$, is a collection of lists $\{\hat{L}_i\}_{1\le i\le d}$ where for every $1\le i\le d$, $\hat{L}_i=[L_i[1],\ldots,L_i[b_i]]$.*

▶ **Definition 3.** *Let $\mathcal{L}(\mathbf{b})$ be a partial observation and $\mathbf{q}$ be a query with similarity threshold θ. A **stopping condition** is a boolean function $\varphi(\mathcal{L}(\mathbf{b}),\mathbf{q},\theta)$ and a **traversal strategy** is a function $\mathcal{T}(\mathcal{L}(\mathbf{b}),\mathbf{q},\theta)$ whose domain is $[d]$[5]. When clear from the context, we denote them simply by $\varphi(\mathbf{b})$ and $\mathcal{T}(\mathbf{b})$ respectively.*

Verification phase (line 6)**.** The verification phase examines each candidate vector $\mathbf{s}$ seen in the gathering phase to verify whether $\mathtt{cos}(\mathbf{q},\mathbf{s})\ge\theta$ by accessing the database. Various

[5] $[d]$ is the set $\{1,\ldots,d\}$

techniques [31, 4, 26] have been proposed to speed up this process. Essentially, instead of accessing all the d dimensions of each $\mathbf{s}$ and $\mathbf{q}$ to compute exactly the cosine similarity, these techniques decide θ-similarity by performing a partial scan of each candidate vector. We review these techniques, which we refer to as *partial verification*, in Appendix B. Additionally, as a novel contribution, we show that in the presence of data skewness, partial verification can have a near-constant performance guarantee (Theorem 25 of the full version) for each candidate.

▶ **Theorem 4** (Informal). *For most skewed vectors, θ-similarity can be computed at constant time.*

Remark on optimizing the gathering phase. Due to these optimization techniques, the *number of sequential accesses* performed during the gathering phase becomes the dominating factor of the overall running time. This reason behind is that the number of sequential accesses is strictly greater than the number of candidates that need to be verified so reducing the sequential access cost also results in better performance of the verification phase. In practice, we observed that the sequential cost is indeed dominating: for 1,000 queries on 1.2 billion vectors with similarity threshold 0.6, the sequential gathering time is 16 seconds and the verification time is only 4.6 seconds. Such observation justifies our goal of designing a traversal strategy with near-optimal sequential access cost, as the dominant cost concerns the gathering stage.

Remark on the suitability of TA-like algorithms. One may wonder whether algorithms that start the gathering phase NOT from the top of the inverted lists may outperform the best TA-like algorithm. In particular, it appears tempting to start the gathering phase from the point closest to q_i in each inverted list and traverse towards the two ends of each list. Appendix E of the full version proves why this idea can lead to poor performance. In particular, we prove that in a general setting, the computation of a tight and complete stopping condition (formally defined in Definition 5 and 6) becomes NP-HARD since it needs to take into account constraints from two pointers (forward and backward) for each inverted list. Furthermore, in many applications, the data skewing leads to small savings from pruning the top area of each list, since the top area is sparsely populated - unlike the densely populated bottom area of each list. Thus it is not justified to use an expensive gathering phase algorithm for small savings.

Section 5.1 reviews additional prior work ideas [31, 32] that avoid traversing some top/bottom regions of the inverted index. Such ideas may provide additional optimizations to TA-like algorithms in variations and/or restrictions of the problem (e.g., a restriction that the threshold is very high) and thus they present future work opportunities in closely related problems.

3 Stopping condition

In this section, we introduce a fine-tuned stopping condition that satisfies the tight and complete requirements to early terminate the index traversal.

First, the stopping condition has to guarantee *completeness* (Definition 5), i.e. when the stopping condition φ holds on a position $\mathbf{b}$, the candidate set $\mathcal{C}$ must contain all the true results. Note that since the input of φ is the partial observation at $\mathbf{b}$, we must guarantee that for all possible databases $\mathcal{D}$ consistent with the partial observation $\mathcal{L}(\mathbf{b})$, the candidate set $\mathcal{C}$ contains all vectors in $\mathcal{D}$ that are θ-similar to the query $\mathbf{q}$. This is equivalent to require

that if a unit vector $\mathbf{s}$ is found below position $\mathbf{b}$ (i.e. $\mathbf{s}$ does not appear above $\mathbf{b}$), then $\mathbf{s}$ is NOT θ-similar to $\mathbf{q}$. We formulate this as follows.

▶ **Definition 5** (Completeness). *Given a query $\mathbf{q}$ with threshold θ, a position vector $\mathbf{b}$ on index $\{L_i\}_{1\leq i\leq d}$ is complete iff for every unit vector $\mathbf{s}$, $\mathbf{s} < L[\mathbf{b}]$ implies $\mathbf{s}\cdot\mathbf{q} < \theta$. A stopping condition $\varphi(\cdot)$ is complete iff for every $\mathbf{b}$, $\varphi(\mathbf{b}) = \texttt{True}$ implies that $\mathbf{b}$ is complete.*

The second requirement of the stopping condition is *tightness*. It is desirable that the algorithm terminates immediately once the candidate set $\mathcal{C}$ contains a complete set of candidates, such that no additional unnecessary access is made. This can reduce not only the number of index accesses but also the candidate set size, which in turn reduces the verification cost. Formally,

▶ **Definition 6** (Tightness). *A stopping condition $\varphi(\cdot)$ is tight iff for every complete position vector $\mathbf{b}$, $\varphi(\mathbf{b}) = \texttt{True}$.*

It is desirable that a stopping condition be both complete and tight. However, as we shown in Appendix C of the full version, the baseline stopping condition $\varphi_{\mathsf{BL}} = \big(\mathbf{q}\cdot L[\mathbf{b}] < \theta\big)$ is complete but not tight as it does not capture the unit vector constraint to terminate as soon as no unseen unit vector can satisfy $\mathbf{s}\cdot\mathbf{q} \geq \theta$. Next, we present a new stopping condition that is both complete and tight.

To guarantee tightness, one can check at every snapshot during the traversal whether the current position vector $\mathbf{b}$ is complete and stop once the condition is true. However, directly testing the completeness is impractical since it is equivalent to testing whether there exists a real vector $\mathbf{s} = (s_1, \ldots, s_d)$ that satisfies the following following set of quadratic constraints:

$$(a)\quad \sum_{i=1}^{d} s_i \cdot q_i \geq \theta, \qquad (b)\quad s_i \leq L_i[b_i],\ \forall\ i \in [d], \qquad \text{and} \qquad (c)\quad \sum_{i=1}^{d} s_i^2 = 1. \tag{1}$$

We denote by $\mathbf{C}(\mathbf{b})$ (or simply $\mathbf{C}$) the set of $\mathbb{R}^d$ points defined by the above constraints. The set $\mathbf{C}(\mathbf{b})$ is infeasible (i.e. there is no satisfying $\mathbf{s}$) if and only if $\mathbf{b}$ is complete, but directly testing the feasibility of $\mathbf{C}(\mathbf{b})$ requires an expensive call to a quadratic programming solver. Depending on the implementation, the running time can be exponential or of high-degree polynomial [10]. We address this challenge by deriving an equivalently strong stopping condition that guarantees tightness and is efficiently testable:

▶ **Theorem 7.** *Let τ be the solution of the equation $\sum_{i=1}^{d} \min\{q_i\cdot\tau, L_i[b_i]\}^2 = 1$ and*

$$\mathsf{MS}(L[\mathbf{b}]) = \sum_{i=1}^{d} \min\{q_i\cdot\tau, L_i[b_i]\}\cdot q_i \tag{2}$$

called the max-similarity*. The stopping condition $\varphi_{\mathsf{TC}}(\mathbf{b}) = (\mathsf{MS}(L[\mathbf{b}]) < \theta)$ is tight and complete.*

Proof. The tight and complete stopping condition is obtained by applying the Karush-Kuhn-Tucker (KKT) conditions [23] for solving nonlinear programs. We first formulate the set of constraints in (1) as an optimization problem over $\mathbf{s}$:

$$\text{maximize}\quad \sum_{i=1}^{d} s_i\cdot q_i \qquad \text{subject to}\quad \sum_{i=1}^{d} s_i^2 = 1 \quad \text{and} \quad s_i \leq L_i[b_i], \quad \forall i \in [d] \tag{3}$$

So checking whether $\mathbf{C}$ is feasible is equivalent to verifying whether the maximal $\sum_{i=1}^{d} s_i \cdot q_i$ is at least θ. So it is sufficient to show that $\sum_{i=1}^{d} s_i \cdot q_i$ is maximized when $s_i = \min\{q_i \cdot \tau, L_i[b_i]\}$ as specified above.

The KKT conditions of the above maximization problem specify a set of necessary conditions that the optimal $\mathbf{s}$ needs to satisfy. More precisely, let

$$L(\mathbf{s}, \mu, \lambda) = \sum_{i=1}^{d} s_i q_i - \sum_{i=1}^{d} \mu_i (L_i[b_i] - s_i) - \lambda \left(\sum_{i=1}^{d} s_i^2 - 1 \right)$$

be the Lagrangian of (3) where $\lambda \in \mathbb{R}$ and $\mu \in \mathbb{R}^d$ are the Lagrange multipliers. Then,

▶ **Lemma 8** (derived from KKT). *The optimal* $\mathbf{s}$ *in (3) satisfies the following conditions:*

$\nabla_{\mathbf{s}} L(\mathbf{s}, \mu, \lambda) = 0$	*(Stationarity)*
$\mu_i \geq 0, \ \forall\ i \in [d]$	*(Dual feasibility)*
$\mu_i (L_i[b_i] - s_i) = 0, \ \forall\ i \in [d]$	*(Complementary slackness)*

in addition to the constraints in (3) (called the Primal feasibility *conditions).*

By the Complementary slackness condition, for every i, if $\mu_i \neq 0$ then $s_i = L_i[b_i]$. If $\mu_i = 0$, then from the Stationarity condition, we know that for every i, $q_i + \mu_i - 2\lambda \cdot s_i = 0$ so $s_i = q_i/2\lambda$. Thus, the value of s_i is either $L_i[b_i]$ or $q_i/2\lambda$.

If $L_i[b_i] < q_i/2\lambda$ then since $s_i \leq L_i[b_i]$, the only possible case is $s_i = L_i[b_i]$. For the remaining dimensions, the objective function $\sum_{i=1}^{d} s_i \cdot q_i$ is maximized when each s_i is proportional to q_i, so $s_i = q_i/2\lambda$. Combining these two cases, we have $s_i = \min\{q_i/2\lambda, L_i[b_i]\}$.

Thus, for the λ that satisfies $\sum_{i=1}^{d} \min\{q_i/2\lambda, L_i[b_i]\}^2 = 1$, the objective function $\sum_{i=1}^{d} s_i \cdot q_i$ is maximized when $s_i = \min\{q_i/2\lambda, L_i[b_i]\}$ for every i. The theorem is obtained by letting $\tau = 1/2\lambda$. ◀

Remark of φ_{TC}. The tight stopping condition φ_{TC} computes the vector $\mathbf{s}$ below $L(\mathbf{b})$ with the maximum cosine similarity $\mathsf{MS}(L[\mathbf{b}])$ with the query $\mathbf{q}$. At the beginning of the gathering phase, $b_i = 0$ for every i so $\mathsf{MS}(L[\mathbf{b}]) = 1$ as $\mathbf{s}$ is not constrained. The cosine score is maximized when $\mathbf{s} = \mathbf{q}$ where $\tau = 1$. During the gathering phase, as b_i increases, the upper bound $L_i[b_i]$ of each s_i decreases. When $L_i[b_i] < q_i$ for some i, s_i can no longer be q_i. Instead, s_i equals $L_i[b_i]$, the rest of $\mathbf{s}$ increases proportional to $\mathbf{q}$ and τ increases. During the traversal, the value of τ monotonically increases and the score $\mathbf{s}(L[\mathbf{b}])$ monotonically decreases. This is because the space for $\mathbf{s}$ becomes more constrained by $L(\mathbf{b})$ as the pointers move deeper in the inverted lists.

Testing the tight and complete condition φ_{TC} requires solving τ in Theorem (7), for which a direct application of the bisection method takes $\mathcal{O}(d)$ time. We show a novel efficient algorithm (Appendix D) in the full version of the paper based on incremental maintenance which takes only $\mathcal{O}(\log d)$ time for each test of φ_{TC}.

▶ **Theorem 9.** *The stopping condition* $\varphi_{\mathsf{TC}}(\mathbf{b})$ *can be incrementally computed in* $\mathcal{O}(\log d)$ *time.*

4 Near-Optimal Traversal Strategy

Given the inverted lists index and a query, there can be many stopping positions that are both complete and tight. To optimize the performance, we need a traversal strategy that reaches one such position as fast as possible. Specifically, the goal is to design a traversal

strategy $\mathcal{T}$ that minimizes $|\mathbf{b}| = \sum_{i=1}^{d} b_i$ where $\mathbf{b}$ is the first position vector satisfying the tight and complete stopping condition if $\mathcal{T}$ is followed. Minimizing $|\mathbf{b}|$ also reduces the number of collected candidates, which in turn reduces the cost of the verification phase. We call $|\mathbf{b}|$ the *access cost* of the strategy $\mathcal{T}$. Formally,

▶ **Definition 10** (Access Cost). *Given a traversal strategy $\mathcal{T}$, we denote by $\{\mathbf{b}_i\}_{i\geq 0}$ the sequence of position vectors obtained by following $\mathcal{T}$. The access cost of $\mathcal{T}$, denoted by $\mathsf{cost}(\mathcal{T})$, is the minimal k such that $\varphi_{\mathsf{TC}}(\mathbf{b}_k) = \texttt{True}$. Note that $\mathsf{cost}(\mathcal{T})$ also equals $|\mathbf{b}_k|$.*

▶ **Definition 11** (Instance Optimality). *Given a database $\mathcal{D}$ with inverted lists $\{L_i\}_{1\leq i\leq d}$, a query vector $\mathbf{q}$ and a threshold θ, the optimal access cost $\mathsf{OPT}(\mathcal{D}, \mathbf{q}, \theta)$ is the minimum $\sum_{i=1}^{d} b_i$ for position vectors $\mathbf{b}$ such that $\varphi_{\mathsf{TC}}(\mathbf{b}) = \texttt{True}$. When it is clear from the context, we simply denote $\mathsf{OPT}(\mathcal{D}, \mathbf{q}, \theta)$ as $\mathsf{OPT}(\theta)$ or OPT.*

At a position $\mathbf{b}$, a traversal strategy makes its decision locally based on what has been observed in the inverted lists up to that point, so the capability of making globally optimal decisions is limited. As a result, traversal strategies are often designed as simple heuristics, such as the lockstep strategy in the baseline approach. The lockstep strategy has a $d \cdot \mathsf{OPT}$ near-optimal bound which is loose in the high-dimensionality setting.

In this section, we present a traversal strategy for cosine threshold queries with tighter near-optimal bound by taking into account that the index values are skewed in many realistic scenarios. We approach the (near-)optimal traversal strategy in two steps.

First, we consider the simplified case with the unit-vector constraint ignored so that the problem is reduced to inner product queries. We propose a general traversal strategy that relies on convex hulls pre-computed from the inverted lists during indexing. During the gathering phase, these convex hulls are accessed as auxiliary data during the traversal to provide information on the increase/decrease rate towards the stopping condition. The hull-based traversal strategy not only makes fast decisions (in $\mathcal{O}(\log d)$ time) but is near-optimal (Corollary 18) under a reasonable assumption. In particular, we show that if the distance between any two consecutive convex hull vertices of the inverted lists is bounded by a constant c, the access cost of the strategy is at most $\mathsf{OPT} + c$. Experiments on real data show that this constant is small in practice.

The hull-based traversal strategy is quite general, as it applies to a large class of functions beyond inner product called the *decomposable functions*, which have the form $\sum_{i=1}^{d} f_i(s_i)$ where each f_i is a non-decreasing real function of a single dimension s_i. Obviously, for a fixed query $\mathbf{q}$, the inner product $\mathbf{q} \cdot \mathbf{s}$ is a special case of decomposable functions, where each $f_i(s_i) = q_i \cdot s_i$. We show that the near-optimality result for inner product queries can be generalized to any decomposable function (Theorem 16).

Next, in Section 4.4, we consider the cosine queries by taking the normalization constraint into account. Although the function $\mathsf{MS}(\cdot)$ used in the tight stopping condition φ_{TC} is not decomposable so the same technique cannot be directly applied, we show that the hull-based strategy can be adapted by approximating $\mathsf{MS}(\cdot)$ with a decomposable function. In addition, we show that with a properly chosen approximation, the hull-based strategy is near-optimal with a small adjustment to the input threshold θ, meaning that the access cost is bounded by $\mathsf{OPT}(\theta - \epsilon) + c$ for a small ϵ (Theorem 20). Under the same experimental setting, we verify that ϵ is indeed small in practice.

4.1 Decomposable Functions

We start with defining the decomposable functions for which the hull-based traversal strategies can be applied:

▶ **Definition 12** (Decomposable Function). *A decomposable function $F(\mathbf{s})$ is a d-dimensional real function where $F(\mathbf{s}) = \sum_{i=1}^{d} f_i(s_i)$ and each f_i is a non-decreasing real function.*

Given a decomposable function F, the corresponding stopping condition is called a *decomposable condition*, which we define next.

▶ **Definition 13** (Decomposable Condition). *A decomposable condition φ_F is a boolean function $\varphi_F(\mathbf{b}) = \big(F(L[\mathbf{b}]) < \theta\big)$ where F is a decomposable function and θ is a fixed threshold.*

When the unit vector constraint is lifted, the decomposable condition is tight and complete for any scoring function F and threshold θ. As a result, the goal of designing a traversal strategy for F is to have the access cost as close as possible to OPT when the stopping condition is φ_F.

4.2 The max-reduction traversal strategy

To illustrate the high-level idea of the hull-based approach, we start with a simple greedy traversal strategy called the *Max-Reduction* traversal strategy $\mathcal{T}_{\mathsf{MR}}(\cdot)$. The strategy works as follows: at each snapshot, move the pointer b_i on the inverted list L_i that results in the maximal reduction on the score $F(L[\mathbf{b}])$. Formally, we define

$$\mathcal{T}_{\mathsf{MR}}(\mathbf{b}) = \operatorname*{argmax}_{1\le i\le d} \left(F(L[\mathbf{b}]) - F(L[\mathbf{b}+\mathbb{1}_i])\right) = \operatorname*{argmax}_{1\le i\le d} \left(f_i(L_i[b_i]) - f_i(L_i[b_i+1])\right)$$

where $\mathbb{1}_i$ is the vector with 1 at dimension i and 0's else where. Such a strategy is reasonable since one would like $F(L[\mathbf{b}])$ to drop as fast as possible, so that once it is below θ, the stopping condition φ_F will be triggered and terminate the traversal.

It is obvious that there are instances where the max-reduction strategy can be far from optimal, but is it possible that it is optimal under some assumption? The answer is positive: if for every list L_i, the values of $f_i(L_i[b_i])$ are decreasing at decelerating rate, then we can prove that its access cost is optimal. We state this ideal assumption next.

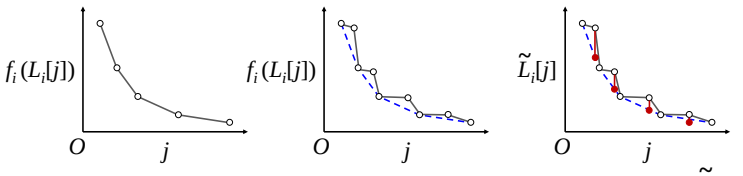
▶ **Assumption 1** (Ideal Convexity). *For every inverted list L_i, let $\Delta_i[j] = f_i(L_i[j]) - f_i(L_i[j+1])$ for $0 \le j < |L_i|$.[6] The list L_i is ideally convex if the sequence Δ_i is non-increasing, i.e., $\Delta_i[j+1] \le \Delta_i[j]$ for every j. Equivalently, the piecewise linear function passing through the points $\{(j, f_i(L_i[j]))\}_{0\le j\le |L_i|}$ is convex for each i. A database $\mathcal{D}$ is ideally convex if every L_i is ideally convex.*

An example of an inverted list satisfying the above assumption is shown in Figure 2(a). The max-reduction strategy $\mathcal{T}_{\mathsf{MR}}$ is optimal under the ideal convexity assumption:

▶ **Theorem 14** (Ideal Optimality). *Given a decomposable function F, for every ideally convex database $\mathcal{D}$ and every threshold θ, the access cost of $\mathcal{T}_{\mathsf{MR}}$ is exactly* OPT.

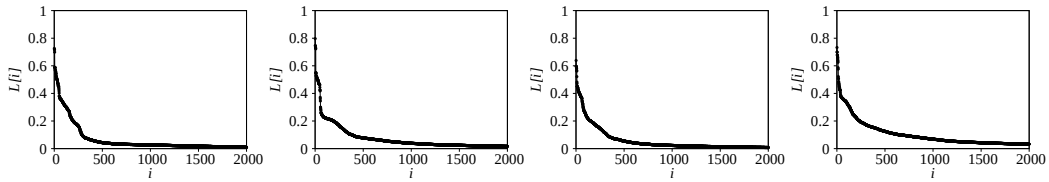
We prove Theorem 14 with a simple greedy argument (see Appendix F for more details): each move of $\mathcal{T}_{\mathsf{MR}}$ always results in the globally maximal reduction in the scoring function as guaranteed by the convexity condition.

[6] Recall that $L_i[0] = 1$.



(a) Convex (b) Near-convex (c) Constructing $\tilde{L}_i[j]$

Figure 2 Convexity and near-convexity.



(a) list L_0 (b) list L_1 (c) list L_2 (d) list L_3

Figure 3 The skewed inverted lists in mass spectrometry.

4.3 The hull-based traversal strategy

Theorem 14 provides a strong performance guarantee but the ideal convexity assumption is usually not true on real datasets. Without the ideal convexity assumption, the strategy suffers from the drawback of making locally optimal but globally suboptimal decisions. The pointer b_i to an inverted list L_i might never be moved if choosing the current b_i only results in a small decrease in the score $F(L[\mathbf{b}])$, but there is a much larger decrease several steps ahead. As a result, the $\mathcal{T}_{\mathsf{MR}}$ strategy has no performance guarantee in general.

In most practical scenarios that we have seen, we can bring the traversal strategy $\mathcal{T}_{\mathsf{MR}}$ to practicality by considering a relaxed version of Assumption 1. Informally, instead of assuming that each list $f_i(L_i)$ forms a convex piecewise linear function, we assume that $f_i(L_i)$ is "mostly" convex, meaning that if we compute the *lower convex hull* [14] of $f_i(L_i)$, the gap between any two consecutive vertices on the convex hull is small.[7] Intuitively, the relaxed assumption implies that the values at each list are decreasing at "approximately" decelerating speed. It allows list segments that do not follow the overall deceleration trend, as long as their lengths are bounded by a constant. We verified this property in the mass spectrometry dataset as illustrated in Figure 3, a document dataset, and an image dataset (see Appendix I of the full version for details).

▶ **Assumption 2** (Near-Convexity). *For every inverted list L_i, let H_i be the lower convex hull of the set of 2-D points $\{(j, f_i(L_i[j]))\}_{0\le j\le|L_i|}$ represented by a set of indices $\mathsf{H}_i = \{j_1, \ldots, j_n\}$ where for each $1 \le k \le n$, $(j_k, f_i(L_i[j_k]))$ is a vertex of the convex hull. The list L_i is near-convex if for every k, $j_{k+1} - j_k$ is upper-bounded by some constant c. A database $\mathcal{D}$ is near-convex if every inverted list L_i is near-convex with the same constant c, which we refer to as the convexity constant.*

▶ **Example 15.** Intuitively, the near-convexity assumption captures the case where each $f_i(L_i)$ is decreasing with *approximately* decelerating speed, so the number of points between two convex hull vertices should be small. For example, when f_i is a linear function, the list L_i shown in Figure 2(b) is near-convex with convexity constant 2 since there is at most 1 point between each pair of consecutive vertices of the convex hull (dotted line). In the ideal case shown in Figure 2(a), the constant is 1 when the decrease between successive values is strictly decelerating.

[7] We denote by $f_i(L_i)$ the list $[f_i(L_i[0]), f_i(L_i[1]), \ldots]$ for every L_i.

Imitating the max-reduction strategy, for every pair of consecutive indices j_k, j_{k+1} in H_i and for every index $j \in [j_k, j_{k+1})$, let $\tilde{\Delta}_i[j] = \dfrac{f_i(L_i[j_k]) - f_i(L_i[j_{k+1}])}{j_{k+1} - j_k}$. Since the $(j_k, f_i(L_i[j_k]))$'s are vertices of a lower convex hull, each sequence $\tilde{\Delta}_i$ is non-decreasing. Then the *hull-based* traversal strategy is simply defined as

$$\mathcal{T}_{\mathsf{HL}}(\mathbf{b}) = \operatorname*{argmax}_{1 \le i \le d} (\tilde{\Delta}_i[b_i]). \tag{4}$$

Remark on data structures. In a practical implementation, to answer queries with scoring function F using the hull-based strategy, the lower convex hulls need to be ready before the traversal starts. If F is a general function unknown a priori, the convex hulls need to be computed online which is not practical. Fortunately, when F is the inner product $F(\mathbf{s}) = \mathbf{q} \cdot \mathbf{s}$ parameterized by the query $\mathbf{q}$, each convex hull H_i is exactly the convex hull for the points $\{(j, L_i[j])\}_{0 \le i \le |L_i|}$ from L_i. This is because the slope from any two points $(j, f_i(L_i[j]))$ and $(k, f_i(L_i[k]))$ is $\dfrac{q_i L_i[j] - q_i L_i[k]}{j - k}$, which is exactly the slope from $(j, L_i[j])$ and $(k, L_i[k])$ multiplied by q_i. So by using the standard convex hull algorithm [14], H_i can be pre-computed in $\mathcal{O}(|L_i|)$ time. Then the set of the convex hull vertices H_i can be stored as inverted lists and accessed for computing the $\tilde{\Delta}_i$'s during query processing. In the ideal case, H_i can be as large as $|L_i|$ but is much smaller in practice.

Moreover, during the traversal using the strategy $\mathcal{T}_{\mathsf{HL}}$, choosing the maximum $\tilde{\Delta}_i[b_i]$ at each step can be done in $\mathcal{O}(\log d)$ time using a max heap. This satisfies the requirement that the traversal strategy is efficiently computable.

Near-optimality results. We show that the hull-based strategy $\mathcal{T}_{\mathsf{HL}}$ is near-optimal under the near-convexity assumption.

▶ **Theorem 16.** *Given a decomposable function F, for every near-convex database $\mathcal{D}$ and every threshold θ, the access cost of $\mathcal{T}_{\mathsf{HL}}$ is strictly less than $\mathsf{OPT} + c$ where c is the convexity constant.*

When the assumption holds with a small convexity constant, this near-optimality result provides a much tighter bound compared to the $d \cdot \mathsf{OPT}$ bound in the TA-inspired baseline. This is achieved under data assumption and by keeping the convex hulls as auxiliary data structure, so it does not contradict the lower bound results on the approximation ratio [16].

Proof. Let $\mathcal{B} = \{\mathbf{b}_i\}_{i \ge 0}$ be the sequence of position vectors generated by $\mathcal{T}_{\mathsf{HL}}$. We call a position vector $\mathbf{b}$ a *boundary position* if every b_i is the index of a vertex of the convex hull H_i. Namely, $b_i \in \mathsf{H}_i$ for every $i \in [d]$. Notice that if we break ties consistently during the traversal of $\mathcal{T}_{\mathsf{HL}}$, then in between every pair of consecutive boundary positions $\mathbf{b}$ and $\mathbf{b}'$ in $\mathcal{B}$, $\mathcal{T}_{\mathsf{HL}}(\mathbf{b})$ will always be the same index. We call the subsequence positions $\{\mathbf{b}_i\}_{l \le i < r}$ of $\mathcal{B}$ where $\mathbf{b}_l = \mathbf{b}$ and $\mathbf{b}_r = \mathbf{b}'$ a *segment* with boundaries $(\mathbf{b}_l, \mathbf{b}_r)$. We show the following lemma.

▶ **Lemma 17.** *For every boundary position vector $\mathbf{b}$ generated by $\mathcal{T}_{\mathsf{HL}}$, we have $F(L[\mathbf{b}]) \le F(L[\mathbf{b}^*])$ for every position vector $\mathbf{b}^*$ where $|\mathbf{b}^*| = |\mathbf{b}|$.*

Intuitively, the above lemma says that if the traversal of $\mathcal{T}_{\mathsf{HL}}$ reaches a boundary position $\mathbf{b}$, then the score $F(L[\mathbf{b}])$ is the minimal possible score obtained by any traversal sequence of at most $|\mathbf{b}|$ steps. We prove Lemma 17 by generalizing the greedy argument in the proof of Theorem 14. More details can be found in Appendix G of the full version.

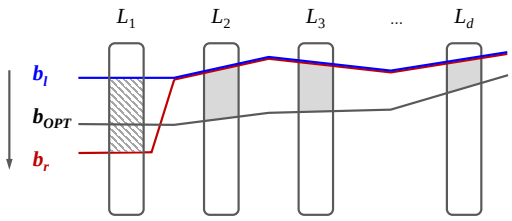


Figure 4 ($\mathbf{b}_l$, $\mathbf{b}_r$): the two boundary positions surrounding the stopping position $\mathbf{b}_{\mathsf{stop}}$ of $\mathcal{T}_{\mathsf{HL}}$; $\mathbf{b}_{\mathsf{OPT}}$: the optimal stopping position; It is guaranteed that (1) $|\mathbf{b}_{\mathsf{stop}}| - |\mathbf{b}_l| < |\mathbf{b}_r| - |\mathbf{b}_l| \leq c$ and (2) $|\mathbf{b}_l| < |\mathbf{b}_{\mathsf{OPT}}|$.

Lemma 17 is sufficient for Theorem 16 because of the following. Suppose $\mathbf{b}_{\mathsf{stop}}$ is the stopping position in $\mathcal{B}$, which means that $\mathbf{b}_{\mathsf{stop}}$ is the first position in $\mathcal{B}$ that satisfies φ_F and the access cost is $|\mathbf{b}_{\mathsf{stop}}|$. Let $\{\mathbf{b}_i\}_{l \leq i < r}$ be the segment that contains $\mathbf{b}_{\mathsf{stop}}$. Given Lemma 17, Theorem 16 holds trivially if $\mathbf{b}_{\mathsf{stop}} = \mathbf{b}_l$. It remains to consider the case $\mathbf{b}_{\mathsf{stop}} \neq \mathbf{b}_l$. Since the traversal does not stop at $\mathbf{b}_l$, we have $F(L[\mathbf{b}_l]) \geq \theta$. By Lemma 17, $\mathbf{b}_l$ is the position with minimal $F(L[\cdot])$ obtained in $|\mathbf{b}_l|$ steps so $|\mathbf{b}_l| \leq \mathsf{OPT}$. Since $|\mathbf{b}_{\mathsf{stop}}| - |\mathbf{b}_l| < |\mathbf{b}_r| - |\mathbf{b}_l| \leq c$, we have that $|\mathbf{b}_{\mathsf{stop}}| < \mathsf{OPT} + c$. We illustrate this in Figure 4. ◀

Since the baseline stopping condition φ_{BL} is tight and complete for inner product queries, one immediate implication of Theorem 16 is that

▶ **Corollary 18** (Informal). *The hull-based strategy $\mathcal{T}_{\mathsf{HL}}$ for inner product queries is near-optimal.*

Verifying the assumption. We demonstrate the practical impact of the near-optimality result in real mass spectrometry datasets. The same experiment is repeated on a document and an image dataset (see Appendix I of the full version). The near-convexity assumption requires that the gap between any two consecutive convex hull vertices has bounded size, which is hard to achieve in general. According to the proof of Theorem 16, for a given query, the difference from the optimal access cost is at most the size of the gap between the two consecutive convex hull vertices containing the last move of the strategy (the $\mathbf{b}_l$ and $\mathbf{b}_r$ in Figure 4). The size of this gap can be much smaller than the global convexity constant c, so the overall precision can be much better in practice. We verify this by running a set of 1,000 real queries on the dataset[8]. The gap size is 163.04 in average, which takes only 1.3% of the overall access cost of traversing the indices. This indicates that the near-optimality guarantee holds in the mass spectrometry dataset. Similar results are obtained in a document and an image dataset, where the gap size takes only 7.9% and 0.4% of the overall access cost respectively.

4.4 The traversal strategy for cosine

Next, we consider traversal strategies which take into account the unit vector constraint posed by the cosine function, which means that the tight and complete stopping condition is φ_{TC} introduced in Section 3. However, since the scoring function MS in φ_{TC} is not decomposable, the hull-based technique cannot be directly applied. We adapt the technique by approximating the original MS with a decomposable function $\tilde{F}$. Without changing the

[8] `https://proteomics2.ucsd.edu/ProteoSAFe/index.jsp`

stopping condition φ_{TC}, the hull-based strategy can then be applied with the convex hull indices constructed with the approximation $\tilde{F}$. In the rest of this section, we first generalize the result in Theorem 16 to scoring functions having decomposable approximations and show how the hull-based traversal strategy can be adapted. Next, we show a natural choice of the approximation for MS with practically tight near-optimal bounds. Finally, we discuss data structures to support fast query processing using the traversal strategy.

We start with some additional definitions.

▶ **Definition 19.** *A d-dimensional function F is decomposably approximable if there exists a decomposable function $\tilde{F}$, called the* decomposable approximation *of F, and two non-negative constants ϵ_1 and ϵ_2 such that $\tilde{F}(\mathbf{s}) - F(\mathbf{s}) \in [-\epsilon_1, \epsilon_2]$ for every vector $\mathbf{s}$.*

When applied to a decomposably approximable function F, the hull-based traversal strategy $\mathcal{T}_{\mathsf{HL}}$ is adapted by constructing the convex hull indices and the $\{\tilde{\Delta}_i\}_{1\le i\le d}$ using the approximation $\tilde{F}$. The following can be obtained by generalizing Theorem 16:

▶ **Theorem 20.** *Given a function F approximable by a decomposable function $\tilde{F}$ with constants (ϵ_1, ϵ_2), for every near-convex database $\mathcal{D}$ wrt $\tilde{F}$ and every threshold θ, the access cost of $\mathcal{T}_{\mathsf{HL}}$ is strictly less than $\mathsf{OPT}(\theta - \epsilon_1 - \epsilon_2) + c$ where c is the convexity constant.*

Proof. Recall that $\mathbf{b}_l$ is the last boundary position generated by $\mathcal{T}_{\mathsf{HL}}$ that does not satisfy the tight stopping condition for F (which is φ_{TC} when F is MS) so $F(L[\mathbf{b}_l]) \ge \theta$. It is sufficient to show that for every vector $\mathbf{b}^*$ where $|\mathbf{b}^*| = |\mathbf{b}_l|$, $F(L[\mathbf{b}^*]) \ge \theta - \epsilon_1 - \epsilon_2$ so no traversal can stop within $|\mathbf{b}_l|$ steps, implying that the final access cost is no more than $|\mathbf{b}_l| + c$ which is bounded by $\mathsf{OPT}(\theta - \epsilon_1 - \epsilon_2) + c$.

By Lemma 17, we know that for every such $\mathbf{b}^*$, $\tilde{F}(L[\mathbf{b}^*]) \ge \tilde{F}(L[\mathbf{b}_l])$. By definition of the approximation $\tilde{F}$, we know that $F(L[\mathbf{b}^*]) \ge \tilde{F}(L[\mathbf{b}^*]) - \epsilon_1$ and $\tilde{F}(L[\mathbf{b}_l]) \ge F(L[\mathbf{b}_l]) - \epsilon_2$. Combined together, for every $\mathbf{b}^*$ where $|\mathbf{b}^*| = |\mathbf{b}_l|$, we have

$$F(L[\mathbf{b}^*]) \ge \tilde{F}(L[\mathbf{b}^*]) - \epsilon_1 \ge \tilde{F}(L[\mathbf{b}_l]) - \epsilon_1 \ge F(L[\mathbf{b}_l]) - \epsilon_1 - \epsilon_2 \ge \theta - \epsilon_1 - \epsilon_2.$$

This completes the proof of Theorem 20. ◀

Choosing the decomposable approximation. By Theorem 20, it is important to choose an approximation $\tilde{F}$ of MS with small ϵ_1 and ϵ_2 for a tight near-optimality result. By inspecting the formula (2) of MS, one reasonable choice of $\tilde{F}$ can be obtained by replacing the term τ with a fixed constant $\tilde{\tau}$. Formally, let

$$\tilde{F}(L[\mathbf{b}]) = \sum_{i=1}^{d} \min\{q_i \cdot \tilde{\tau}, L_i[b_i]\} \cdot q_i \tag{5}$$

be the decomposable approximation of MS where each component is a non-decreasing function $f_i(x) = \min\{q_i \cdot \tilde{\tau}, x\} \cdot q_i$ for $i \in [d]$.

Ideally, the approximation is tight if the constant $\tilde{\tau}$ is close to the final value of τ which is unknown in advance. We argue that when $\tilde{\tau}$ is properly chosen, the approximation parameter $\epsilon_1 + \epsilon_2$ is very small. With a detailed analysis in Appendix H, we obtain the following upper bound of ϵ:

$$\epsilon \le \max\{0, \tilde{\tau} - 1/\mathsf{MS}(L[\mathbf{b}_l])\} + \mathsf{MS}(L[\mathbf{b}_l]) - \tilde{F}(L[\mathbf{b}_l]). \tag{6}$$

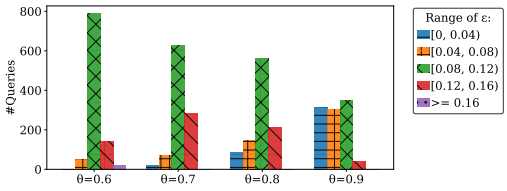


Figure 5 The distribution of ϵ.

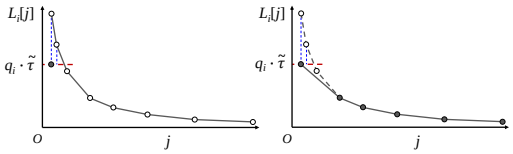


Figure 6 The construction of convex hull $\tilde{\mathsf{H}}_i$.

Verifying the near-optimality. Next, we verify that the above upper bound of ϵ is small in practice. We ran the same set of queries as in Section 4.3 and show the distribution of ϵ's upper bounds in Figure 5. We set $\tilde{\tau} = 1/\theta$ for all queries so the first term of (6) becomes zero. Note that more aggressive pruning can yield better ϵ, but it is not done here for simplicity. Overall, the fraction of queries with an upper bound <0.12 (the sum of the first 3 bars for all θ) is 82.5% and the fraction of queries with $\epsilon > 0.16$ is 0.5%. Similar to the case with inner product queries, the average of the convexity constant c is 193.39, which is only 4.8% of the overall access cost.

Remark on data structures. Similar to the inner product case, it is necessary that the convex hulls for $\mathcal{T}_{\mathsf{HL}}$ can be efficiently obtained without a full computation when a query comes in. For every $i \in [d]$, we let $\tilde{\mathsf{H}}_i$ be the convex hull for the i-th component f_i of $\tilde{F}$ and H_i be the convex hull constructed directly from the original inverted list L_i. Next, we show that each $\tilde{\mathsf{H}}_i$ can be efficiently obtained from H_i during query time so we only need to pre-compute the H_i's.

We observe that when $L_i[b_i] \geq q_i \cdot \tilde{\tau}$, $f_i(L_i[b_i])$ equals a fixed value $q_i^2 \cdot \tilde{\tau}$ otherwise is proportional to $L_i[b_i]$. As illustrated in Figure 6 (left), the list of values $\{f_i(L_i[j])\}_{j \geq 0}$ is essentially obtained by replacing the $L_i[j]$'s greater than $q_i \cdot \tilde{\tau}$ with $q_i \cdot \tilde{\tau}$.

The following can be shown using properties of convex hulls:

▶ **Lemma 21.** *For every $i \in [d]$, the convex hull $\tilde{\mathsf{H}}_i$ is a subset of H_i where an index j_k of H_i is in $\tilde{\mathsf{H}}_i$ iff $k = 1$ or*

$$\big(q_i \cdot \tilde{\tau} - L_i[j_k]\big) \;/\; j_k \geq \big(L_i[j_k] - L_i[j_{k+1}]\big) \;/\; (j_{k+1} - j_k). \tag{7}$$

Lemma 21 provides an efficient way to obtain each convex hull $\tilde{\mathsf{H}}_i$ from the pre-computed H_i's. When a query $\mathbf{q}$ is given, we perform a binary search on each H_i to find the first $j_k \in \mathsf{H}_i$ that satisfies (7). Then $\tilde{\mathsf{H}}_i$ is the set of indices $\{0, j_k, j_{k+1} \dots\}$. We illustrate the construction in Figure 6 (right).

Suppose that the maximum size of all H_i is h. The computation of the $\tilde{\mathsf{H}}_i$'s adds an extra $\mathcal{O}(d \log h)$ of overhead to the query processing time, which is insignificant in practice since h is likely to be much smaller than the size of the database.

5 Related work

In this section, we present the main related work and defer the additional related work (e.g., dimensionality reduction, mass spectrometry search, inner product queries) to the full version.

5.1 Cosine similarity search

The cosine threshold querying studied in this work is a special case of the *cosine similarity search* (CSS) problem [7, 3, 4] mentioned in Section 1. We first survey the techniques developed for CSS.

LSH. A widely used technique for cosine similarity search is locality-sensitive hash (LSH) [27, 5, 20, 22, 30]. The main idea of LSH is to partition the whole database into buckets using a series of hash functions such that similar vectors have high probability to be in the same bucket. However, LSH is designed for *approximate* query processing, meaning that it is not guaranteed to return all the true results. In contrast, this work focuses on exact query processing which returns all the results.

TA-family algorithms. Another technique for cosine similarity search is the family of TA-like algorithms. Those algorithms were originally designed for processing top-k ranking queries that find the top k objects ranked according to an aggregation function (see [21] for a survey). We have summarized the classic TA algorithm [16], presented a baseline algorithm inspired by it, and explained its shortcomings in Section 1. The Gathering-Verification framework introduced in Section 2 captures the typical structure of the TA-family when applied to our setting.

The variants of TA (e.g., [18, 6, 15, 12]) can have poor or no performance guarantee for cosine threshold queries since they do not fully leverage the data skewness and the unit vector condition. For example, Güntzer et al. developed Quick-Combine [18]. Instead of accessing all the lists in a lockstep strategy, it relies on a heuristic traversal strategy to access the list with the highest rate of changes to the ranking function in a fixed number of steps ahead. It was shown in [17] that the algorithm is not instance optimal. Although the hull-based traversal strategy proposed in this paper roughly follows the same idea, the number of steps to look ahead is variable and determined by the next convex hull vertex. Thus, for decomposable functions, the hull-based strategy makes globally optimal decisions and is near-optimal under the near-convexity assumption, while Quick-Combine has no performance guarantee because of the fixed step size even when the data is near-convex.

COORD. Teflioudi et al. proposed the COORD algorithm based on inverted lists for CSS [32, 31]. The main idea is to scan the whole lists but with an optimization to prune irrelevant entries using upper/lower bounds of the cosine similarity with the query. Thus, instead of traversing the whole lists starting from the top, it scans only those entries within a feasible range. We can also apply such a pruning strategy to the Gathering-Verification framework by starting the gathering phase at the top of the feasible range. However, there is no optimality guarantee of the algorithm. Also the optimization only works for high thresholds (e.g., 0.95), which are not always the requirement. For example, a common and well-accepted threshold in mass spectrometry search is 0.6, which is a medium-sized threshold, making the effect of the pruning negligible.

Partial verification. Anastasiu and Karypis proposed a technique for fast verification of θ-similarity between two vectors [3] without a full scan of the two vectors. We can apply the same optimization to the verification phase of the Gathering-Verification framework. Additionally, we prove that it has a novel near-constant performance guarantee in the presence of data skewness.

Other variants. There are several studies focusing on cosine similarity join to find out all pairs of vectors from the database such that their similarity exceeds a given threshold [7, 3, 4]. However, this work is different since the focus is comparing to a given query vector $\mathbf{q}$ rather than join. As a result, the techniques in [7, 3, 4] are not directly applicable: (1) The inverted index is built online instead of offline, meaning that at least one full scan of the whole data

is required, which is inefficient for search. (2) The index in [7, 3, 4] is built for a fixed query threshold, meaning that the index cannot be used for answering arbitrary query thresholds as concerned in this work. The theoretical aspects of similarity join were discussed recently in [2, 20].

5.2 Euclidean distance threshold queries

The cosine threshold queries can also be answered by techniques for distance threshold queries (the threshold variant of nearest neighbor search) in Euclidean space. This is because there is a one-to-one mapping between the cosine similarity θ and the Euclidean distance r for unit vectors, i.e., $r = 2\sin(\arccos(\theta)/2)$. Thus, finding vectors that are θ-similar to a query vector is equivalent to finding the vectors whose Euclidean distance is within r. Next, we review exact approaches for distance queries while leaving the discussion of approximate approaches in the full version.

Tree-based indexing. Several tree-based indexing techniques (such as R-tree, KD-tree, Cover-tree [8]) were developed for range queries (so they can also be applied to distance queries), see [9] for a survey. However, they are not scalable to high dimensions (say thousands of dimensions as studied in this work) due to the well known dimensionality curse issue [34].

Pivot-based indexing. The main idea is to pre-compute the distances between data vectors and a set of selected pivot vectors. Then during query processing, use triangle inequalities to prune irrelevant vectors [13, 19]. However, it does not scale in high-dimensional space as shown in [13] since it requires a large space to store the pre-computed distances.

Clustering-based (or partitioning-based) methods. The main idea of clustering is to partition the database vectors into smaller clusters of vectors during indexing. Then during query processing, irrelevant clusters are pruned via the triangle inequality [29, 28]. Clustering is an optimization orthogonal to the proposed techniques, as they can be used to process vectors within each cluster to speed up the overall performance.

6 Conclusion

In this work, we proposed optimizations to the index-based, TA-like algorithms for answering the cosine threshold queries, which lie at the core of numerous applications. The novel techniques include a complete and tight stopping condition computable incrementally in $\mathcal{O}(\log d)$ time and a family of convex hull-based traversal strategies with near-optimality guarantees for a larger class of decomposable functions beyond cosine. With these techniques, we show near-optimality first for inner-product threshold queries, then extend the result to the full cosine threshold queries using approximation. These results are significant improvements over a baseline approach inspired by the classic TA algorithm. In addition, we have verified with experiments on real data the assumptions required by the near-optimality results.

References

1 Ruedi Aebersold and Matthias Mann. Mass-spectrometric exploration of proteome structure and function. *Nature*, 537:347–355, 2016.

2 Thomas Dybdahl Ahle, Rasmus Pagh, Ilya Razenshteyn, and Francesco Silvestri. On the Complexity of Inner Product Similarity Join. In *PODS*, pages 151–164, 2016.

3 David C. Anastasiu and George Karypis. L2AP: Fast cosine similarity search with prefix L-2 norm bounds. In *ICDE*, pages 784–795, 2014.

4 David C. Anastasiu and George Karypis. PL2AP: Fast parallel cosine similarity search. In *IA3*, pages 8:1–8:8, 2015.

5 Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and Optimal LSH for Angular Distance. In *NIPS*, pages 1225–1233, 2015.

6 Holger Bast, Debapriyo Majumdar, Ralf Schenkel, Martin Theobald, and Gerhard Weikum. IO-Top-k: Index-access Optimized Top-k Query Processing. In *VLDB*, pages 475–486, 2006.

7 Roberto J. Bayardo, Yiming Ma, and Ramakrishnan Srikant. Scaling Up All Pairs Similarity Search. In *WWW*, pages 131–140, 2007.

8 Alina Beygelzimer, Sham Kakade, and John Langford. Cover Trees for Nearest Neighbor. In *ICML*, pages 97–104, 2006.

9 Christian Böhm, Stefan Berchtold, and Daniel A. Keim. Searching in High-dimensional Spaces: Index Structures for Improving the Performance of Multimedia Databases. *CSUR*, 33(3):322–373, 2001.

10 Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

11 Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. Efficient Query Evaluation Using a Two-level Retrieval Process. In *CIKM*, pages 426–434, 2003.

12 Nicolas Bruno, Luis Gravano, and Amélie Marian. Evaluating top-k queries over Web-accessible databases. In *ICDE*, pages 369–380, 2002.

13 Lu Chen, Yunjun Gao, Baihua Zheng, Christian S. Jensen, Hanyu Yang, and Keyu Yang. Pivot-based Metric Indexing. *PVLDB*, 10(10):1058–1069, 2017.

14 Mark De Berg, Otfried Cheong, Marc Van Kreveld, and Mark Overmars. *Computational Geometry: Introduction*. Springer, 2008.

15 Prasad M Deshpande, Deepak P, and Krishna Kummamuru. Efficient Online top-K Retrieval with Arbitrary Similarity Measures. In *EDBT*, pages 356–367, 2008.

16 Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal Aggregation Algorithms for Middleware. In *PODS*, pages 102–113, 2001.

17 Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *JCSS*, 66(4):614–656, 2003.

18 Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kiebling. Optimizing Multi-Feature Queries for Image Databases. In *VLDB*, pages 419–428, 2000.

19 Vagelis Hristidis, Nick Koudas, and Yannis Papakonstantinou. PREFER: A system for the efficient execution of multi-parametric ranked queries. In *SIGMOD*, pages 259–270, 2001.

20 Xiao Hu, Yufei Tao, and Ke Yi. Output-optimal Parallel Algorithms for Similarity Joins. In *PODS*, pages 79–90, 2017.

21 Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A Survey of Top-k Query Processing Techniques in Relational Database Systems. *CSUR*, 40(4):1–58, 2008.

22 Piotr Indyk and Rajeev Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *ICDT*, pages 604–613, 1998.

23 Harold W Kuhn and Albert W Tucker. Nonlinear programming. In *Traces and Emergence of Nonlinear Programming*, pages 247–258. Springer, 2014.

24 B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*, pages 2130–2137, 2009.

25 Henry Lam, Eric W. Deutsch, James S. Eddes, Jimmy K. Eng, Nichole King, Stephen E. Stein, and Ruedi Aebersold. Development and validation of a spectral library searching method for peptide identification from MS/MS. *Proteomics*, 7(5), 2007.

26 Hui Li, Tsz Nam Chan, Man Lung Yiu, and Nikos Mamoulis. FEXIPRO: Fast and Exact Inner Product Retrieval in Recommender Systems. In *SIGMOD*, pages 835–850, 2017.

27 Anand Rajaraman and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2011.

28 Sharadh Ramaswamy and Kenneth Rose. Adaptive Cluster Distance Bounding for High-Dimensional Indexing. *TKDE*, 23(6):815–830, 2011.

29 Hanan Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers Inc., 2005.

30 Yufei Tao, Ke Yi, Cheng Sheng, and Panos Kalnis. Quality and Efficiency in High Dimensional Nearest Neighbor Search. In *SIGMOD*, pages 563–576, 2009.

31 Christina Teflioudi and Rainer Gemulla. Exact and Approximate Maximum Inner Product Search with LEMP. *TODS*, 42(1):5:1–5:49, 2016.

32 Christina Teflioudi, Rainer Gemulla, and Olga Mykytiuk. LEMP: Fast Retrieval of Large Entries in a Matrix Product. In *SIGMOD*, pages 107–122, 2015.

33 Mingxun Wang and Nuno Bandeira. Spectral Library Generating Function for Assessing Spectrum-Spectrum Match Significance. *Journal of Proteome Research*, 12(9):3944–3951, 2013.

34 Roger Weber, Hans-Jörg Schek, and Stephen Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *VLDB*, pages 194–205, 1998.

An Experimental Study of the Treewidth of Real-World Graph Data

Silviu Maniu
LRI, CNRS, Université Paris-Sud, Université Paris-Saclay, Orsay, France
silviu.maniu@lri.fr

Pierre Senellart
DI ENS, ENS, CNRS, PSL University, Paris, France
Inria Paris, France
LTCI, Télécom ParisTech, Paris, France
pierre@senellart.com

Suraj Jog
University of Illinois at Urbana–Champaign, Urbana-Champaign, USA
sjog2@illinois.edu

Abstract

Treewidth is a parameter that measures how tree-like a relational instance is, and whether it can reasonably be decomposed into a tree. Many computation tasks are known to be tractable on databases of small treewidth, but computing the treewidth of a given instance is intractable. This article is the first large-scale experimental study of treewidth and tree decompositions of real-world database instances (25 datasets from 8 different domains, with sizes ranging from a few thousand to a few million vertices). The goal is to determine which data, if any, can benefit of the wealth of algorithms for databases of small treewidth. For each dataset, we obtain upper and lower bound estimations of their treewidth, and study the properties of their tree decompositions. We show in particular that, even when treewidth is high, using partial tree decompositions can result in data structures that can assist algorithms.

2012 ACM Subject Classification Theory of computation → Logic and databases

Keywords and phrases Treewidth, Graph decompositions, Experiments, Query processing

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.12

Related Version An extended version of this paper is available at `http://arxiv.org/abs/1901.06862`.

Acknowledgements We are grateful to Antoine Amarilli and Mikaël Monet for feedback on parts of this paper.

1 Introduction and Related Work

A number of data management tasks related to query evaluation are computationally intractable when rich query languages or complex tasks are involved, even when the query is assumed to be fixed (that is, when we consider *data complexity* [60]). For example:

- query evaluation of Boolean monadic second-order (MSO) queries is hard for every level of the polynomial hierarchy [4];
- unless P = NP, there is no polynomial-time enumeration or counting algorithm for first-order (FO) queries with free second-order variables [57, 29];
- computing the probability of conjunctive queries (CQs) over tuple-independent databases, a very simple model of probabilistic databases, is #P-hard [25];

- unless P = NP, there is no polynomial-time algorithm to construct a deterministic decomposable negation normal form (d-DNNF) representation of the Boolean provenance of some CQ [25, 40]; furthermore, there is no polynomial bound on the size of a *structured* d-DNNF representation of the Boolean provenance of unions of conjunctive queries with disequalities [11, Theorem 33].

Other problems yield complexity classes usually considered tractable, such as AC_0 for Boolean FO query evaluation [1], but may still result in impractical running times on large database instances.

To face this intractability and practical inefficiency, one possible approach has been to determine conditions on the structure of databases that ensure tractability, often through a series of *algorithmic meta-theorems* [44]. This has led, for instance, to the introduction of the notions of *locally tree-decomposable structures* for near-linear-time evaluation of Boolean FO queries [33], or to that of *structures of bounded expansion* for constant-delay enumeration of FO queries [41].

Treewidth. A particularly simple and widely used way to restrict database instances that ensures a wide class of tractability results is to bound the *treewidth* of the instance (this is actually a special case of both notions of locally tree-decomposable and bounded expansion). *Treewidth* [56] is a graph-theoretic parameter that characterizes how tree-like a graph, or more generally a relational instance, is, and hence whether it can be reasonably transformed into a tree structure (a *tree decomposition*). Indeed:

- query evaluation of MSO queries is linear-time over bounded-treewidth structures [24, 32];
- counting [13] and enumeration [14, 7] of MSO queries on bounded-treewidth structures is linear-time;
- computing the probability of MSO queries over a bounded-treewidth tuple-independent database is linear-time assuming constant-time rational arithmetic [8];
- a linear-sized structured d-DNNF representation of the provenance of any MSO query over bounded-treewidth databases can be computed in linear-time [9, 11].

These results mostly stem from the fact that, on trees, MSO queries can be rewritten to tree automata [58], though this process is non-elementary in general (which impacts the *combined complexity* [60], but not the data complexity). We can see these results as *fixed-parameter tractability*, with a complexity in $\mathcal{O}(f(|Q|, k) \times |D|)$ where $|D|$ is the size of the database, k its treewidth, $|Q|$ the size of the query, and f some computable function. Note that another approach for tractability, out of the scope of this paper, is to restrict the queries instead of the instances, e.g., by enforcing low treewidth on the queries [37] or on the provenance of the queries [39].

Such results have been, so far, of mostly theoretical interest – mainly due to the high complexity of the function f of $|Q|$ and k. However, algorithms that exploit the low treewidth of instances have been proposed and successfully applied to real-world and synthetic data: for shortest path queries in graphs [62, 54], distance queries in probabilistic graphs [47], or ad-hoc queries compiled to tree automata [50]. In other domains, low treewidth is an indicator for efficient evaluation of quantified Boolean formulas [55].

Sometimes, treewidth even seems to be the *sole* criterion that may render an intractable problem tractable, under some technical assumptions: [45] shows that, unless the exponential-time hypothesis is false, MSO_2 query evaluation is intractable over *subinstance-closed* families of instances of treewidth *strongly unbounded poly-logarithmically*; [34, 9] show that MSO query evaluation is intractable over *subinstance-closed* families of instances of treewidth that are *densely unbounded poly-logarithmically* (a weaker notion); [9] shows that counting MSO query

results is intractable over *subinstance-closed* families of instances *of unbounded treewidth that are treewidth-constructible* (an even weaker notion, simply requiring large-treewidth instances to be efficiently constructible, see [9, Definition 4.1]); finally, [8, 9] shows that one can exhibit FO queries whose probability evaluation is polynomial-time on structures of bounded treewidth, but #P-hard on *any* treewidth-constructible family of instances of unbounded treewidth.

For this reason, and because of the wide variety of problems that become tractable on bounded-treewidth instances, treewidth is an especially important object of study.

Treewidth of real-world databases. If there is hope for practical applicability of treewidth-based approaches, one needs to answer the following two questions: *Can one efficiently compute the treewidth of real-world databases?* and *What is the treewidth of real-world data?* The latter question is the central problem addressed in this paper.

The answer to the former is that, unfortunately, treewidth cannot reliably be computed efficiently in practice. Indeed, computing the treewidth of a graph is an NP-hard problem [13] and, in practice, exact computation of treewidth is possible only for very small instances, with no more than dozens of vertices [18]. An additional theoretical result is that, given a width w, it is possible to check whether a graph has treewidth w and produce a tree decomposition of the graph in linear time [17]; however, the large constant terms make this procedure impractical. Known exact treewidth computation algorithms [18] may be usable on small graphs, but they are impossible to apply for our purposes. Indeed, in [18], the largest graph for which algorithms finished running had a mere 40 vertices.

A more realistic approach is to compute *estimations* of treewidth, i.e., an interval formed of a *lower bound* and an *upper bound* on the treewidth. Upper bound algorithms (surveyed in [19]) use multiple approaches for estimation, which all output a tree decomposition. One particularly important class of methods for generating tree decompositions relies on *elimination orderings*, that also appear in junction tree algorithms used in belief propagation [46]. For lower bounds (surveyed in [20]), where no decomposition can be obtained, one can use degree-based or minor-based measures on graphs, which themselves act as proxies for treewidth.

Some upper bound and lower bound algorithms have been implemented and experimented with in [59, 19, 12, 20]. However, in all cases these algorithms were evaluated on graphs that are either very small (of the order of dozens of vertices), as in [59], or on slightly larger synthetic graphs (with up to 1 000 vertices) generated with exact treewidth values in mind, as in [19, 20]. The main purpose of these experiments was to evaluate the estimators' performance. Recently, the PACE challenge has had a track dedicated to the estimation of treewidth [26]: exact treewidth on relatively small graphs, upper bounds on treewidth on larger graphs. Local improvements of upper bounds have been also evaluated on small graphs in [31]. Since all these works aim at comparing estimation algorithms, they do not investigate the actual treewidth of real-world data.

Another relevant work is [3], which studied the *core-periphery structure* of social networks, by building tree decompositions via node elimination ordering heuristics, but without establishing any treewidth bounds. In this work, we use the same heuristics to compute bounds on treewidth.

Finally, there have been some work on analyzing properties of real-world *queries*. Queries are usually much smaller than database instances, but it turns out that they are also much simpler in structure: [53] shows that an astounding 99.99% of conjunctive patterns present in a SPARQL query log are acyclic, i.e., of treewidth 1. [22] similarly showed that the

overwhelming majority of graph pattern queries in SPARQL query logs had treewidth 1, less than 0.003% had treewidth 2, and a single one (out of more than 15 million) had treewidth 3. We shall see that the situation is much different with the treewidth of database instances. Note that, in many settings, low-treewidth of queries does not suffice for tractability: in probabilistic databases, for instance, #P-hardness holds even for acyclic queries [25].

Contributions. In this experimental study, our contributions are twofold.

First, using previously studied algorithms for treewidth estimation, we set out to find classes of real-world data that may exhibit relatively low values of treewidth, thus identifying potential cases in which treewidth-based approaches are of practical interest. For this, after formally defining tree decompositions and treewidth (Section 2), we select the algorithms that are able to deal with large-scale data instances, for both lower- and upper-bound estimations (Section 3). Our aim here is not to propose new algorithms for treewidth estimation, and not to exhaustively evaluate existing treewidth estimation algorithms, but rather to identify algorithms that can give acceptable treewidth estimation values in reasonable time, in order to apply them to real-world data. Then, we use these algorithms to obtain lower and upper bound intervals on treewidth for 25 databases from 8 different domains (Section 4). We mostly consider graph data, for which the notion of treewidth was initially designed (the treewidth of an arbitrary relational instance is simply defined as that of its Gaifman graph). The graphs we consider, all obtained from real-world applications, have between several thousands and several millions of vertices. To the best of our knowledge, this is the first comprehensive study of the treewidth of real-world data of large scale from a variety of application domains.

Our finding is that, *generally*, the treewidth is too large to be able to use treewidth-based algorithms directly with any hope of efficiency.

Second, from this finding, we investigate how a *relaxed (or partial) decomposition* can be used on real-world graphs. In short, we no longer look for complete tree decompositions; instead, we allow the graph to be only partially decomposed. In complex networks, there often exists a dense core together with a tree-like fringe structure [52]; it is hence possible to decompose the fringe into a tree, and to place the rest of the graph in a dense "root". It has been shown that this approach can improve the efficiency of some graph algorithms [62, 5, 47]. In Section 5, we analyze its behavior on real-world graphs. We conclude the paper in Section 6 with a discussion of lessons learned, as to which real-world data admit (full or partial) low-treewidth tree decompositions, and how this impacts query evaluation tasks.

Due to lack of space, some details and additional experiments can be found in an extended version of this article [48].

2 Preliminaries on Treewidth

To make the concepts in the following clear, we start by formally introducing the concept of *treewidth*. Following the original definitions in [56], we first define a tree decomposition:

▶ **Definition 1** ((Tree Decomposition)). *Given an* undirected graph $G = (V, E)$*, where* V *represents the set of vertices (or nodes) and* $E \subseteq V \times V$ *the set of edges, a* tree decomposition *is a pair* (T, B) *where* $T = (I, F)$ *is a tree and* $B : I \to 2^V$ *is a labeling of the nodes of* T *by subsets of* V *(called* bags*), with the following properties:*

1. $\bigcup_{i \in I} B(i) = V$*;*
2. $\forall (u, v) \in E$*,* $\exists i \in I$ *s.t.* $\{u, v\} \subseteq B(i)$*; and*

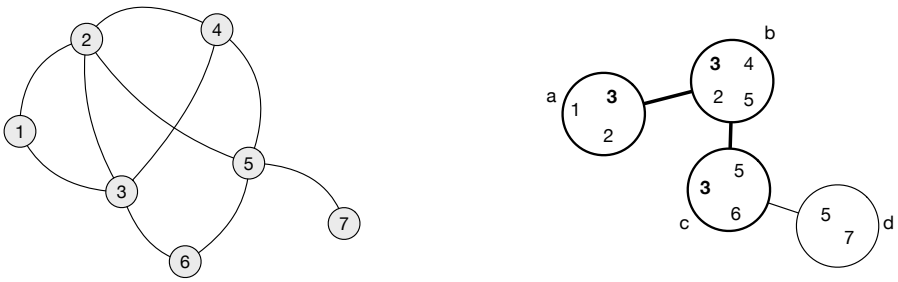


Figure 1 Example undirected, unlabeled, graph (left) and decomposition of width 3 (right).

3. $\forall v \in V$, $\{i \in I \mid v \in B(i)\}$ *induces a subtree of* T.

Intuitively, a tree decomposition groups the vertices of a graph into bags so that they form a tree-like structure, where a link between bags is established when there exists common vertices in both bags.

▶ **Example 2.** Figure 1 illustrates such a decomposition. The resulting decomposition is formed of 4 bags, each containing a subset of the nodes in the graph. The bags containing node 3 (in bold) form a connected subtree of the tree decomposition.

Based on the number of vertices in a bag, we can define the concept of *treewidth*:

▶ **Definition 3** ((Treewidth)). *Given a graph* $G = (V, E)$ *the* width *of a tree decomposition* (T, B) *is equal to* $\max_{i \in I}(|B(i)| - 1)$. *The* treewidth *of* G, $w(G)$, *is equal to the minimal width of all tree decompositions of* G.

It is easy to see that an isolated point has treewidth 0, a tree treewidth 1, a cycle treewidth 2, and a $(k + 1)$-clique (a complete graph of k nodes) treewidth k.

▶ **Example 4.** The width of the decomposition in Figure 1 is 3. This tells us the graph has a treewidth of at most 3. The treewidth of this graph is actually exactly 3: indeed, the 4-clique, which has treewidth 3, is a *minor* of the graph in Figure 1 (it is obtained by removing nodes 1 and 7, and by contracting the edges between 3 and 6 and 5 and 6), and treewidth never increases when taking a minor (see, for instance, [38]).

As previously mentioned, the treewidth of an arbitrary relational instance is defined as that of its *Gaifman graph*, the graph whose vertices are constants of the instances and where there is an edge between two vertices if they co-occur in the same fact. We will therefore implicitly represent relational database instances by their Gaifman graphs in what follows.

We are now ready to present algorithms for lower and upper bounds on treewidth.

3 Treewidth Estimation

The objective of our experimental evaluation is to obtain reasonable estimations of treewidth, using algorithms with reasonable execution time on real-world graphs.

Once we know we do not have the luxury of an exact computation of the treewidth, we are left with estimations of the range of possible treewidths, between a *lower bound* and an *upper bound*. For the purposes of this experimental survey, we restrict ourselves to the most efficient estimation algorithms from the literature. We refer the reader to [19] and [20], respectively, for a more complete survey of treewidth upper and lower bound estimation algorithms on synthetic data.

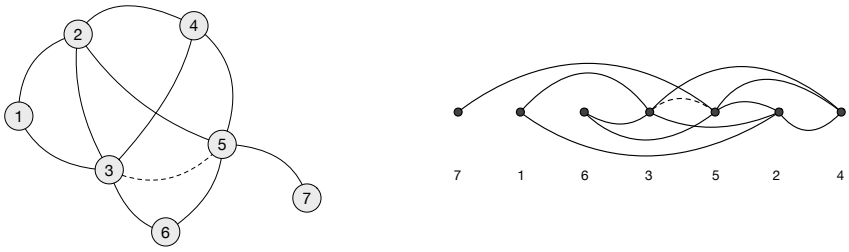


Figure 2 Graph triangulation for the graph of Figure 1 (left) and its elimination ordering (right).

Treewidth Upper Bounds. As we have defined, the treewidth is the smallest width among all possible tree decompositions. In other words, the width of any decomposition of a graph is an upper bound of the actual treewidth of that graph. A treewidth upper bound estimation algorithm can thus be seen as an algorithm to find a decomposition whose width is as close as possible to the treewidth of the graph. To understand how one can do that, we need to introduce the classical concept of *elimination ordering* and to explain its connection to treewidth.

We start by introducing *triangulations* of graphs, which transform a graph G into a graph G^Δ that is *chordal*:

▶ **Definition 5.** *A* chordal *graph is a graph G such that every cycle in G of at least four vertices has a* chord *– an edge between two non-successive vertices in the cycle.*

A triangulation *(or* chordal completion*) of a graph G is a minimal chordal supergraph G^Δ of G: a graph obtained from G by adding a minimal set of edges to obtain a chordal graph.*

▶ **Example 6.** The graph in Figure 1 is not chordal, since, for example, the cycle 3–4–5–6–3 does not have a chord. If one adds an edge between 3 and 5, as in Figure 2 (left), one can verify that the resulting graph is chordal, and thus a triangulation of the graph of Figure 1.

One way to obtain triangulations of graphs is *elimination orderings*. An elimination ordering ω of a graph $G = (V, E)$ of n nodes is an ordering of the vertices of G, i.e., it can be seen as a bijection from V onto $\{1, \ldots, n\}$. From this ordering, one obtains a triangulation by applying sequentially the following *elimination procedure* for each vertex v: first, edges are added between remaining neighbors of v as needed so that they form a clique, then v is *eliminated* (removed) from the graph. For every elimination ordering ω, G along with all edges added to G in the elimination procedure forms a graph, denoted G^Δ_ω. This graph is chordal (indeed, we know that the two neighbors of the first node of any cycle we encounter in the elimination ordering have been connected by a chord by the elimination procedure). It is also a supergraph of G, and it can be shown it is a minimal chordal supergraph, i.e., a triangulation of G.

▶ **Example 7.** Figure 2 (right) shows a possible elimination ordering $(7, 1, 6, 3, 5, 2, 4)$ of the graph of Figure 1. The elimination procedure adds a single edge, when processing node 6, between nodes 3 and 5. The resulting triangulation is the graph on the left of Figure 2.

Elimination orderings are connected to treewidth by the following result:

▶ **Theorem 8.** [19] *Let $G = (V, E)$ a graph, and $k \leqslant n$. The following are equivalent:*

1. *G has treewidth k.*
2. *G has a triangulation G^{Δ}, such that the maximum clique in G^{Δ} has size $k+1$.*
3. *There exists an elimination ordering ω such that the maximum clique size in G^{Δ}_{ω} is $k+1$.*

Obtaining the treewidth of the graph is thus equivalent to finding an optimal elimination ordering. Moreover, constructing a tree decomposition from an elimination ordering is a natural process: each time a vertex is processed, a new bag is created containing the vertex and its neighbors. Note that, in practice, we do not need to compute the full elimination ordering: we can simply stop when we know that the number of remaining vertices is lower that the largest clique found thus far.

▶ **Example 9.** In the triangulation of Figure 2 (left), corresponding to the elimination ordering on the right, the maximum clique has size 4: it is induced by the vertices $2, 3, 4, 5$. This proves the existence of a tree decomposition of width 3. Indeed, it is exactly the tree decomposition in Figure 1 (right): bag `d` is constructed when 7 is eliminated, bag `a` when 1 is eliminated, bag `c` when 6 is eliminated, and finally bag `b` when 3 is eliminated.

Finding a "good" upper bound on the treewidth can thus be done by finding a "good" elimination ordering. This is still an intractable problem, of course, but there are various heuristics for generating elimination orderings leading to good treewidth upper bounds. One important class of such elimination ordering heuristics are the greedy heuristics. Intuitively, the elimination ordering is generated in an incremental manner: each time a new node has to be chosen in the elimination procedure, it is chosen using a criterion based on its neighborhood. In our study, we have implemented the following greedy criteria (with ties broken arbitrarily):

- Degree. The node with the minimum degree is chosen. [49, 16]
- FillIn. The node with the minimum needed "fill-in" (i.e., the minimum number of missing edges for its neighbors to form a clique) is chosen. [19]
- Degree+FillIn. The node with the minimum sum of degree and fill-in is chosen.

▶ **Example 10.** The elimination ordering of Figure 2 (right) is an example of the use of Degree+FillIn. Indeed, 7 is first chosen, with value 1, then 1 with value 2, then 6 with value $2 + 1 = 3$. After that, the order is arbitrary since 2, 3, 4, and 5 form a clique (and thus have initial value 3).

Previous studies [59, 42, 19] have found these greedy criteria give the closest estimations of the real treewidth. An alternative way of generating an elimination ordering is based on maximum cardinality search [42, 19]; however, it is both less precise than the greedy algorithms – due to its reliance on how the first node in the ordering is chosen – and slower to run.

Treewidth Lower Bounds. In contrast to upper bounds, obtaining treewidth lower bounds is not constructive. In other words, lower bounds do not generate decompositions; instead, the estimation of a lower bound is made by computing other measures on a graph, that are a proxy for treewidth. In this study, we implement algorithms using three approaches: subgraph-based bounds, minor-based bounds, and bounds obtained by constructing improved graphs.

Given a graph $G = (V, E)$, let $\delta(G)$ be its lowest degree, and $\delta_2(G) \geqslant \delta(G)$ its second lowest degree (i.e., the degree of the second vertex when ordered by degree). It is known that $\delta_2(G)$ is itself a lower bound on the treewidth [43]. This, however, is too coarse an

estimation, and we need better bounds. We shall use two degeneracy measures of the graphs. The first, the *degeneracy* of G, $\delta D(G)$, is the maximum value of $\delta(H)$ over all subgraphs H of G. Similarly, the δ_2*-degeneracy*, $\delta_2 D(G)$, of a graph is the maximum value of $\delta_2(H)$ over all subgraphs H.

We have the following lemma:

▶ **Lemma 11.** [20] *Let $G = (V, E)$ be a graph, and $W \subseteq V$ be a set of vertices. The treewidth of the subgraph of G induced by W is at most the treewidth of G.*

A corollary of the above lemma is that the values $\delta D(G)$ and $\delta_2 D(G)$ are themselves lower bounds of treewidth:

▶ **Corollary 12.** [20] *For every graph G, the treewidth of G is at least $\delta_2 D(G) \geqslant \delta D(G)$.*

To compute $\delta D(G)$ and $\delta_2 D(G)$ exactly, the following natural algorithms can be used [43, 20]: repeatedly remove a vertex of smallest degree – or smallest except for some fixed node v, respectively – from the graph, and keep the maximum value thus encountered. As in [20], we refer to these algorithms as Mmd (Maximum Minimum Degree) and Delta2D, respectively. Ties are broken arbitrarily.

▶ **Example 13.** Let us apply the Mmd algorithm to the graph of Figure 1 (left). The algorithm may remove, in order, 7 (degree 1), 1 (degree 2), 6 (degree 2), 3 (degree 2), 5 (degree 2), 4 (degree 1), 2 (degree 0). This gives a lower bound of 2 on the treewidth, which is not tight as we saw.

An equivalent of Lemma 11 on treewidth also holds for *minors* of a graph G: if H is a minor of G, then the treewidth of H is at most the treewidth of G [38]. A minor H of a graph G is a graph obtained by allowing, in addition to edge and node deletion as when taking subgraphs, *edge contractions*. Then the concepts of *contraction degeneracy*, $\delta C(G)$, and δ_2*-contraction degeneracy*, $\delta_2 C(G)$, are defined analogously to $\delta D(G)$ and $\delta_2 D(G)$ by considering all minors instead of all subgraphs:

▶ **Lemma 14.** [20] *For every graph G, the treewidth of G is at least $\delta_2 C(G) \geqslant \delta C(G)$.*

Unfortunately, computing $\delta C(G)$ or $\delta_2 C(G)$ is NP-hard [21]; hence, only heuristics can be used. One such heuristic for $\delta C(G)$ is a simple change to the Mmd algorithm, called Mmd+ [21, 36]: at each step, instead of removing a vertex, a neighbor is chosen and the corresponding edge is contracted. Choosing a neighbor node to contract requires some heuristic also; in line with previous studies, in this study we choose the neighbor node that has the least overlap in neighbors – this is called the *least-c* heuristic [63].

Finally, another approach to treewidth lower bounds that we consider are *improved graphs*, an approach that can be used in combination with any of the lower bound estimation algorithms presented so far. Consider a graph G and an integer k and the following operation: while there are non-adjacent vertices v and w that have at least $k+1$ common neighbors, add the edge (v, w) to the improved graph G'. The resulting graph G' is the $(k+1)$-neighbor improved graph of G. Using these improved graphs can lead to a lower bound on treewidth: the $(k+1)$-neighbor improved graph G' of a graph G having at most treewidth k also has treewidth at most k.

To use this property, one can start from an already computed estimation k of a lower bound (by using Mmd, Mmd+, or Delta2D for example) and then repeatedly generate a $(k+1)$-neighbor improved graph, estimate a new lower bound on treewidth, and repeat the process until the graph cannot be improved. This algorithm is known as Lbn in the

literature [23], and can be combined with any other lower bound estimation algorithm. A refinement of LBN, that alternates improvement and contraction steps, LBN+, has also been proposed [21].

▶ **Example 15.** Let us illustrate the use of LBN together with MMD on the graph of Figure 1 (left). As shown in Example 13, a first run of MMD yields $k = 2$. We compute a 3-neighbor improved graph for G by adding an edge between nodes 3 and 5 (that share neighbors 2, 4, 6). Now, running MMD one more time yields the possible sequence 7 (degree 1), 1 (degree 2), 6 (degree 2), 3 (degree 3), 5 (degree 2), 4 (degree 1), 2 (degree 0). We thus obtain a lower bound of 3 on the treewidth, which is this time tight.

4 Estimation Results

We now present our main experimental study, first introducing the 25 datasets we are considering, then upper and lower bound results, running time and estimators, and an aside discussion of the treewidth of synthetic networks. All experiments were made on a server using an 8-core Intel Xeon 1.70GHz CPU, having 32GB of RAM, and using 64bit Debian Linux. All datasets were given at least two weeks to finish, after which the algorithms were stopped and the best lower and upper bounds were recorded.

Datasets. For our study, we have evaluated the treewidth estimation algorithms on 25 datasets from 8 different domains (see Appendix A of [48] for descriptions of how they were obtained): infrastructure networks (road networks, public transportation, power grid), social networks (explicit as in social networking sites, or derived from interaction patterns), web-like networks, a communication network, data with a hierarchical structure (genealogy trees), knowledge bases, traditional OLTP data, as well as a biological interaction network.

Table 1 summarizes the datasets, their size, and the best treewidth estimations we have been able to compute. For reproducibility purposes, all datasets, along with the code that has been used to compute the treewidth estimations, can be freely downloaded from `https://github.com/smaniu/treewidth/`.

Upper Bounds. We show in Figure 3 the results of our estimation algorithms. Lower values mean better treewidth estimations. Focusing on the upper bounds only (red circular points), we notice that, in general, FILLIN does give the smallest upper bound of treewidth, in line with previous findings [20]. Interestingly, the DEGREE heuristic is quite competitive with the other heuristics. This fact, coupled with its lower running time, means that it can be used more reliably in large graphs. Indeed, as can be seen in the figure, on some large graphs only the DEGREE heuristic actually finished at all; this means that, as a general rule, DEGREE seems the best fit for a quick and relatively reliable estimation of treewidth.

We plot both the absolute values of the estimations in Figure 3a, but also their relative values (in Figure 3b, representing the ratio of the estimation over the number of nodes in the graph), to allow for an easier comparison between networks. The absolute value, while interesting, does not yield an intuition on how the bounds can differ between network types. If we look at the relative values of treewidth, it becomes clear that infrastructure networks have a treewidth that is much lower than other networks; in general they seem to be consistently under one thousandth of the original size of the graph. This suggests that, indeed, this type of network may have properties that make them have a lower treewidth. For the other types of networks, the estimations can vary considerably: they can go from one hundredth (e.g., MATH) to one tenth (e.g., WIKITALK) of the size of the graph.

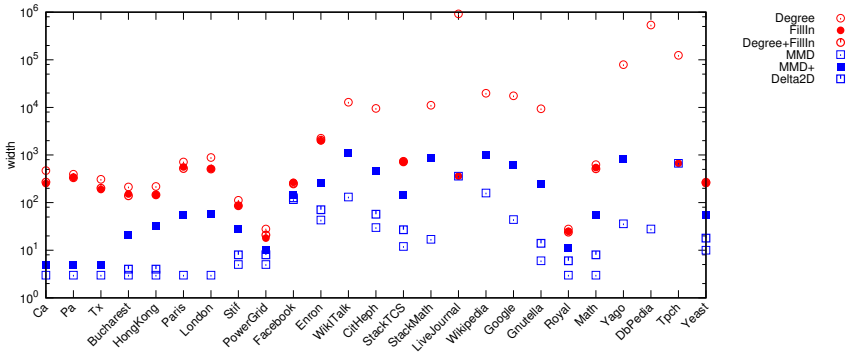
Table 1 Datasets and summary of lower and upper bounds. Bounds with a * are partial (the decomposition process was interrupted before it finished).

type	Dataset name	nodes	edges	Lower width	Upper width
infrastructure	CA	1 965 206	2 766 607	5	252
	PA	1 088 092	1 541 898	5	333
	TX	1 379 917	1 921 660	5	189
	BUCHAREST	189 732	223 143	21	139
	HONGKONG	321 210	409 038	32	145
	PARIS	4 325 486	5 395 531	55	521
	LONDON	2 099 114	2 588 544	57	507
	STIF	17 720	31 799	28	86
	USPOWERGRID	4 941	6 594	10	18
social	FACEBOOK	4 039	88 234	142	247
	ENRON	36 692	183 831	257	1 989
	WIKITALK	2 394 385	4 659 565	1 113	12 843
	CITHEPH	34 546	420 877	469	9 498
	STACK-TCS	25 232	69 026	143	717
	STACK-MATH	1 132 468	2 853 815	850	11 100
	LIVEJOURNAL	3 997 962	34 681 189	360	919 532*
web	WIKIPEDIA	252 335	2 427 434	1 007	19 876
	GOOGLE	875 713	4 322 051	621	17 571
communication	GNUTELLA	65 586	147 892	244	9 374
hierarchy	ROYAL	3 007	4 862	11	24
	MATH	101 898	105 131	56	515
ontology	YAGO	2 635 315	5 216 293	836	79 059*
	DBPEDIA	7 697 211	30 622 392	28	538 805*
database	TPCH	1 381 291	79 352 127	699	124 316*
biology	YEAST	2 284	6 646	54	255

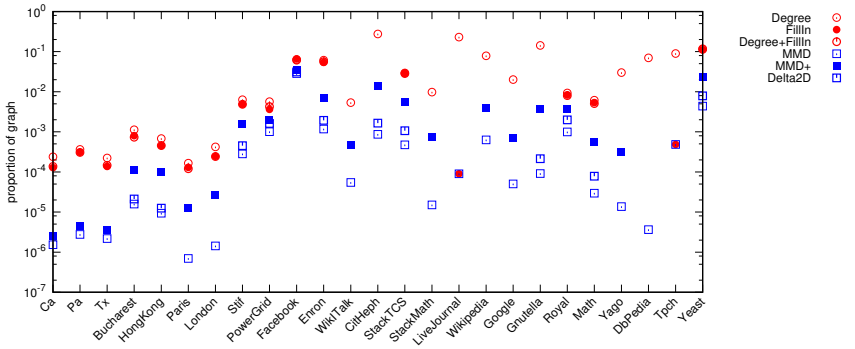
As further explained in Appendix B of [48], the bounds obtained here on infrastructure networks are consistent with a conjectured $\mathcal{O}(\sqrt[3]{n})$ bound on the treewidth of road networks [27]. One relevant property is their *low highway dimension* [2], which helps with routing queries and decomposition into contraction hierarchies. Even more relevant to our results is the fact that they tend to be "almost planar". More specifically, they are k-planar: each edge can allow up to k crossing in their plane embedding. It has been shown in [28] that k-planar graphs have treewidth $\mathcal{O}(\sqrt{(k+1)n})$, a relatively low treewidth that is consistent with our results.

The treewidth of hierarchical networks is surprisingly high, but not for trivial reasons: in both ROYAL and MATH, largest cliques have size 3. More complex structures (cousin marriages, scientific communities) impact the treewidth, along with the fact that treewidth cannot exploit the property that both networks are actually DAGs.

In upper bound algorithms, ties are broken arbitrarily which causes a non-deterministic behavior. We show in Appendix C of [48] that variations due to this non-determinism is of little significance.



(a) absolute values.



(b) relative values.

Figure 3 Treewidth estimation of different algorithms (logarithmic scale).

Lower Bounds. Figure 3 also reports on the lower bound estimations (blue rectangular points). Now, higher values represent a better estimation. The same differentiation between infrastructure networks and the other networks holds in the case of lower bounds – treewidth lower bounds are much lower in comparison with other networks. We observe that the variation between upper and lower bound estimations can be quite large. Generally, we find that degree-based estimators, MMD and DELTA2D, give bounds that are very weak. The contraction-based estimator, MMD+, however, consistently seems to give the best bounds of treewidth, and returns lower bounds that are much larger than the degree-based estimations.

Interestingly, in the case of the networks CA, PA, and TX, the values returned for MMD+ and MMD are always 5 and 3, respectively. This has been remarked on in [21] – there exist instances where heuristics of lower bounds perform poorly, giving very low lower bounds. In our case, this only occurs for some road networks, all coming from the same data source, i.e., the DIMACS challenge on shortest paths.

In Figure 3, we have not plotted the estimations resulting from LBN and LBN+. The reason is that we have found that these algorithms do not generally give any improvement in the estimation of the lower bounds. Specifically, we have found an improvement only in two datasets for the DELTA2D heuristic: for FACEBOOK from 126 originally to 157 for LBN(DELTA2D) and 159 for LBN+(DELTA2D); and for STACK-TCS from 27 originally to 30 for LBN+(DELTA2D). In all cases, however, MMD+ is always competitive by itself.

Running Time. The complexity of different estimation algorithms (see Appendix D of [48]), varies a lot, ranging from quasi-linear for low-treewidth to cubic time. Even if all of the algorithms exhibit polynomial complexity, the cost can become quickly prohibitive, even for

graphs of relatively small sizes; indeed, not all algorithms finished on all datasets within the time bound of two weeks of computation time: indeed, only the fastest algorithms for each bound – Degree and Mmd respectively – finish processing all datasets. In some cases, for upper bounds, even Degree timed out – in this case, we took the value found at the moment of the time-out; this still represents an upper bound. The datasets for which this occurred are LiveJournal, Yago, DbPedia, and Tpch.[1]

Phase Transitions in Synthetic Networks. We have seen that graphs other than the infrastructure networks have treewidth values that are relatively high. An interesting question that these results lead to is: *is it the case for all relatively complex networks, or is it a particularity of the chosen datasets?*

To give a possible answer to this, we have evaluated the treewidth of a range of synthetic graph models and their parameters:

Random. This synthetic network model due to Erdős and Rényi [30] is an entirely random one: given n nodes and a parameter $p \in (0, 1]$, each possible edge between the n nodes is added with probability p. We have generated several network of $n = 10\,000$ nodes, and with values of p ranging from 10^{-5} to 10^{-3}.

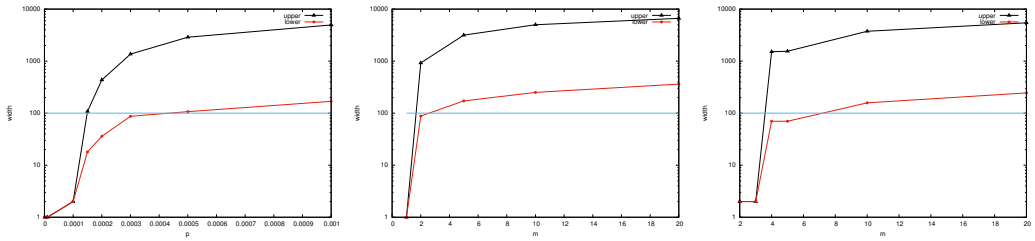
Preferential Attachment. This network model [6] is a generative model, and aims to simulate link formation in social networks: each time a new node is added to the network, it attaches itself to m other nodes, each link being formed with a probability proportional to the number of neighbors the existing node already has. We have generated graphs of $n = 10\,000$ nodes, with the parameter m varying between 1 and 20.

Small-World. The model [61] generates a small-world graph by the following process: first, the n nodes are organized in a ring graph where each node is connected with m other nodes on each side; finally, each edge is rewired (its endpoints are changed) with probability p. In the experiments, we have generated graphs of $n = 10\,000$, with $p = \{0.1, 0.2, 0.5\}$ and m ranging between 1 and 20.

Our objective with evaluating treewidth on these synthetic instances is to evaluate whether some parameters of these models lead to lower treewidth values, and if so, in which model the "phase transition" occurs, i.e., when a low-treewidth regime gives way to a high-treewidth one. For our purpose, and for reasons we explain in Section 5, we consider a treewidth under $\sqrt{n}$ to be low.

Our first finding – see Figure 4 – is that the high-treewidth regime arrives very early, relative to the parameter values. For random graphs, only relatively small value of p allow for a low treewidth; for small-world and preferential attachment only trivial values of m allow low-treewidth – and, even so, it is usually 1 or 2, possibly due to the fact that the graph, in those cases, is composed of several small connected components, i.e., the well-known subcritical regime of random graphs [15]. The low-treewidth regime for random networks seems to be limited to values immediately after $p = \frac{1}{n}$; after this point, the treewidth is high, in line with findings of a linear dependency between graph size and treewidth in random graphs [35]. Moreover, we notice that there is no smooth transition for preferential attachment and small world networks; the treewidth jumps from very low values to high treewidth values. This is understandable in scale-free networks – resulting from the preferential attachment

[1] We show in Figure 9 of Appendix D of [48] the full running time results for the upper and lower bound algorithms.



(a) random. **(b)** preferential attachment. **(c)** small world.

Figure 4 Lower and upper bounds of treewidth in synthetic networks, as a function of generator parameters. The blue line represent the low width regime, where treewidth is less than 100.

model – where a few hubs may exist with degrees that are comparable to the number of nodes in the graph. Comparatively, random networks exhibit a smoother progression – one can clearly see the shift from trivial values of treewidth, to relatively low values, and to high treewidth values. Finally, the gap between lower bound and upper bound tends to increase with the parameter; that is, however, not surprising since all three model graphs tend to have more edges with larger parameter values.

5 Partial Decompositions

Our results show that, in practice, the treewidths of real networks are quite high. Even in the case of road networks, having relatively low treewidths, their value can go in the hundreds, rendering most algorithms whose time is exponential time in the treewidth (or worse) unusable. In practical applications, however, we can still adapt treewidth-based approaches for obtaining data structures – not unlike indexes – which can help with some important graph queries like shortest distances and paths [62, 5] or probability estimations [47, 10].

The manner in which treewidth decomposition can be used starts from a simple observation made in studies on complex graphs, that is, that they tend to exhibit a *tree-like fringe* and a *densely connected core* [52, 51]. The tree-like fringe precisely corresponds to bounded-treewidth parts of the network. This yields an easy adaptation of the upper bound algorithms based on node ordering: given a parameter w representing the highest treewidth the fringe can be, we can run any greedy decomposition algorithm (Degree, FillIn, DegreeFillIn) until we only find nodes of degree $w+1$, at which point the algorithm stops. At termination, we obtain a data structure formed of a set of treewidth w elements (w-trees) interfacing through cliques that have size at most $w+1$ with a *core graph*. The core graph contains all the nodes not removed in the bag creation process, and has unbounded treewidth. Figure 5 illustrates the notion of partial decompositions.

The resulting structure can be thought of as a *partial decomposition* (or *relaxed decomposition*), a concept introduced in [62, 5] in the context of answering shortest path queries, and used in [47] for probabilistic distance queries. A partial decomposition can be extremely useful. The tree-like fringe can be used to quickly precompute answers to partial queries (e.g., precompute distances in the graph). Once the precomputation is done, these (partial) answers are added to the core graph, where queries can be answered directly. If the resulting core graph is much smaller than the original graph, the gains in running time can be considerable, as shown in [62, 5, 47]. Hence, the objective of our experiments in this section is to check how feasible partial decompositions are.

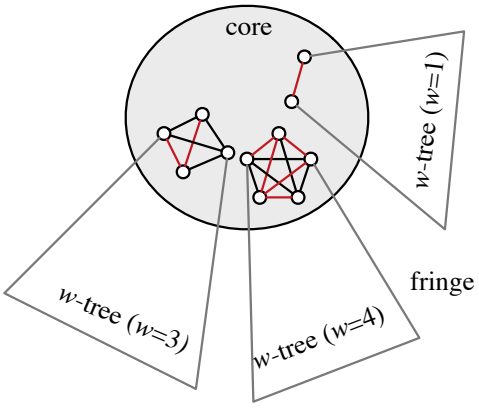
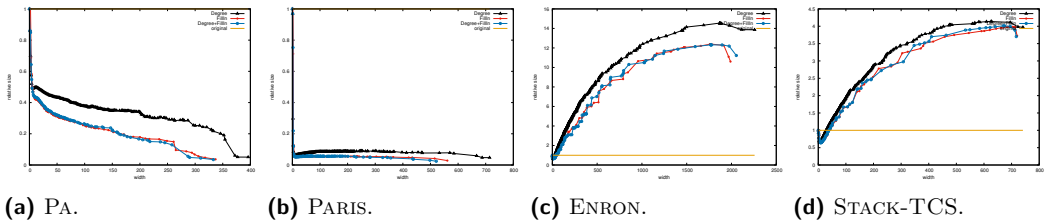


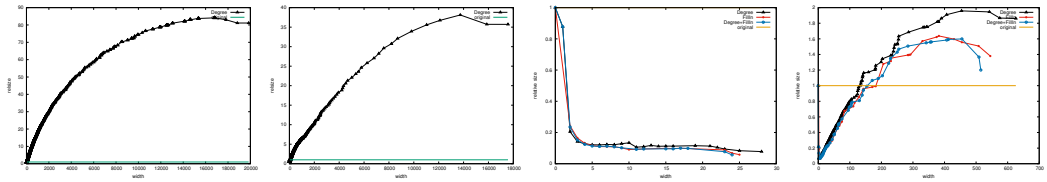
Figure 5 An abstract view of partial decompositions. Partial decompositions are formed of a *core* graph, which interfaces with w-trees through w-cliques (the *fringe*).

An interesting aspect of greedy upper bound algorithms is that they generate at any point a partial decomposition, with width equal to the highest degree encountered in the greedy ordering so far. The algorithm can then be stopped at any width, and the size of the resulting partial decomposition can be measured.

To evaluate this, we track here the size of the core graph, in terms of edges. We do this because most algorithms on graphs have a complexity that is directly related to the number of edges in the graph. As we discussed in the previous section, we aim that the size of the core graph to be as small as possible. Another aspect to keep in mind is the number of edges that are added by the fill-in process during the decomposition: each time a node of degree w is removed, at most $\frac{w(w-1)}{2}$ edges are added to the graph. Finally, for a graph of treewidth k, the decomposition contains a root of size at most k^2 edges. Hence, to ensure that the core graph is smaller than the original graph we aim for the treewidth to be $\sqrt{n}$. As we saw in Section 4, this is only rarely the case, and it is more likely to occur in infrastructure networks.



(a) PA. **(b)** PARIS. **(c)** ENRON. **(d)** STACK-TCS.



(e) WIKIPEDIA. **(f)** GOOGLE. **(g)** ROYAL. **(h)** MATH.

Figure 6 Relative sizes of core graphs in partial decompositions, after all bags of a given size have been removed in the decomposition.

Indeed, if we plot the core graph size in the partial decompositions of infrastructure networks (Figures 6a, 6b), we see that their size is only a fraction of the original size. We see a large drop in the size for low widths; this is logical, since the fill-in process does not add edges for $w = 1$ and $w = 2$. After this point, the size is relatively stable, and can go as low as 10% of the original size. In this sense, infrastructure networks seem the best fit for indexes based on decompositions, and represents further confirmation of the good behavior of these networks for hierarchical decompositions.

This desired decomposition behavior no longer occurs for social networks (Figures 6c, 6d) and the other networks in our dataset (Figures 6e, 6f). In this case, the decomposition becomes too large and even unwieldy – for some networks, the decomposition started filling the main memory of our computers (32 GB of RAM); for other networks, they did not finish in reasonable time (i.e., a few days of computation). For most of these networks, the resulting treewidth is much larger than our desired bound of at most $\sqrt{n}$); this results in core graph sizes that can be hundreds of times larger than the original size. The only exceptions are hierarchical networks, Royal and Math (Figures 6g, 6h), which are after all supposed to be close to a tree – despite having a relatively high treewidth (remember Figure 3b), partial decompositions work well on them. See Appendix E of [48] for full results.

In practice, however, we do not need to compute full decompositions. For usable indexes, we may want to stop at low treewidths, which allow exact computing. This may be useful for graphs whose decompositions have large core graphs. Indeed (see Appendix G of [48] for details), by stopping at width between 5 and 10, it is often possible to significantly reduce the original size of the graph, even in cases where core graphs are large, such as the Google graph.

Such partial decompositions are an extremely promising tool: on databases where partial decomposition result in a core graph of reduced size, efficient algorithms can be run on the fringe, exploiting its low treewidth, while more costly algorithms are used on the core, which has reduced size. Combining results from the fringe and the core graph can be a challenging process, however, which is worth investigating for separate problems. As a start, we have shown in [47] how to exploit partial decompositions to significantly improve the running time of source-to-target queries in probabilistic graphs.

6 Discussion

In this article, we have estimated the treewidth of a variety of graph datasets from different domains, and we have studied the running time and effectiveness of estimation algorithms. This study was motivated by the central role treewidth plays in many theoretical work attacking query evaluation tasks that are otherwise intractable, where low treewidths are an indicator for low complexity.

Our study on the *algorithms* for estimation leads to results that may seem surprising. For upper bounds, we discovered that greedy treewidth estimators, based on elimination orderings, provide the best cost–estimation trade-off; they also have the advantage of outputting readily-usable tree decompositions. In the case of lower bounds, we discovered that degeneracy-based bounds display the best behavior; moreover, the algorithms which aim to improve the bounds (LBN and LBN+) only very rarely do so.

In terms of *treewidth estimations*, we have discovered that, generally, the treewidth of real-world graphs is quite large. With few exceptions, most of the graphs we have tested in this study are *scale-free*. This may partly explain our findings – scale-free networks exhibit a number of high-degree, or *hub*, nodes that force high values for the treewidth. The few

exceptions to this rule are *infrastructure networks*, where treewidths are comparatively lower. Indeed, we were able to reproduce a $\mathcal{O}(\sqrt[3]{n})$ bound on the treewidth of road networks. We conjecture these relatively low bounds are explained by characteristics of infrastructure networks: specifically, they are similar to very sparse random networks.

Even in the case of infrastructure networks, the absolute value of treewidth still renders algorithms that are exponential in the treewidth impractical. However, one of the main lesson of this work is that it is still possible to exploit the structure of such datasets, by computing partial tree decompositions: following the approach of Section 5, it is often possible to decompose a dataset into a fringe of low treewidth and a smaller core of high treewidth. This has been used in [47], but for a very specific (and limited) application: connectivity queries in probabilistic graphs.

In brief, though low-treewidth data guarantees a wealth of theoretical results on the tractability of various data management tasks, this is unexploitable in most real-world datasets, which do not have low enough treewidth. One direction is of course to find relaxations of the treewidth notion, such as in [33, 41]. But since low treewidth is sometimes the only notion that ensures tractability [45, 34, 9], other approaches are needed; a promising one lies in the notion of partial tree decompositions. We believe future work in database theory should study in more detail the nature of partial tree decompositions, how to obtain optimal or near-optimal partial decompositions, and how to exploit them for improving the efficiency of a wide range of data management problems, from query evaluation, to enumeration, probability estimation, or knowledge compilation.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*. Addison-Wesley, 1995.

2 Ittai Abraham, Amos Fiat, Andrew V Goldberg, and Renato F Werneck. Highway dimension, shortest paths, and provably efficient algorithms. In *SODA*, 2010.

3 Aaron B. Adcock, Blair D. Sullivan, and Michael W. Mahoney. Tree decompositions and social graphs. *Internet Mathematics*, 12(5), 2016.

4 M. Ajtai, R. Fagin, and L. J. Stockmeyer. The Closure of Monadic NP. *JCSS*, 60(3), 2000.

5 Takuya Akiba, Christian Sommer, and Ken-ichi Kawarabayashi. Shortest-Path Queries for Complex Networks: Exploiting Low Tree-width Outside the Core. In *EDBT*, 2012.

6 Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74, 2002.

7 Antoine Amarilli, Pierre Bourhis, Louis Jachiet, and Stefan Mengel. A Circuit-Based Approach to Efficient Enumeration. In *ICALP*, 2017.

8 Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Provenance Circuits for Trees and Treelike Instances. In *ICALP*, 2015.

9 Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Tractable Lineages on Treelike Instances: Limits and Extensions. In *PODS*, 2016.

10 Antoine Amarilli, Silviu Maniu, and Mikaël Monet. Challenges for Efficient Query Evaluation on Structured Probabilistic Data. In *SUM*, 2016.

11 Antoine Amarilli, Mikaël Monet, and Pierre Senellart. Connecting Width and Structure in Knowledge Compilation. In *ICDT*, pages 6:1–6:17, 2018. doi:10.4230/LIPIcs.ICDT.2018.6.

12 Eyal Amir. Approximation Algorithms for Treewidth. *Algorithmica*, 56(4):448–479, 2010.

13 Stefan Arnborg, Derek G. Corneil, and Andrzej Proskuworski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8(2), 1987.

14 Guillaume Bagan. MSO queries on tree decomposable structures are computable with linear delay. In *CSL*, volume 4207, 2006.

15 Albert-László Barabási and Márton Pósfai. *Network science*. Cambridge University Press, 2016.

16 Anne Berry, Pinar Heggernes, and Geneviève Simonet. The Minimum Degree Heuristic and the Minimal Triangulation Process. *Graph-Theoretic Concepts in Computer Science*, 2880(Chapter 6), 2003.

17 Hans L. Bodlaender. A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth. *SIAM J. Comput.*, 25(6), 1996.

18 Hans L. Bodlaender, Fedor V Fomin, Arie M C A Koster, Dieter Kratsch, and Dimitrios M Thilikos. On exact algorithms for treewidth. *ACM TALG*, 9(1), 2012.

19 Hans L. Bodlaender and Arie M C A Koster. Treewidth Computations I. Upper Bounds. *Information and Computation*, 208(3), 2010.

20 Hans L. Bodlaender and Arie M C A Koster. Treewidth Computations II. Lower Bounds. *Information and Computation*, 209(7), 2011.

21 Hans L. Bodlaender, Arie M. C. A. Koster, and Thomas Wolle. Contraction and Treewidth Lower Bounds. In *ESA*, 2004.

22 Angela Bonifati, Wim Martens, and Thomas Timm. An Analytical Study of Large SPARQL Query Logs. *PVLDB*, 11(2), 2017.

23 François Clautiaux, Jacques Carlier, Aziz Moukrim, and Stéphane Nègre. New Lower and Upper Bounds for Graph Treewidth. In *WEA*, 2003.

24 Bruno Courcelle. The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs. *Inf. Comput.*, 85(1), 1990.

25 Nilesh N. Dalvi and Dan Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *PODS*, 2007.

26 Holger Dell, Christian Komusiewicz, Nimrod Talmon, and Mathias Weller. The PACE 2017 Parametrized Algorithms and Computation Experiments Challenge: The Second Iteration. In *IPEC*, 2017.

27 Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Customizable Contraction Hierarchies. *Journal of Experimental Algorithmics*, 21(1), 2016.

28 Vida Dujmović, David Eppstein, and David R Wood. Structure of Graphs with Locally Restricted Crossings. *SIAM J. Discrete Maths*, 31(2), 2017.

29 Arnaud Durand and Yann Strozecki. Enumeration Complexity of Logical Query Problems with Second-order Variables. In *CSL*, 2011.

30 Paul Erdős and Alfréd Rényi. On Random Graphs I. *Publicationes Mathematicae (Debrecen)*, 6, 1959.

31 Johannes K. Fichte, Neha Lodha, and Stefan Szeider. SAT-Based Local Improvement for Finding Tree Decompositions of Small Width. In Serge Gaspers and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing – SAT 2017*, 2017.

32 Jörg Flum, Markus Frick, and Martin Grohe. Query evaluation via tree-decompositions. *J. ACM*, 49(6), 2002.

33 Markus Frick and Martin Grohe. Deciding first-order properties of locally tree-decomposable structures. *J. ACM*, 48(6), 2001.

34 Robert Ganian, Petr Hliněnỳ, Alexander Langer, Jan Obdržálek, Peter Rossmanith, and Somnath Sikdar. Lower bounds on the complexity of MSO1 model-checking. *JCSS*, 1(80), 2014.

35 Yong Gao. Treewidth of Erdős–Rényi random graphs, random intersection graphs, and scale-free random graphs. *Discrete Applied Mathematics*, 160(4–5), 2012.

36 Vibhav Gogate and Rina Dechter. A Complete Anytime Algorithm for Treewidth. In *UAI*, 2004.

37 Martin Grohe, Thomas Schwentick, and Luc Segoufin. When is the Evaluation of Conjunctive Queries Tractable? In *STOC*, 2001.

38 Daniel John Harvey. *On Treewidth and Graph Minors*. PhD thesis, The University of Melbourne, 2014.

39 Abhay Jha and Dan Suciu. On the Tractability of Query Compilation and Bounded Treewidth. In *ICDT*, 2012.

40 Abhay Kumar Jha and Dan Suciu. Knowledge Compilation Meets Database Theory: Compiling Queries to Decision Diagrams. *Theory Comput. Syst.*, 52(3), 2013.

41 Wojciech Kazana and Luc Segoufin. Enumeration of first-order queries on classes of structures with bounded expansion. In *PODS*, 2013.

42 Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press, 2009.

43 Arie M. C. A. Koster, Thomas Wolle, and Hans L. Bodlaender. Degree-Based Treewidth Lower Bounds. In *WEA*, 2005.

44 Stephan Kreutzer. Algorithmic Meta-Theorems. *CoRR*, abs/0902.3616, 2009.

45 Stephan Kreutzer and Siamak Tazari. Lower bounds for the complexity of monadic second-order logic. In *LICS*, 2010.

46 S. L. Lauritzen and D. J. Spiegelhalter. Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems. *Journal of the Royal Statistical Society Series B (Methodological)*, 50(2), 1988.

47 Silviu Maniu, Reynold Cheng, and Pierre Senellart. An Indexing Framework for Queries on Probabilistic Graphs. *ACM Trans. Database Syst.*, 42(2), 2017.

48 Silviu Maniu, Pierre Senellart, and Suraj Jog. An Experimental Study of the Treewidth of Real-World Graph Data (Extended Version). *arXiv.org*, 2019. `arXiv:1901.06862`.

49 Harry M. Markowitz. The Elimination Form of the Inverse and Its Application to Linear Programming. *Management Science*, 3(3), 1957.

50 Mikaël Monet. Probabilistic Evaluation of Expressive Queries on Bounded-Treewidth Instances. In *SIGMOD PhD Symposium*, 2016.

51 M. E. J. Newman, D. J. Watts, and S. H. Strogatz. Random graph models of social networks. In *Proceedings of the National Academy of Sciences*, 2002.

52 Mark E. J. Newman, Steven H. Strogatz, and Duncan J. Watts. Random graphs with arbitrary degree distributions and their applications. *Phys. Rev. E*, 64, July 2001.

53 François Picalausa and Stijn Vansummeren. What are real SPARQL queries like? In *SWIM*, 2011.

54 Léon Planken, Mathijs de Weerdt, and Roman van der Krogt. Computing All-Pairs Shortest Paths by Leveraging Low Treewidth. *Journal of Artificial Intelligence Research*, 43, 2012.

55 Luca Pulina and Armando Tacchella. An Empirical Study of QBF Encodings: from Treewidth Estimation to Useful Preprocessing. *Fundamenta Informaticae*, 102:391–427, 2010.

56 Neil Robertson and Paul D. Seymour. Graph minors. III. Planar tree-width. *J. Comb. Theory, Ser. B*, 36(1), 1984.

57 S. Saluja, K.V. Subrahmanyam, and M.N. Thakur. Descriptive Complexity of #P Functions. *JCSS*, 50(3), 1995.

58 James W. Thatcher and Jesse B. Wright. Generalized Finite Automata Theory with an Application to a Decision Problem of Second-Order Logic. *Math. Systems Theory*, 2(1), 1968.

59 Thomas van Dijk, Jan-Pieter van den Heuvel, and Wouter Slob. Computing treewidth with LibTW. Technical report, University of Utrecht, 2006.

60 M. Y. Vardi. The Complexity of Relational Query Languages (Extended Abstract). In *STOC*, 1982.

61 D. J. Watts and S. H. Strogatz. Collective dynamics of small-world networks. *Nature*, 393, 1998.

62 Fang Wei. TEDI: efficient shortest path query answering on graphs. In *SIGMOD*, 2010.

63 Thomas Wolle. *Computational aspects of treewidth: Lower bounds and network reliability*. PhD thesis, Utrecht University, 2005.

Recursive Programs for Document Spanners

Liat Peterfreund[1]
Technion, Haifa 32000, Israel
liatpf@cs.technion.ac.il

Balder ten Cate
Google, Inc., Mountain View, CA 94043, USA
balder.tencate@gmail.com

Ronald Fagin
IBM Research – Almaden, San Jose, CA 95120, USA
fagin@us.ibm.com

Benny Kimelfeld
Technion, Haifa 32000, Israel
bennyk@cs.technion.ac.il

Abstract

A document spanner models a program for Information Extraction (IE) as a function that takes as input a text document (string over a finite alphabet) and produces a relation of spans (intervals in the document) over a predefined schema. A well-studied language for expressing spanners is that of the regular spanners: relational algebra over regex formulas, which are regular expressions with capture variables. Equivalently, the regular spanners are the ones expressible in non-recursive Datalog over regex formulas (which extract relations that constitute the extensional database). This paper explores the expressive power of recursive Datalog over regex formulas. We show that such programs can express precisely the document spanners computable in polynomial time. We compare this expressiveness to known formalisms such as the closure of regex formulas under the relational algebra and string equality. Finally, we extend our study to a recently proposed framework that generalizes both the relational model and the document spanners.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic; Information systems → Relational database model; Information systems → Data model extensions

Keywords and phrases Information Extraction, Document Spanners, Polynomial Time, Recursion, Regular Expressions, Datalog

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.13

Funding *Liat Peterfreund*: Supported by the Technion Hiroshi Fujiwara Cyber Security Research Center and the Israel Cyber Bureau and by the Israel Science Foundation (ISF) Grant 1295/15.
Benny Kimelfeld: Supported by the Israel Science Foundation (ISF) Grant 1295/15.

1 Introduction

The abundance and availability of valuable textual resources position text analytics as a standard component in data-driven workflows. To facilitate the incorporation of such resources, a core operation is the extraction of structured data from text, a classic task known as Information Extraction (IE). This task arises in a large variety of domains, including healthcare analysis [28], social media analysis [4], customer relationship management [2], and machine log analysis [12]. IE also plays a central role in cross-domain computational challenges such as Information Retrieval [30] and knowledge-base construction [26, 27, 15, 29].

[1] The work was done while the author was at IBM Research – Almaden.

Rule-based IE is incorporated in commercial systems and academic prototypes for text analytics, either as a standalone extraction language or within machine-learning models. IBM's SystemT [20] exposes an SQL-like declarative language, *AQL* (Annotation Query Language), for programming IE. Conceptually, AQL supports a collection of "primitive" extractors of relations from text (e.g., tokenizer, dictionary lookup, part-of-speech tagger and regular-expression matcher), together with a relational algebra for manipulating these relations. Similarly, in Xlog [25], user-defined functions are used as primitive extractors, and non-recursive Datalog is, again, allowed for relation manipulation. In DeepDive [26, 24], rules are used to generate features that are translated into the factors of a statistical model with machine-learned parameters. Feature declaration combines, once again, primitive extractors of relations alongside relational operators on these relations. In addition to the above, different Datalog-like formalisms for IE were previously suggested and studied, including monadic Datalog over trees as web-page languages [13], and a framework for annotating CSV documents [3].

The framework of *document spanners* (or just *spanners* for short) [7] captures the above IE methodology: a spanner is a function that extracts from a document a relation over text intervals, called *spans*, using either a primitive extractor (e.g., a regular expression) or a relational query on top of primitive extractors. More formally, by a *document* we refer to a string $\mathbf{s}$ over a finite alphabet, and a *span* of $\mathbf{s}$ represents a substring of $\mathbf{s}$ by its start and end positions. A spanner is a function P that maps every string $\mathbf{s}$ into a relation $P(\mathbf{s})$, over a fixed schema S_P, over the spans of $\mathbf{s}$. The most studied spanner language is that of the *regular* spanners: primitive extraction is via *regex formulas*, which are regular expressions with capture variables, and relational manipulation is via positive relational algebra: projection, natural join, and union (while difference is expressible and not explicitly needed) [7]. Equivalently, the regular spanners are the ones expressible in non-recursive Datalog, where regex formulas are playing the role of the Extensional Data Base (EDB), that is, the input database [8].

By adding string-equality selection on span variables, Fagin et al. [7] establish the extended class of *core* spanners, viewed as the core language for AQL. A syntactically different language for spanners is SpLog, which is based on the *existential theory of concatenation*, and was shown by Freydenberger [9] to have precisely the expressiveness of core spanners. Such spanners can express more than regular spanners. A simple example is the spanner that extracts from the input $\mathbf{s}$ all spans x and y such that the string $\mathbf{s}_x$ spanned by x is equal to the string $\mathbf{s}_y$ spanned by y. The class of core spanners does not behave as well as that of the regular spanners; for instance, core spanners are not closed under difference, while regular spanners are. Fagin et al. [7] prove this by showing that no core spanner extracts all spans x and y such that $\mathbf{s}_x$ is *not* a substring of $\mathbf{s}_y$. The proof is based on the *core simplification lemma*: every core spanner can be represented as a regular spanner followed by a sequence of string equalities and projections. The same technique has been used for showing that no core spanner extracts all pairs x and y of spans having the same *length* [7].

In this paper we explore the power of *recursion* in expressing spanners. The motivation came from the SystemT developers, who have interest in recursion for various reasons, such as programming basic natural-language parsers by means of context-free grammars [19]. Specifically, we consider the language RGXlog of spanners that are defined by means of Datalog where, again, regex formulas play the role of EDB relations, but this time recursion is allowed. More precisely, given a string $\mathbf{s}$, the regex formulas extract EDB relations from $\mathbf{s}$, and a designated relation Out captures the output of the program. Observe that such a program operates exclusively over the domain of spans of the input string. In particular,

the output is a relation over spans of $\mathbf{s}$, and hence, RGXlog is yet another representation language for spanners. As an example, the following program emits all pairs x and y of spans of equal lengths. (See Section 3 for the formal definition of the syntax and semantics.)

- $\text{EQL}(x, y) \leftarrow \langle x\{\epsilon\}\rangle, \langle y\{\epsilon\}\rangle$
- $\text{EQL}(x, y) \leftarrow \langle x\{x'\{.^*\}.\}\rangle, \langle y\{y'\{.^*\}.\}\rangle, \text{EQL}(x', y')$

The first rule states that two empty spans have same length. The second rule states that two spans x and y have equal lengths if they are obtained by adding a single symbol (represented by dot) to spans x' and y', respectively, of equal lengths.

We explore the expressiveness of RGXlog. Without recursion, RGXlog captures precisely the regular spanners [8]. With recursion, several observations are quite straightforward. First, we can write a program that determines whether x and y span the same string. Hence, we have string equality without explicitly including the string-equality predicate. It follows that every core spanner can be expressed in RGXlog. Moreover, RGXlog can express more than core spanners, an example being expressing that two spans have the same length (which the above program shows can be expressed in RGXlog, but which, as said earlier, is not expressible by a core spanner [7]). What about upper bounds? A clear upper bound is *polynomial time*: every RGXlog program can be evaluated in polynomial time (under data complexity, where the spanner is fixed and the input consists of only the string), and hence, RGXlog can express only spanners computable in polynomial time.

We begin our investigation by diving deeper into the relationship between RGXlog and core spanners. The inexpressiveness results to date are based on the aforementioned core simplification lemma [7]. The proof of this lemma heavily relies on the absence of the difference operator in the algebra. In fact, Freydenberger and Holldack [10] showed that it is unlikely that in the presence of difference, there is a result similar to the core simplification lemma. So, we extend the algebra of core spanners with the difference operator, and call a spanner of this extended language a *generalized core spanner*. We then ask whether *(a)* every generalized core spanner can be expressed in RGXlog (whose syntax is positive and excludes difference/negation), and *(b)* RGXlog can express only generalized core spanners.

The answer to the first question is positive. We establish a negative answer to the second question by deploying the theory of *Presburger arithmetic* [23]. Specifically, we consider Boolean spanners on a unary alphabet. Each such spanner can be viewed as a predicate over natural numbers: the lengths of the strings that are accepted (evaluated to **true**) by the spanner. We prove that every predicate expressible by a Boolean generalized core spanner is also expressible in Presburger arithmetic (first-order theory of the natural numbers with the addition (+) binary function and the constant 0 and 1). Yet, we show a very simple RGXlog program that expresses a predicate that is *not* expressible in Presburger arithmetic, namely being a power of two [17].

We prove that RGXlog can express *every* spanner computable in polynomial time. Formally, recall that a spanner is a function P that maps an input string $\mathbf{s}$ into a relation $P(\mathbf{s})$, over a fixed schema S_P, over the spans of $\mathbf{s}$. We prove that the following are equivalent for a spanner P: *(a)* P is expressible in RGXlog, *and (b)* P is computable in polynomial time. As a special case, Boolean RGXlog captures exactly the polynomial-time languages.

Related formalisms that capture polynomial time include the *Range Concatenation Grammars* (RCG) [5]. In RCG, the grammar defines derivation rules for reducing the input string into the empty string; if reduction succeeds, then the string is accepted. Unlike context-free and context-sensitive grammars, RCGs have predicate names in addition to variables and terminals – this allows us to maintain connections between different parts

of the input string. Another formalism that captures polynomial time is the multi-head alternating automata [16], which are finite state machines with several cursors that can perform alternating transitions. Though related, these results do not seem to imply our results on document spanners.

We prove equivalence to polynomial time via a result by Papadimitriou [22], stating that semipositive Datalog (i.e., Datalog where only EDB relations can be negated) can express every database property computable in polynomial time, under certain assumptions: *(a)* the property is invariant under isomorphism, *(b)* a successor relation that defines a linear order over the domain is accessible as an EDB, and *(c)* the first and last elements in the database are accessible as constants (or single-element EDBs). We show that in the case of RGXlog, we get all of these for free, due to the fact that our EDBs are regex formulas. Specifically, in string logic (over a finite alphabet), isomorphism coincides with identity, negation of EDBs (regex formulas) are expressible as EDBs (regex formulas), and we can express a linear order by describing a successor relation along with its first and last elements.

Interestingly, our construction shows that, to express polynomial time, it suffices to use regex formulas with only two variables. In other words, binary regex formulas already capture the entire expressive power. Can we get away with only unary regex formulas? Using past results on monadic Datalog [14] and non-recursive RGXlog [7] we conclude a negative answer – Boolean RGXlog with unary regex formulas can express *precisely* the class of Boolean regular spanners. In fact, we can characterize explicitly the class of spanners expressible by RGXlog with unary regex formulas.

Lastly, we analyze recursive Datalog programs in a framework that generalizes both the relational and the spanner model. The framework, introduced by Nahshon, Peterfreund and Vansummeren [21] and referred to as Spannerlog⟨RGX⟩, aims to establish a unified query language for combining structured and textual data. In this framework, the input and output databases consist of relations that have two types of attributes: strings and spans. In the associated Datalog program, we refer to the relations of the input database as EDB relations (that is, *extensional*) and to those of the output database as IDB relations (that is, *intensional*, or *inferred*). The body of a Datalog rule may have three types of atoms: EDB, IDB, and regex formulas over string attributes. We prove that Spannerlog⟨RGX⟩ with stratified negation, restricted to string EDB relations, can express *precisely* the queries that are computable in polynomial time.

The remainder of the paper is organized as follows. We provide basic definitions and terminology in Section 2, and introduce RGXlog in Section 3. In Section 4, we illustrate RGXlog in the context of a comparison with (generalized) core spanners. Our main result (equivalence to polynomial time) is proved in Section 5. We describe the generalization of our main result to Spannerlog⟨RGX⟩ in Section 6, and conclude in Section 7.

2 Preliminaries

We first introduce the basic terminology and notation that we use throughout the paper.

2.1 Document Spanners

We begin with the basic terminology from the framework of document spanners [7].

Strings and spans. We fix a finite alphabet Σ of symbols. A *string* $\mathbf{s}$ is a finite sequence $\sigma_1 \cdots \sigma_n$ over Σ (i.e., each $\sigma_i \in \Sigma$). We denote by Σ^* the set of all strings over Σ. A *language* over Σ is a subset of Σ^*. A *span* identifies a substring of $\mathbf{s}$ by specifying its bounding indices.

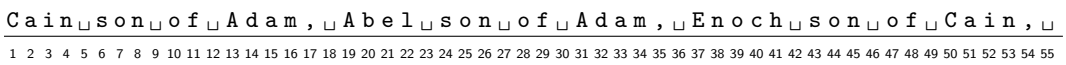


Figure 1 The input string $\mathbf{s}$ in our running example.

Formally, a span of $\mathbf{s}$ has the form $[i, j\rangle$ where $1 \leq i \leq j \leq n+1$. If $[i, j\rangle$ is a span of $\mathbf{s}$, then $\mathbf{s}_{[i,j\rangle}$ denotes the substring $\sigma_i \cdots \sigma_{j-1}$. Note that $\mathbf{s}_{[i,i\rangle}$ is the empty string, and that $\mathbf{s}_{[1,n+1\rangle}$ is $\mathbf{s}$. Note also that the spans $[i, i\rangle$ and $[j, j\rangle$, where $i \neq j$, are different, even though $\mathbf{s}_{[i,i\rangle} = \mathbf{s}_{[j,j\rangle} = \epsilon$ where ϵ stands for the empty string. We denote by Spans the set of all spans of all strings, that is, all expressions $[i, j\rangle$ where $1 \leq i \leq j$. By $\mathsf{Spans}(\mathbf{s})$ we denote the set spans of string $\mathbf{s}$ (and in this case we have $j \leq n+1$).

▶ **Example 1.** In all of the examples throughout the paper, we use the alphabet Σ that consists of the lowercase and capital letters from the English alphabet (i.e., $\mathtt{a}, \ldots, \mathtt{z}$ and $\mathtt{A}, \ldots, \mathtt{Z}$), the comma symbol ",", and the symbol "␣" that stands for whitespace. Figure 1 depicts an example of a prefix of a string $\mathbf{s}$. (For convenience, it also depicts the position of each of the characters in $\mathbf{s}$.) Observe that the spans $[13, 17\rangle$ and $[31, 35\rangle$ are different, yet they span the same substring, that is, $\mathbf{s}_{[13,17\rangle} = \mathbf{s}_{[31,35\rangle} = \mathtt{Adam}$.

Document spanners. We assume an infinite collection Vars of *variables* such that Vars and Σ are disjoint. Let $\mathbf{s}$ be a string and $V \subset \mathsf{Vars}$ a finite set of variables. A $(V, \mathbf{s})$-*record*[2] is a function $r : V \to \mathsf{Spans}(\mathbf{s})$ that maps the variables of V to spans of $\mathbf{s}$. A $(V, \mathbf{s})$-*relation* is a set of $(V, \mathbf{s})$-records. A *document spanner* (or just *spanner* for short) is a function P that maps strings $\mathbf{s}$ to $(V, \mathbf{s})$-relations $P(\mathbf{s})$, for a predefined finite set V of variables that we denote by $\mathsf{Vars}(P)$. As a special case, a *Boolean spanner* is a spanner P such that $\mathsf{Vars}(P) = \emptyset$; in this case, $P(\mathbf{s})$ can be either the singleton that consists of the empty function, a situation denoted by $P(\mathbf{s}) = \mathbf{true}$, or the empty set, a situation denoted by $P(\mathbf{s}) = \mathbf{false}$. A Boolean spanner P *recognizes* the language $\{\mathbf{s} \in \Sigma^* \mid P(\mathbf{s}) = \mathbf{true}\}$.

By a *spanner representation language*, or simply *spanner language* for short, we refer to a collection L of finite expressions p that represent a spanner. For instance, we next define the spanner language RGX of regex formulas. For an expression p in a spanner language, we denote by $[\![p]\!]$ the spanner that is defined by p, and by $\mathsf{Vars}(p)$ the variable set $\mathsf{Vars}([\![p]\!])$. Hence, for a string $\mathbf{s}$ we have that $[\![p]\!](\mathbf{s})$ is a $(\mathsf{Vars}(p), \mathbf{s})$-relation. We denote by $[\![L]\!]$ the class of all spanners $[\![p]\!]$ definable by expressions p in L.

Regex formulas. A *regex formula* is a representation of a spanner by means of a regular expression with *capture variables*. It is defined by $\gamma := \emptyset \mid \epsilon \mid \sigma \mid \gamma \vee \gamma \mid \gamma \cdot \gamma \mid \gamma^* \mid x\{\gamma\}$. Here, ϵ stands for the empty string, $\sigma \in \Sigma$, and the alternative beyond regular expressions is $x\{\gamma\}$ where x is a variable in Vars. We denote the set of variables that occur in γ by $\mathsf{Vars}(\gamma)$. Intuitively, every *match* of a regex formula in an input string $\mathbf{s}$ yields an assignment of spans to the variables of γ. A crucial assumption we make is that the regex formula is *functional* [7], which intuitively means that every match assigns precisely one span to each variable in $\mathsf{Vars}(\gamma)$. For example, the regex formula $\mathtt{a}^* \cdot x\{\mathtt{a} \cdot \mathtt{b}^*\} \cdot \mathtt{a}$ is functional, but $\mathtt{a}^* \cdot (x\{\mathtt{a} \cdot \mathtt{b}\})^* \cdot \mathtt{a}$ is not; similarly, $(x\{\mathtt{a}\}) \vee (\mathtt{b} \cdot x\{\mathtt{a}\})$ is functional, but $(x\{\mathtt{a}\}) \vee (\mathtt{b} \cdot \mathtt{a})$ is not. A regex formula γ defines a spanner, where the matches produce the $(V, \mathbf{s})$-records for

[2] Fagin et al. [7] refer to $(V, \mathbf{s})$-records as $(V, \mathbf{s})$-*tuples*; we use "record" to avoid confusion with the concept of "tuple" that we later use in ordinary relations.

$V = \mathsf{Vars}(\gamma)$. We refer the reader to Fagin et al. [7] for the precise definition of functionality, including its polynomial-time verification, and for the precise definition of the spanner $[\![\gamma]\!]$ represented by γ. As previously said, we denote by RGX the spanner language of (i.e., the set of all) regex formulas.

Throughout the paper, we use the following abbreviations when we define regex formulas. We use "." instead of "$\vee_{\sigma\in\Sigma}\sigma$" (e.g., we use ".*" instead of "$(\vee_{\sigma\in\Sigma}\sigma)^*$"). For convenience, we put regex formulas in brackets and write $\langle\gamma\rangle$ (using angular instead of ordinary brackets) to denote that γ can occur anywhere in the document; that is, $\langle\gamma\rangle := [.^*\,\gamma\,.^*]$.

▶ **Example 2.** Following are examples of regex formulas that we use later on.

- $\gamma_{\mathsf{token}}(x) := \langle\ \sqcup\ x\{(\mathtt{a}-\mathtt{zA}-\mathtt{Z})^*\}\ (\sqcup \vee\ ,)\ \rangle$
- $\gamma_{\mathsf{cap}}(x) := \langle\ \sqcup\ x\{(\mathtt{A}-\mathtt{Z})(\mathtt{a}-\mathtt{zA}-\mathtt{Z})^*\}\ (\sqcup \vee\ ,)\ \rangle$
- $\gamma_{\mathsf{prnt}}(x,y) := \langle y\{.^*\}\sqcup\mathtt{son}\sqcup\mathtt{of}\sqcup x\{.^*\}\rangle$

The regex formula $\gamma_{\mathsf{token}}(x)$ extracts the spans of tokens (defined simplistically for presentation sake), $\gamma_{\mathsf{cap}}(x)$ extracts capitalized tokens, and $\gamma_{\mathsf{prnt}}(x,y)$ extracts spans separated by $\sqcup\mathtt{son}\sqcup\mathtt{of}\sqcup$ (where prnt stands for "parent"). For illustration, applying $[\![\gamma_{\mathsf{cap}}]\!]$ to $\mathbf{s}$ of Figure 1 results in a set of $(\{x\},\mathbf{s})$-records that includes the record r that maps x to $[19,23\rangle$.

2.2 Spanner Algebra

The algebraic operators *union*, *projection*, *natural join*, and *difference* are defined in the usual way, for all spanners P_1 and P_2 and strings $\mathbf{s}$, as follows. For a $(V,\mathbf{s})$-record r and $Y \subseteq V$, we denote by $r\restriction Y$ the $(Y,\mathbf{s})$-record obtained by restricting r to the variables in Y. We say that P_1 and P_2 are *union compatible* if $\mathsf{Vars}(P_1) = \mathsf{Vars}(P_2)$.

- **Union:** Assuming P_1 and P_2 are union compatible, the union $P = P_1 \cup P_2$ is defined by $\mathsf{Vars}(P) := \mathsf{Vars}(P_1)$ with $P(\mathbf{s}) := P_1(\mathbf{s}) \cup P_2(\mathbf{s})$.
- **Projection:** For $Y \subseteq \mathsf{Vars}(P_1)$, the projection $P = \pi_Y P_1$ is defined by $\mathsf{Vars}(P) := Y$ with $P(\mathbf{s}) = \{r\restriction Y \mid r \in P_1(\mathbf{s})\}$.
- **Natural join:** Let $V_i := \mathsf{Vars}(P_i)$ for $i \in \{1,2\}$. The *(natural) join* $P = (P_1 \bowtie P_2)$ is defined by $\mathsf{Vars}(P) := \mathsf{Vars}(P_1) \cup \mathsf{Vars}(P_2)$ with $P(\mathbf{s})$ consisting of all $(V_1 \cup V_2, \mathbf{s})$-records r such that there exist $r_1 \in P_1(\mathbf{s})$ and $r_2 \in P_2(\mathbf{s})$ with $r\restriction V_1 = r_1$ and $r\restriction V_2 = r_2$.
- **Difference:** Assuming P_1 and P_2 are union compatible, the difference $P = P_1 \setminus P_2$ is defined by $\mathsf{Vars}(P_1 \setminus P_2) := \mathsf{Vars}(P_1)$ with $P(\mathbf{s}) := P_1(\mathbf{s}) \setminus P_2(\mathbf{s})$.
- **String-equality selection:** For variables x and y in $\mathsf{Vars}(P_1)$, the string-equality selection $P := \zeta^{=}_{x,y} P_1$ is defined by $\mathsf{Vars}(P) := \mathsf{Vars}(P_1)$ with $P(\mathbf{s})$ consisting of all records $r \in P_1(\mathbf{s})$ such that $\mathbf{s}_{r(x)} = \mathbf{s}_{r(y)}$.

If L is a spanner language and O is a set of operators in a spanner algebra, then L^O denotes the spanner language obtained by closing L under the operations of O.

2.3 Regular and (Generalized) Core Spanners

Following Fagin et al. [7], we define a *regular* spanner to be one definable in $\mathsf{RGX}^{\{\cup,\pi,\bowtie\}}$, that is, a spanner P such that $P = [\![p]\!]$ for some p in $\mathsf{RGX}^{\{\cup,\pi,\bowtie\}}$. Similarly, we define a *core* spanner to be a spanner definable in $\mathsf{RGX}^{\{\cup,\pi,\bowtie,\zeta^{=}\}}$.

▶ **Example 3.** Consider the regex formulas of Example 2. We can take their join and obtain a regular spanner: $\gamma_{\mathsf{prnt}}(x,y) \bowtie \gamma_{\mathsf{cap}}(x) \bowtie \gamma_{\mathsf{cap}}(y)$. This spanner extracts a set of $(\{x,y\},\mathbf{s})$-records r such that r maps x and y to strings that begin with a capital letter and are separated by $\sqcup\mathtt{son}\sqcup\mathtt{of}\sqcup$. Assume we wish to extract a binary relation that holds the

tuples (x, y) such that the span x spans the name of the grandparent of y. (For simplicity, we assume that the name is a unique identifier of a person.) For that, we can define the following core spanner on top of the regex formulas from Example 2: $\pi_{x,w}\zeta^{=}_{y,z}\big(\gamma_{\mathsf{prnt}}(x, y) \bowtie \gamma_{\mathsf{prnt}}(z, w)\big)$. We denote this spanner by $\gamma_{\mathsf{grpr}}(x, w)$.

Note that we did not include *difference* in the definition of regular and core spanners; this does not matter for the class of *regular* spanners, since it is closed to difference (i.e., a spanner is definable in $\mathsf{RGX}^{\{\cup,\pi,\bowtie,\setminus\}}$ if and only if it is definable in $\mathsf{RGX}^{\{\cup,\pi,\bowtie\}}$), but it matters for the class of *core* spanners, which is *not* closed under difference [7]. We define a *generalized core spanner* to be a spanner definable in $\mathsf{RGX}^{\{\cup,\pi,\bowtie,\zeta^{=},\setminus\}}$. We study the expressive power of the class of generalized core spanners in Section 4.

▶ **Example 4.** Recall the definition of $\gamma_{\mathsf{grpr}}(x, w)$ from Example 3. The generalized core spanner $\gamma_{\mathsf{cap}}(w) \setminus (\pi_w\gamma_{\mathsf{grpr}}(x, w))$ finds all spans of capitalized words w such that the text has no mentioning of any grandparent of w.

2.4 Span Databases

We also use the terminology and notation of ordinary relational databases, with the exception that database values are all spans. (In Section 6 we allow more general values in the database.) More formally, a *relation symbol* R has an associated arity that we denote by $\mathsf{arity}(R)$, and a *span relation* over R is a finite set of *tuples* $\mathbf{t} \in \mathsf{Spans}^{\mathsf{arity}(R)}$ over R. We denote the ith element of a tuple $\mathbf{t}$ by $\mathbf{t}_i$. A *(relational) signature* $\mathcal{R}$ is a finite set $\{R_1, \ldots, R_n\}$ of relation symbols. A *span database* D over a signature $\mathcal{R} := \{R_1, \ldots, R_n\}$ consists of span relations R_i^D over the R_i. We call R_i^D the *instantiation* of R_i by D.

3 RGXlog: Datalog over Regex Formulas

In this section, we define the spanner language RGXlog, pronounced "regex-log," that generalizes regex formulas to (possibly recursive) Datalog programs.

Let $\mathcal{R}$ be a signature. By an *atom* over $\mathcal{R}$ we refer to an expresion of the form $R(x_1, \ldots, x_k)$ where $R \in \mathcal{R}$ is a k-ary relation symbol and each x_i is a variable in Vars. Note that a variable can occur more than once in an atom (i.e., we may have $x_i = x_j$ for some i and j with $i \neq j$), and we do not allow constants in atoms. A RGXlog *program* is a triple $\langle \mathcal{I}, \Phi, \textsc{Out}(\mathbf{x})\rangle$ where:

- $\mathcal{I}$ is a signature referred to as the *IDB* signature;
- Φ is a finite set of *rules* of the form $\varphi \leftarrow \psi_1, \ldots, \psi_m$, where φ is an atom over $\mathcal{I}$, and each ψ_i is either an atom over $\mathcal{I}$ or a regex formula;
- $\textsc{Out} \in \mathcal{I}$ is a designated *output* relation symbol;
- $\mathbf{x}$ is a sequence of k distinct variables in Vars, where k is the arity of $\textsc{Out}$.

If ρ is the rule $\varphi \leftarrow \psi_1, \ldots, \psi_m$, then we call φ the *head* of ρ and $\psi_1, \ldots, \psi_m$ the *body* of ρ. Each variable in φ is called a *head variable* of ρ. We make the standard assumption that each head variable of a rule occurs at least once in the body of the rule.

We now define the semantics of evaluating a RGXlog program over a string. Let $Q = \langle \mathcal{I}, \Phi, \textsc{Out}(\mathbf{x})\rangle$ be a RGXlog program, and let $\mathbf{s}$ be a string. We evaluate Q on $\mathbf{s}$ using the usual fixpoint semantics of Datalog, while viewing the regex formulas as extensional-database (EDB) relations. More formally, we view a regex formula γ as a logical assertion over assignments to $\mathsf{Vars}(\gamma)$, stating that the assignment forms a tuple in $[\![\gamma]\!](\mathbf{s})$. The span database with signature $\mathcal{I}$ that results from applying Q to $\mathbf{s}$ is denoted by $Q(\mathbf{s})$, and it is the minimal span database that satisfies all rules, when viewing each left arrow ($\leftarrow$) as a logical implication with all variables being universally quantified.

Next, we define the semantics of RGXlog as a spanner language. Let $Q = \langle \mathcal{I}, \Phi, \textsc{Out}(\mathbf{x}) \rangle$ be a RGXlog program. As a spanner, the program Q constructs $D = Q(\mathbf{s})$ and emits the relation $\textsc{Out}^D$ as assignments to $\mathbf{x}$. More precisely, suppose that $\mathbf{x} = x_1, \ldots, x_k$. The spanner $P = [\![Q]\!]$ is defined as follows.

- $\mathsf{Vars}(P) := \{x_1, \ldots, x_k\}$.
- Given $\mathbf{s}$ and $D = Q(\mathbf{s})$, the set $P(\mathbf{s})$ consists of all records $r_{\mathbf{a}}$ obtained from tuples $\mathbf{a} = (a_1, \ldots, a_k) \in \textsc{Out}^D$ by setting $r_{\mathbf{a}}(x_i) = a_i$.

Finally, *recursive* and *non-recursive* RGXlog programs are defined similarly to ordinary Datalog (e.g., using the acyclicity of the dependency graph over the IDB predicates).

▶ **Example 5.** In the following and later examples of programs, we use the cursor sign ▶ to indicate where a rule begins. Importantly, for brevity we use the following convention: $\textsc{Out}(\mathbf{x})$ is always the left hand side of the last rule.

- ▶ $\textsc{Ancstr}(x, z) \leftarrow \gamma_{\mathsf{prnt}}(x, z)$
- ▶ $\textsc{Ancstr}(x, y) \leftarrow \textsc{Ancstr}(x, z), \gamma_{\mathsf{prnt}}(z, y)$

By our convention, $\textsc{Out}(\mathbf{x})$ is $\textsc{Ancstr}(x, y)$. This program returns the transitive closure of the relation obtained by applying the regex formula $\gamma_{\mathsf{prnt}}(x, z)$ from Example 2.

4 Comparison to Core Spanners

We begin the exploration of the expressive power of RGXlog by a comparison to the class of core spanners and the class of generalized core spanners. We first recall the following observation by Fagin et al. [8] for later reference.

▶ **Proposition 6** ([8]). *The class of spanners definable by non-recursive* RGXlog *is precisely the class of regular spanners, namely* $[\![\mathsf{RGX}^{\{\cup, \pi, \bowtie\}}]\!]$.

In addition to RGXlog being able to express union, projection and natural join, the following program shows that RGXlog can express the string-equality selection, namely $\zeta^=$.

- ▶ $\textsc{StrEq}(x, y) \leftarrow \langle x\{\epsilon\} \rangle, \langle y\{\epsilon\} \rangle$
- ▶ $\textsc{StrEq}(x, y) \leftarrow \langle x\{\sigma\tilde{x}\{.^*\}\} \rangle, \langle y\{\sigma\tilde{y}\{.^*\}\} \rangle, \textsc{StrEq}(\tilde{x}, \tilde{y})$

Here, the second rule is repeated for every alphabet letter σ. (Note that we are using the assumption that the alphabet is finite.) It thus follows that every core spanner is definable in RGXlog. The other direction is false. As an example, no core spanner extracts all spans x and y such that $\mathbf{s}_x$ is *not* a substring of $\mathbf{s}_y$ [7], or all pairs x and y of spans having the same *length* [8]. In the following example, we construct a RGXlog program that extracts both of these relationships.

▶ **Example 7.** In the following program, rules that involve σ and τ are repeated for all letters σ and τ such that $\sigma \neq \tau$, and the ones that involve only σ are repeated for every σ.

- ▶ $\textsc{Len}_=(x, y) \leftarrow \langle x\{\epsilon\} \rangle, \langle y\{\epsilon\} \rangle$
- ▶ $\textsc{Len}_=(x, y) \leftarrow \langle x\{.\tilde{x}\{.^*\}\} \rangle, \langle y\{.\tilde{y}\{.^*\}\} \rangle, \textsc{Len}_=(\tilde{x}, \tilde{y})$
- ▶ $\textsc{Len}_>(x, y) \leftarrow \langle x\{.^+\tilde{y}\{.^*\}\} \rangle, \textsc{Len}_=(\tilde{y}, y)$
- ▶ $\textsc{NotPrfx}(x, y) \leftarrow \langle x\{\sigma.^*\} \rangle, \langle y\{\epsilon\} \vee y\{\tau.^*\} \rangle$
- ▶ $\textsc{NotPrfx}(x, y) \leftarrow \langle x\{\sigma\tilde{x}\{.^*\}\} \rangle, \langle y\{\sigma\tilde{y}\{.^*\}\} \rangle, \textsc{NotPrfx}(\tilde{x}, \tilde{y})$
- ▶ $\textsc{NotCntd}(x, y) \leftarrow \textsc{Len}_>(x, y)$
- ▶ $\textsc{NotCntd}(x, y) \leftarrow \textsc{NotPrfx}(x, y), \langle y\{.\tilde{y}\{.^*\}\} \rangle, \textsc{NotCntd}(x, \tilde{y})$

The program defines the following relations:

- $\textsc{Len}_{=}(x, y)$ contains all spans x and y of the same length.
- $\textsc{Len}_{>}(x, y)$ contains all spans x and y such that x is longer than y.
- $\textsc{NotPrfx}(x, y)$ contains all spans x and y such that $\mathbf{s}_x$ is *not* a prefix of $\mathbf{s}_y$. The rules state that $\mathbf{s}_x$ is not a prefix of $\mathbf{s}_y$ if $\mathbf{s}_x$ is nonempty but $\mathbf{s}_y$ is empty, or the two begin with different letters, or the two begin with the same letter but the rest of $\mathbf{s}_x$ is not a prefix of the rest of $\mathbf{s}_y$.
- $\textsc{NotCntd}(x, y)$ contains all spans x and y such that $\mathbf{s}_x$ is *not* contained in $\mathbf{s}_y$. The rules state that this is the case if x is longer than y, or both of the following hold: $\mathbf{s}_x$ is not a prefix of $\mathbf{s}_y$, and $\mathbf{s}_x$ is not contained in the suffix of $\mathbf{s}_y$ following the first symbol.

In particular, the program defines both equal-length and non-containment relationships.

The impossibility proofs of Fagin et al. [7, 8] are based on the *core simplification lemma* [7], which states that every core spanner can be represented as a regular spanner, followed by a sequence of string-equality selections ($\zeta^{=}$) and projections (π). In turn, the proof of this lemma relies on the absence of the difference operator in the algebra. See Freydenberger and Holldack [10] for an indication of why a result similar to the core simplification lemma is not likely to hold in the presence of difference. Do things change when we consider *generalized core spanners*, where difference is allowed? To be precise, we are interested in two questions:

1. Can RGXlog express every generalized core spanner?
2. Is it true that every spanner definable in RGXlog is a generalized core spanner?

In the next section, we show that the answer to the first question is yes. In the remainder of this section, we show that the answer to the second question is no.

We begin by constructing the following RGXlog program, which defines a Boolean is spanner that returns **true** if and only if the length of the input $\mathbf{s}$ is a power of two.

- ▶ $\textsc{Pow2}(x) \leftarrow \langle x\{.\}\rangle$
- ▶ $\textsc{Pow2}(x) \leftarrow \langle x\{x_1\{.^*\}x_2\{.^*\}\}\rangle, \textsc{Pow2}(x_1), \textsc{Len}_{=}(x_1, x_2)$
- ▶ $\textsc{Out}() \leftarrow [x\{.^*\}], \textsc{Pow2}(x)$

We prove the following.

▶ **Theorem 8.** *There is no Boolean generalized core spanner that determines whether the length of the input string is a power of two.*

Hence, we get a negative answer to the second question. In the remainder of this section, we discuss the proof of Theorem 8. We need to prove that no generalized core spanner recognizes precisely all strings whose length is a power of two.

Let a be a letter, and L_{a} the language of all strings $\mathbf{s}$ that consist of 2^n occurrences of a for $n \geq 0$, that is: $L_{\mathsf{a}} \stackrel{\text{def}}{=} \{\mathbf{s} \in \mathsf{a}^* \mid |\mathbf{s}| \text{ is a power of } 2\}$. We will restrict our discussion to generalized core spanners that accept only strings in a^*, and show that no such spanner recognizes L_{a}. This is enough, since every generalized core spanner S can be restricted into a^* by joining S with the regex formula $[\mathsf{a}^*]$. For simplicity, we will further assume that our alphabet consists of only the symbol a. Then, a language L is identified by a set of natural numbers – the set of all numbers m such that $\mathsf{a}^m \in L$. We denote this set by $\mathbb{N}(L)$.

Presburger Arithmetic (PA) is the first-order theory of the natural numbers with the addition ($+$) binary function and the constants 0 and 1 [23]. For example, the relationship $x > y$ is expressible by the PA formula $\exists z[x = y + z + 1]$ and by the PA formula $x \neq y \wedge \exists z[x = y + z]$. As another example, the set of all even numbers x is definable by the PA formula $\exists y[x = y + y]$. When we say that a set A of natural numbers is *definable in PA* we mean that there is a unary PA formula $\varphi(x)$ such that $A = \{x \in \mathbb{N} \mid \varphi(x)\}$.

It is known that being a power of two is *not* definable in PA [17]. Theorem 8 then follows from the next theorem.

▶ **Theorem 9.** *A language $L \subseteq \{\mathsf{a}\}^*$ is recognizable by a Boolean generalized core spanner if and only if $\mathbb{N}(L)$ is definable in PA.*

5 Equivalence to Polynomial Time

While RGXlog programs output relations (which are sets of tuples), the result of evaluating a spanner on $\mathbf{s}$ is given as a set of $(V, \mathbf{s})$-records. Therefore, to compare the expressiveness of RGXlog programs and spanners, in what follows we implicitly treat tuples as records and vice-versa as described now. We assume that there is a fixed predefined order on Vars (e.g., the lexicographic order on the variables' names) and denote the i'th element in this order by v_i. A tuple $\mathbf{t} \in \mathsf{Spans}^n$ is viewed as the record whose domain is $\{v_1, \ldots, v_n\}$ that maps each v_i to $\mathbf{t}_i$; a $(V, \mathbf{s})$-record r is viewed as the tuple whose i'th element equals the value of r on v where v is the i'th variable of V according to the fixed predefined order on Vars.

An easy consequence of existing literature [11, 1] is that every RGXlog program can be evaluated in polynomial time (as usual, under data complexity). Indeed, the evaluation of a RGXlog program P can be done in two steps: *(1)* materialize the regex atoms on the input string $\mathbf{s}$ and get relations over spans, *and (2)* evaluate P as an ordinary Datalog program over an ordinary relational database, treating the regex formulas as the names of the corresponding materialized relations. The first step can be completed in polynomial time [11], and so can the second [1]. Quite remarkably, RGXlog programs capture *precisely* the spanners computable in polynomial time.

▶ **Theorem 10.** *A spanner is definable in* RGXlog *if and only if it is computable in polynomial time.*

In the remainder of this section, we discuss the proof of Theorem 10. The proof of the "only if" direction is described right before the theorem. To prove the "if" direction, we need some definitions and notation.

Definitions. We apply ordinary Datalog programs to databases over arbitrary domains, in contrast to RGXlog programs that we apply to strings, and that involve databases over the domain of spans. Formally, we define a Datalog program as a quadruple $(\mathcal{E}, \mathcal{I}, \Phi, \textsc{Out})$ where $\mathcal{E}$ and $\mathcal{I}$ are disjoint signatures referred to as the *EDB* (input) and *IDB* signatures, respectively, Out is a designated output relation symbol in $\mathcal{I}$, and Φ is a finite set of Datalog rules.[3] As usual, a *Datalog rule* has the form $\varphi \leftarrow \psi_1, \ldots, \psi_m$, where φ is an atomic formula over $\mathcal{I}$ and $\psi_1, \ldots, \psi_m$ are atomic formulas over $\mathcal{E}$ and $\mathcal{I}$. We again require each variable in the head φ to occur in the body $\psi_1, \ldots, \psi_m$. In this paper, we restrict Datalog programs to ones *without constants*; that is, an atomic formula ψ_i is of the form $R(x_1, \ldots, x_k)$ where R is a k-ary relation symbol and the x_i are (not necessarily distinct) variables. An input for a Datalog program Q is an instance D over $\mathcal{E}$ that instantiates every relation symbol of $\mathcal{E}$ with values from an arbitrary domain. The *active domain* of an instance D, denoted $adom(D)$, is the set of constants that occur in D.

[3] Note that unlike RGXlog, here there is no need to specify variables for Out. This is because a spanner evaluates to assignments of spans to variables, which we need to relate to Out, whereas a Datalog program evaluates to an entire relation, which is Out itself.

An *ordered signature* $\mathcal{E}$ is a signature that includes three distinguished relation symbols: a binary relation symbol Succ, and two unary relation symbols First and Last. An *ordered instance* D is an instance over an ordered signature $\mathcal{E}$ such that Succ is interpreted as a successor relation of some linear (total) order over $adom(D)$, and First and Last determine the first and last elements in this linear order, respectively.

A *semipositive* Datalog program P, or Datalog$^{\perp}$ program in notation, is a Datalog program in which the EDB atoms (i.e., atoms over EDB relation symbols) can be negated. We make the safety assumption that in each rule ρ, every variable that appears in the head of ρ is either (1) a variable appearing in a positive (i.e., non-negated) atom of the body of the rule, or (2) in $\mathsf{Vars}(\gamma)$ for a regex formula γ that appears in the body of the rule. For an instance D over $\mathcal{E}$, we denote by $P(D)$ the database with the signature $\mathcal{I}$ that results from applying P on D.

A *query* Q over a signature $\mathcal{E}$ is associated with a fixed arity $\mathsf{arity}(Q) = k$, and it maps an input database D over $\mathcal{E}$ into a relation $Q(D) \subseteq (adom(D))^k$. As usual, Q is *Boolean* if $k = 0$. We say that Q *respects isomorphism* if for all isomorphic databases D_1 and D_2 over $\mathcal{E}$, and for all isomorphisms $\varphi : adom(D_1) \to adom(D_2)$ between D_1 and D_2, it is the case that $\varphi(Q(D_1)) = Q(D_2)$.

Proof idea for Theorem 10. We now discuss the proof of the "if" direction. The proof is based on Papadimitriou's theorem [22], stating a close connection between semipositive Datalog and polynomial time:

▶ **Theorem 11** ([22, 6]). *Let $\mathcal{E}$ be an ordered signature and let Q be a query over $\mathcal{E}$ such that Q respects isomorphism. Then Q is computable in polynomial time if and only if Q is computable by a Datalog$^{\perp}$ program.*

Our proof continues as follows. Let S be a spanner that is computable in polynomial time. We translate S into a RGXlog program P in two main steps. In the first step, we translate S into a Datalog$^{\perp}$ program P_S by an application of Theorem 11. In the second step, we translate P_S into P. To realize the first step of the construction, we need to encode our input string by a database, since P_S operates over databases (and not over strings). To use Theorem 11, we need to make sure that this encoding is computable in polynomial time, and that it is invariant under isomorphism, that is, the encoding allows to restore the string even if replaced by an isomorphic database. To realize the second step of the construction, we need to bridge several differences between RGXlog and Datalog$^{\perp}$. First, the former takes as input a string, and the latter a database. Second, the latter assumes an ordered signature while the former does not involve any order. Third, the former does not allow negation while in the latter EDB atoms can be negated.

For the first step of our translation, we use a standard representation (which we shall explain shortly) of a string as a logical structure and extend it with a total order on its active domain. Note that we have to make sure that the active domain contains the output domain (i.e., all spans of the input string). We define $\mathcal{R}^{\mathsf{ord}}$ to be an ordered signature with the unary relation symbols R_σ for each $\sigma \in \Sigma$, in addition to the required Succ, First and Last. Let $\mathbf{s} = \sigma_1 \cdots \sigma_n$ be an input string. We define an instance $D_{\mathbf{s}}$ over $\mathcal{R}^{\mathsf{ord}}$ by materializing the relations as follows.

- Each relation R_σ consists of all tuples $([i, i+1\rangle)$ such that $\sigma_i = \sigma$.
- Succ consists of the pairs $([i, i'\rangle, [i, i'+1\rangle)$ and all pairs $([i, n+1\rangle, [i+1, i+1\rangle)$ whenever the involved spans are legal spans of $\mathbf{s}$.
- First and Last consist of $[1, 1\rangle$, and $[n+1, n+1\rangle$, respectively.

▶ **Comment 12.** Observe that we view the linear order as the lexicographic order over the spans. The only difference from the usual lexicographic order on ordered pairs (i, j) in that for spans, we must have $i \leq j$. The successor relation Succ is inferred from this order.

An *encoding instance* (or just *encoding*) D is an instance over $\mathcal{R}^{\mathsf{ord}}$ that is isomorphic to $D_{\mathbf{s}}$ for some string $\mathbf{s}$. In this case, we say that D *encodes* $\mathbf{s}$. Note that the entries of an encoding are not necessarily spans. Nevertheless, every encoding encodes a unique string. The following lemma is straightforward.

▶ **Lemma 13.** *Let D be an instance over $\mathcal{R}^{\mathsf{ord}}$. The following hold:*
1. *Whether D is an encoding can be determined in polynomial time.*
2. *If D is an encoding, then there are unique string $\mathbf{s}$ and isomorphism ι such that D encodes $\mathbf{s}$ and $\iota(D_{\mathbf{s}}) = D$; moreover, both $\mathbf{s}$ and ι are computable in polynomial time.*

Let S be a spanner. We define a query Q_S over $\mathcal{R}^{\mathsf{ord}}$ as follows. If the input database D is an encoding and $\mathbf{s}$ and ι are as in Lemma 13, then $Q_S(D) = \iota([\![S]\!](\mathbf{s}))$; otherwise, $Q_S(D)$ is empty. To apply Theorem 11, we make an observation.

▶ **Observation 14.** *The query Q_S respects isomorphism, and moreover, is computable in polynomial time whenever S is computable in polynomial time.*

We can now apply Theorem 11 on Q_S:

▶ **Lemma 15.** *If S is computable in polynomial time, then there exists a Datalog$^{\perp}$ program P' over $\mathcal{R}^{\mathsf{ord}}$ such that $P'(D) = Q_S(D)$ for every instance D over $\mathcal{R}^{\mathsf{ord}}$.*

The second step of the translation simulates the Datalog$^{\perp}$ program P' using a RGXlog program. With RGXlog, we can construct $D_{\mathbf{s}}$ from $\mathbf{s}$ with the following rules:

▶ $R_\sigma(x) \leftarrow \langle x\{\sigma\}\rangle$ ▶ $\textsc{Succ}(x_1, x_2) \leftarrow \langle x_2\{x_1\{.^*\}\ .\}\rangle \vee [.^* x_2\{\ .\ x_1\{\epsilon\}\ .^*\}]$
▶ $\textsc{First}(x) \leftarrow [x\{\epsilon\}.^*]$ ▶ $\textsc{Last}(x) \leftarrow [.^* x\{\epsilon\}]$

Indeed, if we evaluate the above RGXlog rules on a string $\mathbf{s}$, we get exactly $D_{\mathbf{s}}$. Note that rules in Datalog$^{\perp}$ that do not involve negation can be viewed as RGXlog rules. However, since RGXlog do not allow negation, we need to include the negated EDBs as additional EDBs. Nevertheless, we can negate EDBs without explicit negation, because regular spanners are closed under difference and complement [7]. We therefore conclude the following lemma.

▶ **Lemma 16.** *If P' is a Datalog$^{\perp}$ program over $\mathcal{R}^{\mathsf{ord}}$, then there exists a RGXlog program P such that $P(\mathbf{s}) = P'(D_{\mathbf{s}})$ for every string $\mathbf{s}$.*

To summarize the proof of the "if" direction of Theorem 10, let S be a spanner computable in polynomial time. We defined Q_S to be such that $Q_S(D_{\mathbf{s}}) = [\![S]\!](\mathbf{s})$ for all $\mathbf{s}$. Lemma 15 implies that there exists a Datalog$^{\perp}$ program P' such that $P'(D_{\mathbf{s}}) = Q_S(D_{\mathbf{s}})$ for all $\mathbf{s}$. By Lemma 16, there exists a RGXlog program P such that $P(\mathbf{s}) = P'(D_{\mathbf{s}})$ for all $\mathbf{s}$. Therefore, P is the required RGXlog program such that $P(\mathbf{s}) = S(\mathbf{s})$ for all $\mathbf{s}$.

5.1 RGXlog over Monadic Regex Formulas

Our proof of Theorem 10 showed that RGXlog programs over *binary* regex formulas (i.e., regex formulas with two variables) suffice to capture every spanner that is computable in polynomial time. Next, we show that if we allow only *monadic* regex formulas (i.e., regex formulas with one variable), then we strictly decrease the expressiveness. We call such programs *regex-monadic* programs. We can characterize the class of spanners expressible by regex-monadic programs, as follows.

▶ **Theorem 17.** *Let S be a spanner. The following are equivalent:*

1. *S is definable as a regex-monadic program.*
2. *S is definable as a* RGXlog *program where all the rules have the form*

$$\textsc{Out}(x_1, \ldots, x_k) \leftarrow \gamma_1(x_1), \ldots, \gamma_k(x_k), \gamma()$$

where each $\gamma_i(x_i)$ is a unary regex formula and γ is a Boolean regex formula.

Note that in the second part of Theorem 17, the Boolean $\gamma()$ can be omitted whenever $k > 0$, since $\gamma()$ can be compiled into $\gamma_k(x_k)$. To prove the theorem, we use a result by Levy et al. [18], stating that recursion does not add expressive power when every relation in the EDB is unary. This theorem implies that every spanner definable as a regex-monadic program is regular. We then draw the following direct consequence on Boolean programs.

▶ **Corollary 18.** *A language is accepted by a Boolean regex-monadic program if and only if it is regular.*

For non-Boolean spanners, we can use Theorem 17 to show that regex-monadic programs are *strictly less expressive* than regular spanners. For instance, we can show that the relation "the span x contains the span y" is not expressible as a regex-monadic program, although it is clearly regular. Therefore, we conclude the following.

▶ **Corollary 19.** *The class of regex-monadic programs is strictly less expressive than the class of regular spanners.*

6 Extension to a Combined Relational/Textual Model

In this section, we extend our main Theorem (Theorem 10) to *Spannerlog* – a data and query model introduced by Nahshon et al. [21] that unifies and generalizes relational databases and spanners by considering relations over both strings and spans.

6.1 Spannerlog

The fragment of Spannerlog that we consider is referred to by Nahshon et al. [21] as Spannerlog⟨RGX⟩, and we abbreviate it as simply Spl⟨RGX⟩. A *mixed signature* is a collection of *mixed relation symbols* R that have two types of attributes: *string* attributes and *span* attributes. We denote by $[R]_{\mathsf{str}}$ and $[R]_{\mathsf{spn}}$ the sets of string attributes and span attributes of R, respectively, where an attribute is represented by its corresponding index. Hence, $[R]_{\mathsf{str}}$ and $[R]_{\mathsf{spn}}$ are disjoint and $[R]_{\mathsf{str}} \cup [R]_{\mathsf{spn}} = \{1, \ldots, \mathsf{arity}(R)\}$. A *mixed relation* over R is a set of tuples $(a_1, \ldots, a_m)$ where m is the arity of R and each a_ℓ is a string in Σ^* if $\ell \in [R]_{\mathsf{str}}$ and a span $[i, j\rangle$ if $\ell \in [R]_{\mathsf{spn}}$. A *mixed instance* D over a mixed signature consists of a mixed relation R^D for each mixed relation symbol R. A *query* Q over a mixed signature $\mathcal{E}$ is associated with a mixed relation symbol R_Q, and it maps every mixed instance D over $\mathcal{E}$ into a mixed relation $Q(D)$ over R_Q.

A mixed signature whose attributes are all string attributes (in all of the mixed relation symbols) is called a *span-free signature*. A mixed relation over a relation symbol whose attributes are all string (respectively, span) attributes is called a *string relation* (respectively, *span relation*). To emphasize the difference between mixed signatures (respectively, mixed relation symbols, mixed relations) and the signatures that do not involve types (which we have dealt with up to this section), we often relate to the latter as *standard* signatures (respectively, standard relation symbols, standard relations).

GENEO:

Cain␣son␣of␣Adam,␣Abel␣son␣of␣Adam,␣Enoch␣son␣of␣Cain,␣Irad␣...
Obed␣son␣of␣Ruth,␣Obed␣son␣of␣Boaz,␣Jesse␣son␣of␣Obed,␣David␣...

Figure 2 The input for the program in Example 20.

We consider queries defined by $\mathsf{Spl}\langle\mathsf{RGX}\rangle$ programs, which are defined as follows. We assume two infinite and disjoint sets $\mathsf{Vars}_{\mathsf{str}}$ and $\mathsf{Vars}_{\mathsf{spn}}$ of *string variables* and *span variables*, respectively. To distinguish between the two, we mark a string variable with an overline (e.g., $\overline{x}$). By a *string term* we refer to an expression of the form $\overline{x}$ or $\overline{x}_y$, where $\overline{x}$ is a string variable and y is a span variable. In $\mathsf{Spl}\langle\mathsf{RGX}\rangle$, an *atom* over an m-ary relation symbol R is an expression of the form $R(\tau_1, \ldots, \tau_m)$ where τ_ℓ is a string term if $\ell \in [R]_{\mathsf{str}}$ or a span variable if $\ell \in [R]_{\mathsf{spn}}$. A *regex atom* is an expression of the form $\langle\tau\rangle[\gamma]$ where τ is a string term and γ is a regex formula. Unlike RGXlog, in which there is a single input string, in $\mathsf{Spl}\langle\mathsf{RGX}\rangle$ a regex atom $\langle\tau\rangle[\gamma]$ indicates that the input for γ is τ. We allow regex formulas to use only span variables. An $\mathsf{Spl}\langle\mathsf{RGX}\rangle$ *program* is a quadruple $\langle\mathcal{E}, \mathcal{I}, \Phi, \textsc{Out}\rangle$ where:

- $\mathcal{E}$ is a mixed signature referred to as the *EDB* signature;
- $\mathcal{I}$ is a mixed signature referred to as the *IDB* signature;
- Φ is a finite set of *rules* of the form $\varphi \leftarrow \psi_1, \ldots, \psi_m$ where φ is an atom over $\mathcal{I}$ and each ψ_i is an atom over $\mathcal{I}$, an atom over $\mathcal{E}$, or a regex atom;
- $\textsc{Out} \in \mathcal{I}$ is a designated *output* relation symbol.

We require the rules to be *safe* in the following sense: *(a)* every head variable occurs at least once in the body of the rule, *and (b)* every string variable $\overline{x}$ in the rule occurs, *as a string term*, in at least one relational atom (over $\mathcal{E}$ or $\mathcal{I}$) in the rule.

We extend $\mathsf{Spl}\langle\mathsf{RGX}\rangle$ with *stratified negation* in the usual way: the set of relation symbols in $\mathcal{E} \cup \mathcal{I}$ is partitioned into *strata* $\mathcal{I}_0, \mathcal{I}_1, \ldots, \mathcal{I}_m$ such that $\mathcal{I}_0 = \mathcal{E}$, the body of each rule contains only relation symbols from strata that precede or the same as that of the head, and negated atoms in the body are from strata that strictly precede that of the head. In this case, *safe* rules are those for which every head variable occurs at least once in a *positive* atom in the body of the rule and every string variable $\overline{x}$ in the rule occurs, as a string term, in at least one *positive* relational atom (over $\mathcal{E}$ or $\mathcal{I}$) in the rule.

The semantics of an $\mathsf{Spl}\langle\mathsf{RGX}\rangle$ program (with stratified negation) is similar to the semantics of RGXlog programs (with the standard interpretation of stratified negation in Datalog). Given a mixed instance D over $\mathcal{E}$, the $\mathsf{Spl}\langle\mathsf{RGX}\rangle$ program $P = \langle\mathcal{E}, \mathcal{I}, \Phi, \textsc{Out}\rangle$ computes the mixed instance $P(D)$ over $\mathcal{I}$ and emits the mixed relation $\textsc{Out}$ of $P(D)$. A query Q over $\mathcal{E}$ is *definable* in $\mathsf{Spl}\langle\mathsf{RGX}\rangle$ if there exists an $\mathsf{Spl}\langle\mathsf{RGX}\rangle$ program $P = \langle\mathcal{E}, \mathcal{I}, \Phi, \textsc{Out}\rangle$ such that $\textsc{Out}^{P(D)} = Q(D)$ for all mixed instances D over $\mathcal{E}$.

▶ **Example 20.** Following is an $\mathsf{Spl}\langle\mathsf{RGX}\rangle$ program over the mixed signature of the instance of Figure 2. As usual, $\textsc{Out}$ is the relation symbol in the head of the last rule, here $\textsc{NonRltv}$.

$$\textsc{Ancstr}(\bar{x}, y, \bar{x}, z) \leftarrow \textsc{Geneo}(\bar{x})\,,\ \langle\bar{x}\rangle\gamma_{\mathsf{prnt}}(y, z)$$
$$\textsc{Ancstr}(\bar{w}, y, \bar{x}, z) \leftarrow \textsc{Ancstr}(\bar{w}, y, \bar{v}, y')\,,\ \textsc{Geneo}(\bar{x})\,,\ \langle\bar{x}\rangle\gamma_{\mathsf{prnt}}(z', z)\,,\ \textsc{StrEq}(\bar{x}_{z'}, \bar{v}_{y'})$$
$$\textsc{Rltv}(\bar{w}, y, \bar{x}, z) \leftarrow \textsc{Ancstr}(\bar{v}, y', \bar{w}, y)\,,\ \textsc{Ancstr}(\bar{u}, z', \bar{x}, z,)\,,\ \textsc{StrEq}(\bar{v}_{y'}, \bar{u}_{z'})$$
$$\textsc{NonRltv}(\bar{w}, y, \bar{x}, z) \leftarrow \textsc{Geneo}(\bar{w})\,,\ \langle\bar{w}\rangle\gamma_{\mathsf{prsn}}(y)\,,\ \textsc{Geneo}(\bar{x})\,,\ \langle\bar{x}\rangle\gamma_{\mathsf{prsn}}(z),\ \neg\textsc{Rltv}(\bar{w}, y, \bar{x}, z)$$

The relation $\textsc{Geneo}$ in Figure 2 contains strings that describe (partial) family trees. We assume for simplicity that every name that occurs in such a string is a unique identifier. The regex formulas $\gamma_{\mathsf{prsn}}(x)$ and $\gamma_{\mathsf{prnt}}(y, z)$ are the same as $\gamma_{\mathsf{cap}}(x)$ and $\gamma_{\mathsf{prnt}}(y, z)$ defined in Example 2, respectively. The first two rules of the program extract the relation $\textsc{Ancstr}$

that has four attributes: the first and third are string attributes and the second and fourth are span attributes. The first (respectively, third) attribute is the "context" string of the second (respectively, fourth) span attribute. Observe the similarity to the corresponding definition in Example 5. Here, unlike Example 5, we need also to save the context string of each of the spans, and hence, we need two additional attributes. The second and third rules use the relation StrEq that holds pairs $(\bar{w}_y, \bar{x}_z)$ such that $\bar{w}_y$ and $\bar{x}_z$ are identical. This relation can be expressed in $\mathsf{Spl}\langle\mathsf{RGX}\rangle$ similarly to RGXlog, as described in Section 4.

After evaluating the program, the relation Ancstr holds tuples $(\bar{w}, y, \bar{x}, z)$ such that $\bar{w}_y$ is an ancestor of $\bar{x}_z$. The relation Rltv holds tuples $(\bar{w}, y, \bar{x}, z)$ such that according to the information stored in Geneo, $\bar{w}_y$ is a relative of $\bar{x}_z$ (i.e., they share a common ancestor). The relation NonRltv holds tuples $(\bar{w}, y, \bar{x}, z)$ such that $\bar{w}_y$ is *not* a relative of $\bar{x}_z$.

6.2 Equivalence to Polynomial Time

Let $\mathcal{E}$ be a span-free signature, and D an instance over $\mathcal{E}$. We define the *extended active domain* of D, in notation $adom^+(D)$, to be the union of the following two sets: *(a)* the set of all strings that appear in D, as well as all of their substrings; *and (b)* the set of all spans of strings of D.

Note that for every query Q definable as an $\mathsf{Spl}\langle\mathsf{RGX}\rangle$ program $P = \langle\mathcal{E}, \mathcal{I}, \Phi, \text{Out}\rangle$, and every input database D over $\mathcal{E}$, we have $adom(Q(D)) \subseteq adom^+(D)$, that is, every output string is a substring of some string in D, and every output span is a span of some string in D. Our result in this section states that, under this condition, we can express in $\mathsf{Spl}\langle\mathsf{RGX}\rangle$ with stratified negation every query Q computable in polynomial time.

▶ **Theorem 21.** *Let Q be a query over a span-free signature $\mathcal{E}$, with the property that $adom(Q(D)) \subseteq adom^+(D)$ for all instances D over $\mathcal{E}$. The following are equivalent:*

1. *Q is computable in polynomial time.*
2. *Q is computable in $\mathsf{Spl}\langle\mathsf{RGX}\rangle$ with stratified negation.*

We remark that Theorem 21 can be extended to general mixed signatures $\mathcal{E}$ if we assume that every span mentioned in the input database D is within the boundary of some string in D. We also remark that Theorem 21 is incorrect without negation, and this can be shown using standard arguments of monotonicity. In addition, since we use negation, in order to prevent ambiguity we use the stratified semantics.

Proof idea. We now discuss the proof idea of Theorem 21. The direction $2 \to 1$ is straightforward, so we discuss only the direction $1 \to 2$. Let Q be a query over a span-free signature $\mathcal{E}$, with the property that $adom(Q(D)) \subseteq adom^+(D)$ for all instances D over $\mathcal{E}$. Assume that Q is computable in polynomial time. We need to construct an $\mathsf{Spl}\langle\mathsf{RGX}\rangle$ program P with stratified negation for computing Q. We do so in two steps. In the first step, we apply Theorem 10 to get a (standard) Datalog$^\perp$ program P' that simulates Q. Yet, P' does not necessarily respect the *typing* conditions of $\mathsf{Spl}\langle\mathsf{RGX}\rangle$ with respect to the two types *string* and *span*. So, in the second step, we transform P' to an $\mathsf{Spl}\langle\mathsf{RGX}\rangle$ program P as desired. Next, we discuss each step in more detail.

First step. In order to produce the Datalog$^\perp$ program P', some adaptation is required to apply Theorem 10. First, we need to deal with the fact that the output of Q may include values that are not in the active domain of the input (namely, spans and substrings). Second, we need to establish a linear order over the active domain. Third, we need to assure that the query that Theorem 10 is applied on respects isomorphism. To solve the first problem, we

extend the input database D with relations that contain every substring and every span of every string in D. This can be done using $\mathsf{Spl}\langle\mathsf{RGX}\rangle$ rules with regex atoms. For the second problem, we construct a linear order over the domain of all substrings and spans of strings of D, again using $\mathsf{Spl}\langle\mathsf{RGX}\rangle$ rules. For this part, stratified negation is needed. For the third problem, we show how our extended input database allows us to restore D even if all values (strings and spans) are replaced with other values by applying an injective mapping.

Second step. In order to transform P' into a "legal" $\mathsf{Spl}\langle\mathsf{RGX}\rangle$ program P that obeys the typing of attributes and variables, we do the following. First, we replace every IDB relation symbol R with every possible *typed* version of R by assigning types to attributes. Semantically, we view the original R as the union of all of its typed versions. Second, we replace every rule with every typed version of the rule by replacing relation symbols with their typed versions. Third, we eliminate rules that treat one or more variable inconsistently, that is, the same variable is treated once as a string variable and once as a span variable. The following example demonstrates the steps described above:

▶ **Example 22.** Let us consider the Datalog$^{\perp}$ program that contains the rule $R(x,y) \leftarrow S(x), T(y,z)$. The relation atom $R(x,y)$ has four different typed versions, such as the following.

- $R_{\mathsf{str},\mathsf{str}}(\overline{x},\overline{y})$ wherein both attributes are string attributes.
- $R_{\mathsf{spn},\mathsf{str}}(x,\overline{y})$ wherein the first attribute is a span attribute and the second is a string attribute.

The rule $R(x,y) \leftarrow S(x), T(y,z)$ has 2^5 different typed versions, one for each "type assignment" for its variables, such as the following.

- $R_{\mathsf{str},\mathsf{str}}(\overline{x},\overline{y}) \leftarrow S_{\mathsf{str}}(\overline{x}), T_{\mathsf{str},\mathsf{str}}(\overline{y},\overline{z})$
- $R_{\mathsf{spn},\mathsf{str}}(x,\overline{y}) \leftarrow S_{\mathsf{str}}(\overline{x}), T_{\mathsf{str},\mathsf{str}}(\overline{y},\overline{z})$

Note that the second rule is type inconsistent due to the variable x that is regarded as a span variable in the head atom and as a string variable in the atom $S_{\mathsf{str}}(\overline{x})$, and thus it is eliminated.

Finally, we prove that this replacement preserves the semantics of the program.

7 Conclusions

We studied RGXlog, namely, Datalog over regex formulas. We proved that this language expresses precisely the spanners computable in polynomial time. RGXlog is more expressive than the previously studied language of core spanners and, as we showed here, more expressive than even the language of generalized core spanners. We also observed that it takes very simple binary regex formulas to capture the entire expressive power. Unary regex formulas, on the other hand, do not suffice: in the Boolean case, they recognize precisely the regular languages, and in the non-Boolean case, they produce a strict subset of the regular spanners. Finally, we extended the equivalence result to $\mathsf{Spl}\langle\mathsf{RGX}\rangle$ with stratified negation over mixed instances, a model that generalizes both the relational model and the document spanners.

The expressive power of RGXlog is somewhat mysterious, since we do not yet have a good understanding of how to phrase some simple polynomial-time programs *naturally* in RGXlog. The constructive proof simulates the corresponding polynomial-time Turing machine, and does not lend itself to program clarity. For instance, is there a natural program for computing the *complement* of the transitive closure of a binary relation encoded by the input? An interesting future work is to investigate this aspect by studying the complexity of translating simple formalisms, such as generalized core spanners, into RGXlog.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

2 Jitendra Ajmera, Hyung-Il Ahn, Meena Nagarajan, Ashish Verma, Danish Contractor, Stephen Dill, and Matthew Denesuk. A CRM system for social media: challenges and experiences. In *WWW*, pages 49–58. ACM, 2013.

3 Marcelo Arenas, Francisco Maturana, Cristian Riveros, and Domagoj Vrgoc. A framework for annotating CSV-like data. *PVLDB*, 9:876–887, 2016.

4 Edward Benson, Aria Haghighi, and Regina Barzilay. Event Discovery in Social Media Feeds. In *ACL*, pages 389–398. The Association for Computer Linguistics, 2011.

5 Pierre Boullier. From Contextual Grammars to Range Concatenation Grammars. *Electr. Notes Theor. Comput. Sci.*, 53:41–52, 2001.

6 Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.

7 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Document Spanners: A Formal Approach to Information Extraction. *J. ACM*, 62(2):12, 2015.

8 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Declarative Cleaning of Inconsistencies in Information Extraction. *ACM Trans. Database Syst.*, 41(1):6:1–6:44, 2016.

9 Dominik D. Freydenberger. A Logic for Document Spanners. In *ICDT*, volume 68 of *LIPIcs*, pages 13:1–13:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

10 Dominik D. Freydenberger and Mario Holldack. Document Spanners: From Expressive Power to Decision Problems. In *ICDT*, volume 48 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:17. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016.

11 Dominik D. Freydenberger, Benny Kimelfeld, and Liat Peterfreund. Joining Extractions of Regular Expressions. *CoRR*, abs/1703.10350, 2017. `arXiv:1703.10350`.

12 Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis. In *ICDM*, pages 149–158. IEEE Computer Society, 2009.

13 Georg Gottlob and Christoph Koch. Monadic Datalog and the Expressive Power of Languages for Web Information Extraction. *J. ACM*, 51(1):74–113, January 2004.

14 Alon Y. Halevy, Inderpal Singh Mumick, Yehoshua Sagiv, and Oded Shmueli. Static analysis in Datalog extensions. *J. ACM*, 48(5):971–1012, 2001.

15 Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia. *Artif. Intell.*, 194:28–61, 2013.

16 K. N. King. Alternating Multihead Finite Automata. *Theor. Comput. Sci.*, 61:149–174, 1988.

17 Christopher C. Leary. *A Friendly Introduction to Mathematical Logic*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1999.

18 Alon Y. Levy, Inderpal Singh Mumick, Yehoshua Sagiv, and Oded Shmueli. Equivalence, Query-Reachability, and Satisfiability in Datalog Extensions. In Catriel Beeri, editor, *PODS*, pages 109–122. ACM Press, 1993.

19 Roger Levy and Christopher D. Manning. Deep Dependencies from Context-Free Statistical Parsers: Correcting the Surface Dependency Approximation. In *ACL*, pages 327–334. ACL, 2004.

20 Yunyao Li, Frederick Reiss, and Laura Chiticariu. SystemT: A declarative information extraction system. In *ACL*, pages 109–114. ACL, 2011.

21 Yoav Nahshon, Liat Peterfreund, and Stijn Vansummeren. Incorporating information extraction in the relational database model. In *WebDB*, page 6. ACM, 2016.

22 Christos H. Papadimitriou. A note on the expressive power of Prolog. *Bulletin of the EATCS*, 26:21–22, 1985.

23 M. Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes Rendus du Premier Congrès des Mathématiciens des Pays Slaves*, pages 92–101, Warszawa, 1929.

24 Christopher De Sa, Alexander Ratner, Christopher Ré, Jaeho Shin, Feiran Wang, Sen Wu, and Ce Zhang. DeepDive: Declarative knowledge base construction. *SIGMOD Record*, 45(1):60–67, 2016.

25 Warren Shen, AnHai Doan, Jeffrey F. Naughton, and Raghu Ramakrishnan. Declarative Information Extraction Using Datalog with Embedded Extraction Predicates. In *VLDB*, pages 1033–1044, 2007.

26 Jaeho Shin, Sen Wu, Feiran Wang, Christopher De Sa, Ce Zhang, and Christopher Ré. Incremental Knowledge Base Construction Using DeepDive. *PVLDB*, 8(11):1310–1321, 2015.

27 Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. YAGO: A core of semantic knowledge. In *WWW*, pages 697–706. ACM, 2007.

28 Hua Xu, Shane P. Stenner, Son Doan, Kevin B. Johnson, Lemuel R. Waitman, and Joshua C. Denny. MedEx: a medication information extraction system for clinical narratives. *JAMIA*, 17(1):19–24, 2010. doi:10.1197/jamia.M3378.

29 Alexander Yates, Michele Banko, Matthew Broadhead, Michael J. Cafarella, Oren Etzioni, and Stephen Soderland. TextRunner: Open information extraction on the web. In *ACL-HLT*, pages 25–26. ACL, 2007.

30 Huaiyu Zhu, Sriram Raghavan, Shivakumar Vaithyanathan, and Alexander Löser. Navigating the intranet with high precision. In *WWW*, pages 491–500. ACM, 2007.

Parallel-Correctness and Parallel-Boundedness for Datalog Programs

Frank Neven
Hasselt University and transnational University of Limburg, The Netherlands

Thomas Schwentick
Dortmund University, Germany

Christopher Spinrath
Dortmund University, Germany

Brecht Vandevoort[1]
Hasselt University and transnational University of Limburg, The Netherlands

Abstract

Recently, Ketsman et al. started the investigation of the parallel evaluation of recursive queries in the Massively Parallel Communication (MPC) model. Among other things, it was shown that parallel-correctness and parallel-boundedness for general Datalog programs is undecidable, by a reduction from the undecidable containment problem for Datalog. Furthermore, economic policies were introduced as a means to specify data distribution in a recursive setting. In this paper, we extend the latter framework to account for more general distributed evaluation strategies in terms of communication policies. We then show that the undecidability of parallel-correctness runs deeper: it already holds for fragments of Datalog, e.g., monadic and frontier-guarded Datalog, with a decidable containment problem, under relatively simple evaluation strategies. These simple evaluation strategies are defined w.r.t. data-moving distribution constraints. We then investigate restrictions of economic policies that yield decidability. In particular, we show that parallel-correctness is 2EXPTIME-complete for monadic and frontier-guarded Datalog under hash-based economic policies. Next, we consider restrictions of data-moving constraints and show that parallel-correctness and parallel-boundedness are 2EXPTIME-complete for frontier-guarded Datalog. Interestingly, distributed evaluation no longer preserves the usual containment relationships between fragments of Datalog. Indeed, not every monadic Datalog program is equivalent to a frontier-guarded one in the distributed setting. We illustrate the latter by considering two alternative settings where in one of these parallel-correctness is decidable for frontier-guarded Datalog but undecidable for monadic Datalog.

2012 ACM Subject Classification Theory of computation → Database theory

Keywords and phrases Datalog, distributed databases, distributed evaluation, decision problems, complexity

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.14

1 Introduction

Modern data management systems like Spark [6, 30] and Hadoop [5] embrace massive parallelism to speed up query processing over large volumes of data. The latter inspired an extensive line of research on the foundations of parallel query evaluation which mostly focused on join queries (e.g., [2, 3, 10, 23, 24]). These investigations can be cast within the massively parallel communication model [10] (MPC) where query evaluation proceeds in multiple rounds and where each round consists of a computation and a communication step. In the computation step, every server operates on its local data in isolation, whereas in the communication step data is exchanged between the servers. The Hypercube algorithm [3, 17]

[1] PhD Fellow of the Research Foundation – Flanders (FWO)

which can compute any multiway join in one round was shown to play a fundamental role in several of the above mentioned papers. Ameloot et al. [4] introduced a framework to reason about such Hypercube-style single-round algorithms. The authors considered the problem of *parallel-correctness* asking whether one can always be sure that the corresponding single-round algorithm computes the query result correctly, no matter the actual data, starting from a particular distribution policy. Parallel-correctness and related problems were studied for conjunctive queries (with and without negation) and under set as well as bag semantics [4, 18, 22].

The relevance of Datalog as a query language for expressing recursive queries has only gained in importance in the last decade (e.g,[1, 7, 20, 28, 29]). Ketsman, Albarghouthi, and Koutris [22] started the investigation of the parallel evaluation of Datalog queries in the multi-round MPC model. To this end, *economic policies* were introduced as a means to specify data reshuffling in a recursive setting where intermediate derived facts can be communicated between servers. So, in addition to specifying the initial distribution of extensional database facts, economic policies also determine which servers can produce and consume intensional database facts during evaluation of a Datalog program. Among other things, the authors showed that parallel-correctness and parallel-boundedness (the problem that asks whether a Datalog program can be evaluated in a bounded number of evaluation rounds) for general Datalog programs is undecidable. For this, a reduction from the undecidable containment problem for Datalog was used.

In this paper, we revisit the parallel-correctness and parallel-boundedness problem for Datalog programs. We start by establishing that the undecidability of parallel-correctness runs deeper than the containment problem. We show that parallel-correctness is already undecidable under relatively simple distributed evaluation strategies for fragments of Datalog with a decidable containment problem: monadic and frontier-guarded Datalog. Here, monadic Datalog is the variant of Datalog where intensional predicates have arity at most one, whereas in frontier-guarded Datalog every rule should contain an extensional predicate mentioning all the variables occurring in the head of the rule (c.f., e.g., [8, 9, 13]).

With this undecidability result as a guideline, we focus in this paper on settings for which the above mentioned decision problems become decidable. We start by presenting a generic framework for the distributed evaluation of Datalog within the multi-round MPC model. This framework allows for more general evaluation strategies than the economic policies of [22]. In brief, for a given Datalog program P, its parallel evaluation is determined by a *policy pair* (δ, ρ) consisting of an initial distribution policy δ and a communication policy ρ. A database instance is first distributed according to δ assigning extensional database facts to the computation servers. Then the computation proceeds in rounds consisting of two steps: in the first step each server recursively evaluates the same program P over its local database, and in the second step all derived intensional database facts are communicated according to the communication policy ρ. In their most general form, communication policies are triples of the form $(\mathbf{f}, k, \ell)$ indicating that when a fact $\mathbf{f}$ is derived on server k it is sent to server ℓ.

Since in practice, data distribution is typically done using hash-partitioning (e.g.,[10, 25, 31]), we consider hash-based policies as initial distribution policies δ. For communication policies ρ we consider two approaches: economic hash-based policies and communication policies defined through data-moving constraints. The former is an instantiation of the framework of [22] with hash-partitioning while the latter is a declarative approach based on distribution constraints as introduced in [19], but used here in a much more restricted form to specify relocation of facts in between rounds of computation.

The contributions of this paper can be summarized as follows:

1. We introduce in Section 3 a general framework for the specification of distributed evaluation strategies for Datalog programs.
2. In Section 4 we first show that parallel-correctness for monadic Datalog and frontier-guarded Datalog is undecidable for communication policies defined by simple data-moving distribution constraints. We then study decidable cases for parallel-correctness and show that parallel-correctness is 2ExpTime-complete for monadic Datalog and frontier-guarded Datalog under hash-based communication policies, and that parallel-correctness is 2ExpTime-complete for frontier-guarded Datalog under communication policies defined by a restriction of data-moving distribution constraints.
3. The upper bounds in Section 4 rely on a fine-grained analysis of the complexity of the containment problem for frontier-guarded Datalog in terms of the number of variables, the number of rules and the size of the rules (Proposition 11). This analysis is based on a proof by Bourhis, Krötzsch, and Rudolph [13] and is carried out in Section 5.
4. We show in Section 6 that parallel-boundedness is 2ExpTime-complete for frontier-guarded Datalog and communication policies defined by the same kind of data-moving distribution constraints as for parallel-correctness.
5. In Section 7 we show that the results on parallel-correctness do not change if we restrict each server to derive only facts that it can communicate (similar to [21]), and that there is a family of communication policies for which parallel-correctness of frontier-guarded Datalog is decidable whereas parallel-correctness of monadic Datalog is undecidable.

The latter results in Section 7 show that the usual ability to transform programs from monadic Datalog into frontier-guarded Datalog can break drastically in a distributed setting, which we find quite surprising.

We are aware that 2ExpTime upper bounds are far from practical tractability. Still, this is the best we can hope for as containment of a Datalog fragment easily reduces to parallel-correctness for that fragment and containment for both monadic as well as frontier-guarded Datalog is complete for 2ExpTime [9, 11, 13].

Related work is addressed throughout the paper at the places where it is most relevant. Due to space limitations, many proofs are delegated to the full version of the paper which we plan to publish at arXiv by the time of the conference. We are grateful to Gaetano Geck for inspiring discussions on the subject of this paper.

2 Preliminaries

Datalog. We assume a fixed database *schema* $\mathcal{S}$ throughout the paper. An *instance* G is a finite set of facts over $\mathcal{S}$. We use standard notions, including $ar(R)$ for the arity of a relation name R, $R(\bar{x})$ for an atom (with variables in $\bar{x}$), $R(\bar{t})$ for a *fact* (with domain elements in $\bar{t}$) $\mathsf{val}(G)$ for the set of domain elements in G, and $I_{|S}$ for the set of facts in I over $S \subseteq \mathcal{S}$.

A *Datalog rule* τ is usually written[2] as $R(\bar{x}) \leftarrow S_1(\bar{y}_1), \ldots, S_n(\bar{y}_n)$. By $head_\tau$ and $body_\tau$ we refer to its head and body, respectively. By $vars(\tau)$ and $vars(A)$, we denote the set of all variables in τ and in an atom A, respectively. A *Datalog program* P is a finite set of Datalog rules. By $\textsc{edb}(P)$, $\textsc{idb}(P)$ and $out(P)$, we denote the extensional, intensional and output relation names of P, respectively. P is *monadic* if the arity of every relation in $\textsc{idb}(P)$ is at most one. It is *frontier-guarded*, if every rule τ is frontier-guarded, that is, if there is an $\textsc{edb}$-atom in $body_\tau$ that contains all the variables from $head_\tau$. We denote the class of monadic and frontier-guarded Datalog programs with mon-Datalog and fg-Datalog, respectively. The semantics of Datalog programs is defined in the usual way.

[2] We only consider Datalog programs without constant symbols.

▶ **Example 1.** Let $E_r(x, y)$ denote red edges and let $E_s(x, y)$ denote sea blue edges. The following monadic program P_{mon} computes all nodes reachable from a starting node x (indicated by $\text{Start}(x)$) by a path containing only red and a path containing only sea blue edges:

$$R(x) \leftarrow \text{Start}(x) \qquad S(x) \leftarrow \text{Start}(x) \qquad \text{Out}(x) \leftarrow R(x), S(x)$$
$$R(x) \leftarrow R(y), E_r(y, x) \qquad S(x) \leftarrow S(y), E_s(y, x)$$

Let $I_{\text{mon}} = \{\text{Start}(1), E_r(1, 3), E_r(1, 4), E_s(1, 2), E_s(2, 3)\}$, then $P(I_{\text{mon}})_{|out(P)} = \{\text{Out}(1), \text{Out}(3)\}$.

Networks and Distributed Databases. We model a *network* of database servers as a finite set $\mathcal{N}$ of *servers*. A distributed instance $D = (G, \mathbf{I})$ consists of a global instance G over $\mathcal{S}$ and a family $\mathbf{I} = (I_k)_{k\in\mathcal{N}}$ of local instances over $\mathcal{S}$, one for each server of $\mathcal{N}$ such that $\bigcup_{k\in\mathcal{N}} I_k = G$. We also denote G by $\text{global}(D)$. We write $\mathbf{f}@k$ for a fact $\mathbf{f}$ in I_k. For two distributed instances $D = (G, \mathbf{I})$ and $D' = (G', \mathbf{I}')$ we write $D \subseteq D'$, if $G \subseteq G'$ and $I_k \subseteq I'_k$ for every $k \in \mathcal{N}$. We emphasise that we do not allow facts to be "skipped". That is, every global fact should occur somewhere as a local fact.

3 Framework

We adapt the MPC model [10] and define a generic framework that allows us to reason about Datalog programs in distributed settings. The framework is based on *policy pairs* (δ, ρ), consisting of a (partial) specification of the initial data distribution δ and a communication policy ρ over the same network. Policy pairs allow to specify initial distributions and communication of facts independently and in different ways (thus extending the framework of [21]). We then define the hash-based initial distributions that we use and provide several concrete ways to define communication policies: hash-based and constraint-based. We also define the parallel-correctness problem.

As mentioned before, we reason about Datalog programs relative to policy pairs (δ, ρ), where δ is a distribution policy and ρ a communication policy. These terms are defined next. We require that the distribution of EDB-facts is only described by δ and that communication policies are restricted to IDB-facts.

In general, a *distribution policy* δ over a network $\mathcal{N}$ is a function mapping global instances G to distributed instances $D = (G, \mathbf{I})$ over $\mathcal{N}$. A distributed instance $D = (G, \mathbf{I})$ over $\mathcal{N}$ is *valid with respect to* a distribution policy δ over $\mathcal{N}$ if $\delta(G) \subseteq D$. Thus, we allow that D distributes facts to more servers than required by δ. This is because we do not understand δ as a specification of a communication round but rather as a constraint that is met at the beginning of the evaluation of a Datalog program. However, the definition ensures that the global instance of $\delta(G)$ is just G. We define the specific (hash-based) distribution policies used in this paper below in Section 3.2.

For a network $\mathcal{N}$ and a (global) database G, a *communicated fact* $(\mathbf{f}, k, \ell)$ consists of a fact $\mathbf{f} \in G$ and two servers k and ℓ of $\mathcal{N}$ and has the intended meaning that $\mathbf{f}$ is communicated from k to ℓ. A *communication policy* ρ for a Datalog program P over $\mathcal{N}$ is a monotone mapping that assigns to every distributed instance $D = (G, \mathbf{I})$ over $\mathcal{S} \cup \text{IDB}(P)$ and $\mathcal{N}$ a set of communicated facts. Here, monotonicity means that $\rho(D) \subseteq \rho(D')$ whenever $D \subseteq D'$. The communication policies studied in this paper are defined in Section 3.3.

3.1 Distributed Evaluation of Datalog Programs

A communication policy ρ induces a distributed multi-round evaluation strategy for P over $D = (G, \mathbf{I})$ as follows.[3] Each round consists of a computation and a communication phase. By $D^i = (G^i, \mathbf{I}^i)$ we denote the distributed instance after the i-th communication phase. The initial instance D^0 is just D. Then, for $i \geq 1$, the following phases are performed:

- *Computation:* Every server computes the local fixpoint of P over its local instance. That is, $D' = (G', \mathbf{J})$ where $G' = \bigcup_{k \in \mathcal{N}} J_k$ and, for each $k \in \mathcal{N}$, $J_k = P(I_k^{i-1})$.
- *Communication:* For each $(\mathbf{f}, k, \ell) \in \rho(D')$, $\mathbf{f}$ is copied from k to ℓ. That is, for each $\ell \in \mathcal{N}$, $I_\ell^i = J_\ell \cup \{\mathbf{f} \mid (\mathbf{f}, k, \ell) \in \rho(D'), \mathbf{f} \in J_k\}$. Then, $D^i = (G^i, \mathbf{I}^i)$ where $G^i = \bigcup_{k \in \mathcal{N}} I_k^i$.

The evaluation terminates when a global fixpoint is reached. We note that this fixpoint always exists. By $[P, \rho](D)$, we denote the union of all facts found at any server in this fixpoint. By $[P, \rho](D)_{|out(P)}$ we denote the output of the query computed by P on D according to ρ.

We note that our setting differs slightly from [21]. We provide more details in Section 7 and show that it does not have any influence on our results for parallel-correctness.

▶ **Example 2.** Consider P_{mon} and I_{mon} from Example 1. Let $D_{\text{mon}} = (I_{\text{mon}}, \mathbf{I})$ be defined over the network $\mathcal{N} = [1, 4]$ with four servers and let $I_1 = \{\text{Start}(1), E_r(1,3), E_r(1,4)\}$, $I_2 = \{\text{Start}(1), E_s(1,2), E_s(2,3)\}$ and $I_3 = I_4 = \emptyset$. That is, Start-facts are duplicated over server 1 and 2, while red edges are sent to server 1 and sea blue edges to server 2. Let ρ be the communication policy that maps a distributed database $D' = (G', \mathbf{I}')$ to the set of triples $\{(R(i), 1, f(i)) \mid R(i) \in I_1'\} \cup \{(S(i), 2, f(i)) \mid S(i) \in I_2'\}$ with f a function mapping each value i to $((i-1) \bmod 4) + 1$. So, when server 1 derives a fact $R(c)$, it is sent to server $f(c)$. A fact $S(c)$ derived on server 2 is also sent to server $f(c)$. The distributed computation then proceeds as follows. Only newly added IDB facts are shown in each round. Underlined facts are received through communication.

$I_1^1 = \{R(1), R(3), R(4), \underline{S(1)}\}$,	$I_2^1 = \{S(1), S(2), S(3)\}$,	$I_3^1 = \{\underline{R(3)}, \underline{S(3)}\}$,	$I_4^1 = \{\underline{R(4)}\}$
$I_1^2 = \{\text{Out}(1)\}$,	$I_2^2 = \emptyset$,	$I_3^2 = \{\text{Out}(3)\}$,	$I_4^2 = \emptyset$
$I_1^3 = \emptyset$,	$I_2^3 = \emptyset$,	$I_3^3 = \emptyset$,	$I_4^3 = \emptyset$

A global fixpoint is reached in the third iteration.

Now we can formally define parallel-correctness for Datalog programs and policy pairs.

▶ **Definition 3** (Parallel-Correctness). *Let P be a Datalog program and (δ, ρ) a policy pair, consisting of a distribution policy δ and a communication policy ρ for P over the same network. We call P* parallel-correct *w.r.t. (δ, ρ), if $[P, \rho](D)_{|out(P)} = P(global(D))_{|out(P)}$, for every distributed instance D that is valid with respect to δ. A program P is* parallel-correct w.r.t. a family $\mathcal{F}$ of policy pairs *if it is parallel-correct w.r.t. every policy pair in $\mathcal{F}$.*

For classes $\mathcal{P}$ of Datalog programs, and $\mathcal{C}$ of families of policy pairs, Para-Correct$(\mathcal{P}, \mathcal{C})$ denotes the decision problem that asks, for a program $P \in \mathcal{P}$ and a family $\mathcal{F} \in \mathcal{C}$, whether P is parallel-correct with respect to $\mathcal{F}$.

Since all our evaluation strategies are sound (i.e., $[P, \rho](D)_{|out(P)} \subseteq P(\text{global}(D))_{|out(P)}$), and communication strategies are monotone, parallel-correctness can be decided by testing parallel-completeness (i.e., $P(\text{global}(D))_{|out(P)} \subseteq [P, \rho](D)_{|out(P)}$).

[3] We recall that we do not view the distribution policy δ as a specification of a communication round.

3.2 Distribution Policies

As mentioned before, we only consider hash-based distribution policies for the specification of initial distrbutions.

Let $H = (h_1, \ldots, h_p)$ be a tuple of hash functions over some network $\mathcal{N}$, where, for each i, $h_i : U^{\alpha_i} \to 2^{\mathcal{N}} - \{\emptyset\}$, for some *arity* α_i and with U the set of domain elements. A *hash policy scheme* Z is a set of triples $(R, i, \bar{b})$ such that R is a relation symbol, i is a number and $\bar{b} \in [1, ar(R)]^n$ for some $n \geq 0$. We say that Z is *consistent* if for all triples $(R, i, \bar{b})$ and $(S, i, \bar{c})$ it holds $|\bar{b}| = |\bar{c}|$. We only consider consistent hash policy schemes. We say that Z is *compatible with* H, if $|\bar{b}| = \alpha_i$, for every triple $(R, i, \bar{b})$.

If Z is consistent and compatible with H, the two induce the distribution policy $\delta_{Z,H}$ that maps every fact $R(\bar{u})$ to the set of servers k, for which there is a triple $(R, i, \bar{b}) \in Z$ such that $k \in h_i(\bar{v})$, where $\bar{v} = \pi_{\bar{b}}(\bar{u})$ is the projection of $\bar{u}$ to $\bar{b}$. Formally, $\pi_{\bar{b}}(\bar{a})$ is $(a_{b_1}, \ldots, a_{b_{\alpha_i}})$. By $\mathcal{F}(Z)$ we denote the family of such distribution policies $\delta_{Z,H}$.

We note that $\delta_{Z,H}$ is *fact-based*, in the sense that it maps each fact over $\mathcal{S}$ to a set of servers from $\mathcal{N}$, independent of the other facts in G. Like [21] we only consider fact-based distribution policies for the specification of initial distributions.

▶ **Example 4.** Recall I_{mon} as defined in Example 1. Consider the hash policy scheme $Z = \{(\text{Start}, 1, ()), (\text{Start}, 2, ()), (E_r, 1, ()), (E_s, 2, ())\}$. Then Z is consistent and is compatible with any tuple $H = (h_1, h_2)$ of hash functions where both h_1 and h_2 are nullary. Let $\mathcal{N} = [1, 4]$. Furthermore, let h_1 map the empty tuple to 1 and let h_2 map the empty tuple to 2. Then $\delta_{Z,H}(I_{\text{mon}}) = D_{\text{mon}}$ where D_{mon} is as defined in Example 2. We refer to Example 5 and 6 below for more involved hash policy schemes.

3.3 Communication Policies

In this paper, we study two ways to specify communication policies. We first consider policies that are based on hash-based distribution policies and afterwards policies that are specified with the help of distribution constraints.

In [21] the communication policies that controlled the evaluation of Datalog programs, were called *economic policies*. An economic policy ρ is induced by a pair (π, γ) of fact-based distribution policies. Here, π is referred to as the *production policy* determining which servers are allowed to derive IDB-facts whereas γ is the *consumption policy* determining which servers can consume which IDB- and EDB-facts. Such a pair defines a communication policy in which each fact $\mathbf{f}$ induces the set $\{\mathbf{f}\} \times \pi(\mathbf{f}) \times \gamma(\mathbf{f})$ of communicated facts.

Hash-Based Communication. If Z_1, Z_2 are hash policy schemes such that $Z_1 \cup Z_2$ is consistent and compatible with a tuple H of hash functions, they induce a communication policy $\rho_{Z_1,Z_2,H}$ as $\rho_{\pi,\gamma}$ where $\pi = \delta_{Z_1,H}$ and $\gamma = \delta_{Z_2,H}$. So, hash-based communication policies are an instantiation of economic policies. If Z, Z_1, Z_2 are hash policy schemes such that $Z \cup Z_1 \cup Z_2$ is consistent[4], $\mathcal{F}(Z, Z_1, Z_2)$ denotes the set of policy pairs $(\delta_{Z,H}, \rho_{Z_1,Z_2,H})$, where H is any tuple of hash functions with which Z, Z_1, Z_2 are compatible. We denote by Hash-Hash the class of families that are defined in this way by triples Z, Z_1, Z_2 of hash policy schemes. We note that economic policies and hash-based communication policies are *fact-based* in the sense that they map $\mathcal{S}$-facts $\mathbf{f}$ to sets of communicated facts $(\mathbf{f}, k, \ell)$, independent of other facts.

[4] Further on, we will simply say that Z, Z_1, Z_2 are consistent.

▶ **Example 5.** Consider the hash policy scheme Z as described in Example 4. Let $Z_1 = \{(R, 1, ()), (S, 2, ())\}$ and $Z_2 = \{(R, 3, (1)), (S, 3, (1))\}$. Then, Z, Z_1, Z_2 are consistent. Consider now the tuple $H = (h_1, h_2, h_3)$ of hash functions, where h_1 and h_2 are as in Example 4, and h_3 is a unary hash function mapping each value i onto $((i-1) \bmod 4) + 1$. Clearly, Z, Z_1 and Z_2 are compatible with H, and $\rho_{Z_1,Z_2,H} = \rho$, where ρ is as defined in Example 2. For $\delta_{Z,H}$ as defined in Example 4, P_{mon} is parallel-correct w.r.t. $(\delta_{Z,H}, \rho_{Z_1,Z_2,H})$. Furthermore, this holds for every tuple H of hash functions with which Z, Z_1, Z_2 are compatible. Or equivalently, P_{mon} is parallel-correct w.r.t. $\mathcal{F}(Z, Z_1, Z_2)$.

Constraint-Based Communication. We next introduce a constraint-based formalism to define communication policies. We make use of the formalism of distribution constraints as introduced in [19]. It is important to note that the distribution constraints in [19] are far more general and are targeted at specifying classes of distributions (including co-partitionings). They are in particular based on distributed tuple- and equality-generating dependencies. Here, we use the formalism of distribution constraints to define the communication of facts between servers and only consider so-called data-moving distribution constraints. In the terminology of [19] they are data-collecting and do not refer to the global database.

A *distributed atom* $A@\kappa$ consists of an atom A and a server variable κ. A *distribution constraint* is a rule of the form $\sigma = \mathcal{A} \to K$ for a set of distributed atoms $\mathcal{A}$ forming its *body* and a distributed atom K forming its *head*. We denote $head_\sigma = K$ and $body_\sigma = \mathcal{A}$, respectively. We further assume in this paper that $body_\sigma$ contains a distributed atom of the form $B@\kappa$ when $head_\sigma = A@\kappa$, that is, using the same server variable κ as in the head. We denote by $vars(\sigma)$ and $nvars(\sigma)$ the set of data and server variables occurring in σ.

A distribution constraint is *data-moving* when the atom occurring in the distributed atom in the head also occurs in a distributed atom in the body.[5] That is, when its head equals $A@\kappa$ then its body contains a distributed atom of the form $A@\lambda$ (possibly $\kappa = \lambda$). This will then imply that the atom A will be sent from server λ to server κ.

A *valuation* for σ over a distributed instance D with network $\mathcal{N}$, is a mapping V which maps server variables to servers, and the other variables to domain values. For a distributed atom $A' = A@\kappa$, we write $V(A') \in D$ if $V(A) \in I_{V(\kappa)}$. Each set Σ of data-moving distribution constraints over $\mathcal{S}$ induces for every network $\mathcal{N}$ a communication policy $\rho_{\Sigma,\mathcal{N}}$ as follows: for each distributed instance D over $\mathcal{N}$, $(\mathbf{f}, k, \ell)$ is in $\rho_{\Sigma,\mathcal{N}}(D)$ if there is a valuation V and a constraint $\sigma \in \Sigma$ such that $V(body_\sigma) \subseteq D$, $V(head_\sigma) = \mathbf{f}@\ell$ and $\mathbf{f}@k \in V(body_\sigma)$.

Let Σ be a set of data-moving distribution constraints and let Z be a hash policy. By $\mathcal{F}(Z, \Sigma)$ we denote the set of all policy pairs $(\delta_{Z,H}, \rho_{\Sigma,\mathcal{N}})$, where H is a tuple of hash functions over $\mathcal{N}$ and Z is compatible with H. By Hash-Constraints we denote the class of families $\mathcal{F}(Z, \Sigma)$, where Z is a hash policy and Σ is a set of data-moving distribution constraints.

▶ **Example 6.** Consider the following set Σ of data-moving distribution constraints:

$$R(y)@\kappa, E_r(y, x)@\lambda \to R(y)@\lambda, \quad S(y)@\kappa, E_s(y, x)@\lambda \to S(y)@\lambda.$$

Intuitively, for an arbitrary network $\mathcal{N}$, the induced communication policy $\rho_{\Sigma,\mathcal{N}}$ communicates each fact $R(y)$ to every server containing a matching fact $E_r(y, x)$. Analogously, $S(y)$ is sent to every server containing at least one $E_s(y, x)$.

[5] While data-moving constraints are similar to VWL of [1], it is not the case that WL can express *all* distribution constraints: node-generating constraints can not be defined (c.f., [19]).

Next, consider the hash policy scheme $Z = \{(\text{Start}, 1, (1)), (E_r, 1, (2)), (E_s, 1, (2))\}$, and let P_{mon} be as in Example 1. Then P_{mon} is parallel-correct w.r.t. every $(\delta_{Z,H}, \rho_{\Sigma,\mathcal{N}})$ where $\mathcal{N}$ is a network and H is a tuple consisting of a single unary hash function h_1 over $\mathcal{N}$ with which Z is compatible. In other words, P_{mon} is parallel-correct w.r.t. $\mathcal{F}(Z, \Sigma)$.

Notice in particular that Σ does not have a constraint enforcing $R(c)$ and $S(c)$ to end up on the same server, as this already follows from Z and P_{mon}. Indeed, every derivation of $R(c)$ is witnessed by either $\text{Start}(c)$ or some $E_r(b, c)$, and every derivation of $S(c)$ is witnessed by $\text{Start}(c)$ or some $E_s(b, c)$. By construction of Z, these facts are always distributed onto the same set of servers $h_1(b)$. As a result, every $R(c)$ and corresponding $S(c)$ are always derived on the same set of servers $h_1(c)$.

4 Parallel-Correctness

In this section, we first show that for the fragments mon-Datalog and fg-Datalog with a decidable containment problem, parallel-correctness is undecidable for relatively simple distribution policies. Then we study settings where deciding parallel-correctness becomes decidable for mon-Datalog and fg-Datalog.

The following result sharpens the undecidability result for parallel-correctness for Datalog in [22], since it states undecidability for fragments of Datalog for which the containment problem is decidable.

▶ **Theorem 7.** Para-Correct *(fg-Datalog, Hash-Constraints) and* Para-Correct *(mon-Datalog, Hash-Constraints) are undecidable.*

The proof is by reduction from the halting problem of deterministic Minsky machines that is well-known to be undecidable (cf., e.g., [27]). The reduction yields a monadic *and* frontier-guarded Datalog program and therefore serves for both stated results.

We now turn to restrictions with decidable parallel-correctness. All our upper bounds rely on the fact that to decide parallel-correctness for all allowed distributed instances and communication policies it suffices to study distributed instances (and communication policies) that distribute facts in a maximally scattered way.

We first make this notion of scatteredness more precise and then present our results on parallel-correctness for Hash-Hash and Hash-MConstraints (a fragment of Hash-Constraints to be defined below) in the two subsequent subsections.

We say that a tuple $H = (h_1, \ldots, h_p)$ of hash functions *scatters* a global instance G if,

- $h_i(\bar{a}) \cap h_j(\bar{b}) = \emptyset$, for all $i \neq j$ and all tuples $\bar{a} \in \mathsf{val}(G)^{\alpha_i}$, $\bar{b} \in \mathsf{val}(G)^{\alpha_j}$, and
- $h_i(\bar{a}) \cap h_i(\bar{b}) = \emptyset$, for all tuples $\bar{a}, \bar{b} \in \mathsf{val}(G)^{\alpha_i}$ with $\bar{a} \neq \bar{b}$.

A distribution policy $\delta_{Z,H}$ *scatters* G if H scatters G. Thus, if $\delta_{Z,H}$ scatters G then two facts $R(\bar{a})$ and $R'(\bar{a}')$ meet at some server, if and only if there is some i and triples $(R, i, \bar{b})$ and $(R', i, \bar{b}')$ such that $\pi_{\bar{b}}(\bar{a}) = \pi_{\bar{b}'}(\bar{a}')$. Similarly, a pair $(\delta_{Z,H}, \rho_{Z_1,Z_2,H})$ *scatters* G if H scatters G. For a global instance G and a hash policy scheme Z, we define the canonical scattered instance $\delta_{Z,H}(G)$ where for each $i \leq p$ and $\bar{a} \in \mathsf{val}(G)^{\alpha_i}$, $h_i(\bar{a}) = \{(i, \bar{a})\}$.

4.1 Parallel-Correctness for Hash-Hash

The main result of this section is the following:

▶ **Theorem 8.** Para-Correct *(fg-Datalog, Hash-Hash) and* Para-Correct *(mon-Datalog, Hash-Hash) are* 2ExpTime*-complete.*

The lower bound is, in both cases, by a reduction from the containment problem for monadic Datalog, which is known to be 2ExpTime-hard [11].

Towards the upper bound, the following lemma shows that it suffices to restrict attention to scattering policies to establish parallel-correctness.

▶ **Lemma 9.** *Let P be a Datalog program and Z, Z_1, Z_2 consistent hash policy schemes. Then P is parallel-correct w.r.t. $\mathcal{F}(Z, Z_1, Z_2)$ if and only if for all $(\delta, \rho) \in \mathcal{F}(Z, Z_1, Z_2)$ and all global instances G that are scattered by (δ, ρ), it holds that $[P, \rho](\delta(G))_{|out(P)} = P(G)_{|out(P)}$.*

Next we show that over scattered instances the distributed evaluation of mon-Datalog and fg-Datalog programs can be simulated by fg-Datalog programs over the global instance.

▶ **Lemma 10.** *For every frontier-guarded or monadic Datalog program P and family $\mathcal{F}(Z, Z_1, Z_2)$ of hash-based policy pairs, a frontier-guarded datalog program P' can be constructed such that for every pair (δ, ρ) from $\mathcal{F}$ and for every global instance G, for which (δ, ρ) scatters G, it holds $P'(G)_{|\text{IDB}(P)} = [P, \rho](\delta(G))_{|\text{IDB}(P)}$. Furthermore, the number of variables and the length of the rules in P' is polynomial in the size of P, Z, Z_1 and Z_2 and the number of rules of P' is at most exponential.*

Proof. We first construct a Datalog program P'' that has the claimed equivalence property and afterwards we "guard" it. The program P'' uses one relation symbol R_i of arity $\alpha_i + ar(R)$ for every relation symbol R from P and every $i \leq p$ occurring in the hash policy schemes. Furthermore it has four kinds of rules:

- For every triple $(R, i, \bar{b})$ in Z, P_1 has a rule $R_i(\bar{v}; \bar{x}) \leftarrow R(\bar{x})$, where $\bar{x}$ is a tuple of mutually different variables, and $\bar{v} = \pi_{\bar{b}}(\bar{x})$ is the projection of $\bar{x}$ to $\bar{b}$.
- For each rule $R(\bar{x}) \leftarrow S^1(\bar{y}_1), \ldots, S^n(\bar{y}_n)$ of P there is an indexed version $R_i(\bar{v}; \bar{x}) \leftarrow S_i^1(\bar{v}; \bar{y}_1), \ldots, S_i^n(\bar{v}; \bar{y}_n)$, where $\bar{v}$ is a tuple of pairwise distinct variables.
- For every triple $(R, i, \bar{b})$ in Z_1 and every triple $(R, j, \bar{c})$ in Z_2 there is a rule $R_i(\bar{v}; \bar{x}) \leftarrow R_j(\bar{w}; \bar{x})$, where $\bar{x}$ is a tuple of mutually different variables, $\bar{v}$ is the projection of $\bar{x}$ to $\bar{b}$, and $\bar{w}$ is the projection of $\bar{x}$ to $\bar{c}$.
- Finally, for each IDB-relation of P and every i, there is a rule $R(\bar{x}) \leftarrow R_i(\bar{v}; \bar{x})$, where $\bar{v}$ and $\bar{x}$ are (disjoint) tuples of mutually different variables.

Let (δ, ρ) and G be as in the statement of the lemma. We argue that $P''(G)_{|\text{IDB}(P)} = [P, \rho](\delta(G))_{|\text{IDB}(P)}$. First of all, the rules of the first kind basically define $\delta(G)$, where a fact $R_i(\bar{c}, \bar{a})$ corresponds to $R(\bar{a})$ being at server $(i, \bar{c})$. The second set of rules mimics the local evaluation of Datalog rules at each such server. The third set of rules accounts for the communication of IDB-facts according to ρ. Finally, the last set of rules produces the output tuples. We show in the full paper how P'' can be guarded. ◀

The two lemmas show that testing for parallel-correctness reduces to testing of equivalence of fg-Datalog programs over global databases. Notice that we need both reductions in the previous lemma, as we show in Section 7 that monadic Datalog can not always be rewritten into an equivalent frontier-guarded Datalog program in the distributed setting.

As a final step, we need the following result, which already follows from the construction in [13, Theorem 7] but has a refined statement about the complexity with respect to the number of program rules. A proof sketch can be found in Section 5.

▶ **Proposition 11** ([13]). *The containment problem for Datalog programs (on the left-hand side) and frontier-guarded Datalog programs (on the right-hand side) is decidable in time*

- *doubly exponential in the number of variables in P and P',*
- *doubly exponential in the maximal size of a rule of P and P', and*
- *singly exponential in the number of rules of P and P'.*

Now we are ready to give a proof sketch for the upper bound in Theorem 8. The proof of the lower bound is given in the full version of this paper.

Proof of Theorem 8, upper bound. Let P be a mon-Datalog or fg-Datalog program and Z, Z_1, Z_2 consistent hash policy schemes. Let P' be the fg-Datalog program that can be constructed thanks to Lemma 10. By combing Lemma 10 with Lemma 9 it follows that P is parallel-correct for $\mathcal{F}(Z, Z_1, Z_2)$ if and only if P and P' are equivalent.

The containment problem (and therefore the equivalence problem) for frontier-guarded Datalog is known to be in 2ExpTime [13]. Since the size of P' is at most exponential in the size of P, Z, and Σ, and both programs are frontier-guarded, a triply exponential upper bound follows. However, only the number of rules of P' might be non-polynomial in P, Z, Z_1, Z_2, but the number of variables and the size of each rule is polynomial. Therefore, by Proposition 11 the desired 2ExpTime upper bound follows. ◀

4.2 Parallel-Correctness for Hash-MConstraints

Given the undecidability of Para-Correct(fg-Datalog, Hash-Constraints), we study in this subsection a restriction of distribution constraints that yields decidable parallel-correctness for fg-Datalog with the same complexity as Para-Correct(fg-Datalog, Hash-Hash).

The restriction has two ingredients. First, we require that to test whether a fact $\mathbf{f}$ should be communicated from a server to some other server, no other data values than those in $\mathbf{f}$ need to be communicated. In particular, for constraints, in which the body has only two node variables λ and κ, for the sending and the receiving server, we want to enable λ and κ to test "their" atoms independently, only communicating $\mathbf{f}$. To this end, we say that a data-moving distribution constraint σ has *guarded communication* if whenever its body contains atoms $A_1@\kappa_1$ and $A_2@\kappa_2$, with $\kappa_1 \neq \kappa_2$, then $vars(A_1) \cap vars(A_2) \subseteq vars(head_\sigma)$. The second ingredient of our restriction is a kind of linearity condition for the proof trees that result from computations. A set Σ of data-moving distribution constraints is *modest* if all its constraints have guarded communication and for every $\sigma \in \Sigma$, there is exactly[6] one atom in the body with a relation symbol that occurs in the head of some constraint of Σ. By Hash-MConstraints we denote the class of families $\mathcal{F}(Z, \Sigma)$, where Z is a hash policy and Σ is a modest set of data-moving distribution constraints. The main result of this subsection is the following theorem.

▶ **Theorem 12.** Para-Correct*(fg-Datalog, Hash-MConstraints) is* 2ExpTime*-complete.*

The lower bound is by a reduction from the containment problem for frontier-guarded Datalog, which is known to be 2ExpTime-hard (cf. [9, 13]).

The approach for the upper bound is very similar to that for Theorem 8. Let P be a Datalog program, $D = (G, \mathbf{I})$ be a distributed database over network $\mathcal{N}$, and Σ a set of data-moving distribution constraints. We use proof trees for facts that are produced in a distributed evaluation in the straightforward way by combining proof trees for Datalog facts with proof trees for communicated facts, whose communication is specified by some tgd. For a tree t, we denote its set of nodes by $nodes(t)$, its root by $root(t)$, and its set of leaves by $fringe(t)$. We denote the set of children of a node v by $children_t(v)$. Moreover, for a set $\mathbf{F}$ of facts and a server k, we denote by $\mathbf{F}@k$ the set $\{\mathbf{f}@k \mid \mathbf{f} \in \mathbf{F}\}$.

[6] Clearly, this is the atom $A@\lambda$ required by the definition of *data-moving*.

▶ **Definition 13.** *A* proof tree *t for a distributed fact $\mathbf{f}@k$ with respect to a Datalog program P and a set Σ of data-moving constraints is a tree, in which every node v is labelled by a distributed fact*[7] *$fact_t(v)$, and which has the following properties:*

- *$fact_t(root(t)) = \mathbf{f}@k$;*
- *for every inner node v labelled $\mathbf{g}@\ell$, there is either*
  - *a rule $\tau \in P$ and a valuation V for τ such that $V(head_\tau) = \mathbf{g}$ and $V(body_\tau)@\ell = \{fact_t(w) \mid w \in children_t(v)\}$, or* *(computed facts)*
  - *a constraint $\sigma \in \Sigma$ and a valuation V for σ such that $V(head_\sigma) = \mathbf{g}@\ell$ and $V(body_\sigma) = \{fact_t(w) \mid w \in children_t(v)\}$.* *(communicated facts)*

We call such a proof tree *computation-free*, if all its inner nodes are witnessed by constraints from Σ.

▶ **Definition 14.** *A class $\mathcal{P}$ of Datalog programs and a class $\mathcal{C}$ of distribution constraints have the* polynomial communication property *if there is a polynomial q such that the following holds, for each program $P \in \mathcal{P}$ and each finite set $\Sigma \subseteq \mathcal{C}$: if a distributed fact has a computation-free proof tree with respect to P and Σ, and with a set $\mathbf{F}$ of leaf facts, then there is such a proof tree of size at most $q(|P|, |\Sigma|)$ (with leaf facts from $\mathbf{F}$).*

▶ **Lemma 15.** *The class of Datalog programs and the class of modest distribution constraints have the polynomial communication property.*

The following lemma shows that it suffices to restrict attention to scattering policies to establish parallel-correctness.

▶ **Lemma 16.** *Let P be a Datalog program, Z be a hash policy scheme, and Σ be a set of data-moving distribution constraints. Then P is parallel-correct w.r.t. $\mathcal{F}(Z, \Sigma)$ if and only if for all $(\delta, \rho) \in \mathcal{F}(Z, \Sigma)$ and all global instances G that are scattered by δ, it holds that $[P, \rho](\delta(G))_{|out(P)} = P(G)_{|out(P)}$.*

The last step in the upper bound proof is the following lemma, similar to Lemma 10, showing that over scattered instances, a fg-Datalog program over the global instance can simulate distributed evaluation. As before, parallel-correctness can thus be reduced to testing equivalence of fg-Datalog programs.

▶ **Lemma 17.** *For every frontier-guarded Datalog program P, hash policy scheme Z, and set Σ of data-moving distribution constraints, which has the polynomial communication property, a frontier-guarded Datalog program P' can be constructed such that for every $(\delta, \rho) \in \mathcal{F}(Z, \Sigma)$ and every global instance G that is scattered by δ, it holds $[P, \rho](\delta(G))_{|out(P)} = P'(G)_{|out(P)}$. Furthermore, the number of variables and the length of the rules in P' is polynomial in the size of P, Z, and Σ and the number of rules of P' is at most exponential.*

Proof sketch. Similarly as in the proof of Lemma 10, we first construct an intermediate program P'' which is actually a sub-program of the one constructed there. However, P'' does not have any rules for the communication phases, therefore the second step takes care of communication *and* guarding.

As in the proof of Lemma 10, the program P'' uses one relation symbol R_i of arity $\alpha_i + ar(R)$, for every relation symbol R from P and every $i \leq p$ occurring in the hash policy scheme Z. Furthermore, it has frontier-guarded versions of three kinds of rules:

[7] We omit the subscript t, if it is clear from the context.

- For every triple $(R, i, \bar{b})$ in Z, P_1 has a rule $R_i(\bar{v}; \bar{x}) \leftarrow R(\bar{x})$, where $\bar{x}$ is a tuple of mutually different variables, and $\bar{v}$ is the projection of $\bar{x}$ to $\bar{b}$.
- For each rule $R(\bar{x}) \leftarrow S^1(\bar{y}_1), \ldots, S^n(\bar{y}_n)$ of P, there is an indexed version $R_i(\bar{v}, \bar{x}) \leftarrow S_i^1(\bar{v}, \bar{y}_1), \ldots, S_i^n(\bar{v}, \bar{y}_n)$, where $\bar{v}$ is a tuple of pairwise distinct variables.
- For each rule $R(\bar{x}) \leftarrow S^1(\bar{y}_1), \ldots, S^n(\bar{y}_n)$ of P, there is a second version $R(\bar{x}) \leftarrow S_i^1(\bar{v}, \bar{y}_1), \ldots, S_i^n(\bar{v}, \bar{y}_n)$.

As before, the rules of the first kind basically define the canonical scattered instance of G with respect to Z, where a fact $R_i(\bar{c}, \bar{a})$ corresponds to $R(\bar{a})$ being at server $(i, \bar{c})$, and the second set of rules mimics the local evaluation of Datalog rules at each such server. The rules of the third kind are responsible for the production of output facts. They can be guarded similarly as in Lemma 10.

However, these rules do not account yet for the redistribution of facts during the communication phases. In a nutshell, we replace IDB-atoms by sets of leaf (IDB- and EDB-) atoms of computation-free subtrees of a proof tree. Thanks to the polynomial communication property these sets only have polynomial size. The rules of P' result from P'' by replacing, in all possible ways, *some* IDB-atoms in bodies by such (polynomial-size) sets of atoms accounting for communication. ◀

5 The Containment Problem for Frontier-Guarded Datalog

In this section we prove Proposition 11 which states an upper bound for the complexity of the containment problem for frontier-guarded Datalog. In principle, it was shown in [13, Theorem 7], but we need that the upper bound is at most singly exponential in the number of rules of the two programs.

Each Datalog program P induces a (possibly infinite) set $C(P)$ of conjunctive queries (CQs) in a natural way by finite "unravellings" of P. It is well-known that for two Datalog programs P, P' it holds $P \subseteq P'$ if and only if[8] each CQ from $C(P)$ is contained in some CQ from $C(P')$ [26], which in turn is equivalent to the existence of a homomorphism from every CQ in $C(P')$ to some CQ from $C(P)$ [14]. The idea in [13], going back to [16], is to use tree automata to test the existence of such homomorphisms. This approach requires to encode CQs by trees over a finite alphabet, even though the number of variables in CQs from $C(P)$ and $C(P')$ can be unlimited. The idea is to encode CQs by symbolic trees in which different, unconnected occurrences of the same variable can represent different variables in a CQ.

The main step of the proof constructs from two fg-Datalog programs P, P' an alternating two-way tree automaton $\mathcal{A}_{P \sqsubseteq P'}$ for the set $T_{P \sqsubseteq P'}$ of symbolic proof trees representing CQs resulting from P that are contained in CQs resulting from P'. In the following, we define symbolic proof trees and, afterwards, we give a high-level description of $\mathcal{A}_{P \sqsubseteq P'}$ in terms of a 2-player game on symbolic proof trees t (for P) and consider its size.

▶ **Definition 18.** *A* symbolic proof tree *for a Datalog program P is a tree in which every node v is labelled by a rule instantiation $head_v \leftarrow body_v$ of a rule of P induced by a variable substitution $s : vars \rightharpoonup vars$, and which has the following properties.*[9]

- *For every node v of t, $body_{v|\mathrm{IDB}(P)} = \{head_w \mid w \in children_t(v)\}$ where $body_{v|\mathrm{IDB}(P)}$ is the set of all the IDB-atoms in $body_v$.*
- *All variables that occur in $body_v$ but not in $head_v$, do* not *occur in the label of the parent of v.*

We note that, in particular, for every leaf v of t, $body_v$ only contains EDB-atoms.

[8] The result is only stated for finite unions in [26] but the proof is just the same for countable unions.
[9] This definition does not yet yield a finite alphabet, but this issue will be addressed below.

With each symbolic proof tree t we associate a CQ $q[t]$ which is the CQ obtained from t in the obvious way, but replacing variable x in t in each x-connected component by a different fresh variable. With a set T of symbolic proof trees we associate a (possibly infinite) set $q[T]$ of CQs via $q[T] = \{q[t] \mid t \in T\}$. Thanks to [15] we can assume that in every symbolic proof tree t of every Datalog program P with n variables only variables among $x_1, \ldots, x_{2n}$ occur. For a Datalog program P we denote by T_P the set of symbolic proof trees t for P over set $\{x_1, \ldots, x_{2n}\}$ of variables (which we fix for each program P). T_P is a tree language over a (finite) alphabet. More precisely, the alphabet size is at most polynomial in the number of rules of P and exponential in the number of variables occurring in P (i.e. the number of substitutions).

To test containment of P in P' it now suffices to test whether $T_P \subseteq T_{P \sqsubseteq P'}$, where the tree language $T_{P \sqsubseteq P'}$ is defined as

$$T_{P \sqsubseteq P'} = \{t \in T_P \mid \text{ there is a tree } t' \in T_{P'} \text{ s.t. } q[t] \subseteq q[t']\}.$$

We show the following result, which is a slight refinement of a result in [13] in which the complexity in terms of the number of rules in P is better suited for our purposes.

▶ **Proposition 19.** *For each Datalog program P and frontier-guarded Datalog program P' one can construct a two-way alternating tree automaton $\mathcal{A}_{P \sqsubseteq P'}$ for $T_{P \sqsubseteq P'}$ of size*

- *exponential in the number of variables in P and P',*
- *exponential in the maximal size of a rule of P and P', and*
- *polynomial in the number of rules of P and P'.*

To decide containment of P in P' one can construct a non-deterministic automaton for the complement of $T_{P \sqsubseteq P'}$ from $\mathcal{A}_{P \sqsubseteq P'}$ in size exponential in the size of $\mathcal{A}_{P \sqsubseteq P'}$. Then P is contained in P' if and only if the intersection (of the language) of the resulting automaton with an automaton for T_P is empty. Altogether this proves Proposition 11.

Our proof of Proposition 19 uses a direct construction of the automaton $\mathcal{A}_{P \sqsubseteq P'}$ constructed in [13] for fg-Datalog, avoiding some technical complications in [13] caused by the more general *Guarded Queries*. However, we describe $\mathcal{A}_{P \sqsubseteq P'}$ only in terms of a 2-player game on symbolic proof trees t for P. The players are Arthur and Morgana. Morgana's task is to show that t is in $T_{P \sqsubseteq P'}$ and if she succeeds, she wins. Arthur's task is to challenge her proof. We only need to consider the game for trees from T_P since other trees are not relevant for the further processing. For that purpose, Morgana builds up a tree $t' \in T_{P'}$ and a (partial) homomorphism. During the game, both players can traverse the tree t in a two-way fashion but t' is only "traversed" top-down along a single path.

After each round, there is a current node v and a pair (h, M) where h is a partial mapping $h : \mathsf{var}(\tau') \to \{x_1, \ldots, x_{2|\mathsf{var}(P)|}\}$ and $M \subseteq \mathit{body}_{\tau'}$, for some rule instantiation τ' of P'. Intuitively, M consists of the atoms of $q[t']$, induced by τ', that still have to be mapped into $q[t]$ and h is a partial homomorphism that has to be extended by Morgana to do so.

In the first round, Morgana chooses a rule τ' of P' and a (partial) mapping h which maps the head of τ' to the head of root_t (i.e. the head of $q[t]$). Thus, $M = \mathit{body}_{\tau'}$. In all further rounds, there are three possible cases.

Case 1 $|M| > 1$: If there is a subset $M' \subsetneq M$ such that $\mathsf{var}(M') \cap \mathsf{var}(M - M') \subseteq \mathsf{dom}(h)$, i.e. if h is defined for all variables occurring in M' and its complement, Arthur chooses M' or $M - M'$ and the new state is (h', M') or $(h', M - M')$ where h' is the restriction to $\mathsf{var}(M')$ or $\mathsf{var}(M - M')$, respectively. Otherwise, Morgana has to extend the mapping

h to another variable x in M as follows. Morgana has to choose a (possibly new) current node v' of the input tree t such that all nodes on the path from v to v' contain all variables[10] of $h(M)$. Then she has to choose a variable appearing at v' for $h(x)$.

Case 2 $|M| = 1$ and $\mathsf{var}(M) \not\subseteq \mathsf{dom}(h)$: Morgana needs to extend the mapping h to all variables in the remaining atom $B \in M$. To this end, she has to choose a node v' such that all nodes on the path from v to v' contain all variables of $h(B)$. Then she has to extend h such that it maps all remaining variables of M to variables at v'.

Case 3 $|M| = 1$ and $\mathsf{var}(M) \subseteq \mathsf{dom}(h)$: Morgana chooses a node v' such that all nodes on the path from v to v' contain all variables of $h(B)$ where B is the remaining atom in M.

Case 3.1 B is extensional: Morgana wins if the image $h(B)$ is in the label of v', otherwise she loses.

Case 3.2 B is intensional: Morgana has to choose a rule instantiation $\tau' \in P'$ whose head equals B and a guard atom C of τ'. Then she extends, if necessary, h to all variables occurring in C (but not in B). If $h(C)$ is *not* at v', Morgana loses. Otherwise, the game continues at v' with the homomorphism h' that is the restriction of h to C and the atom set $body_{\tau'}$.

Due to lack of space we omit the formal definition of $\mathcal{A}_{P \sqsubseteq P'}$ and even the formal definition of alternating two-way tree automata. However, it should be clear that the game can be translated into such an automaton whose states basically consist of pairs (h, M) where $M \subseteq body_{\tau'}$ for a rule instantiation τ' of P' and a partial mapping $h : \mathsf{var}(\tau') \to \{x_1, \ldots, x_{2|\mathsf{var}(P)|}\}$. In particular, the number of states is at most exponential in the number of variables of P and P' (i.e. the number of partial mappings h), exponential in the size of the rules in P', and polynomial in the number of rules of P' (choices for M).

6 Parallel-Boundedness

In this section, we study parallel boundedness of Datalog programs in our distributed setting.

▶ **Definition 20** (Parallel-Boundedness). *A Datalog program P is* parallel-bounded *w.r.t. a family $\mathcal{F}$ of policy pairs if there is an $r \in \mathbb{N}$ such that for all policy pairs $(\delta, \rho) \in \mathcal{F}$ and all distributed instances D valid w.r.t. δ, no new* output facts *are computed on any server after r rounds in the distributed evaluation of P on D w.r.t. ρ.*

We note that our definition of parallel-boundedness is more generous than the one in [21], since there the requirement is that no facts whatsoever are produced any more. Of course, complexity upper bounds for our notion translate to the notion of [21].

For classes $\mathcal{P}$ of Datalog programs, and $\mathcal{C}$ of families of policy pairs, Para-Bound$(\mathcal{P}, \mathcal{C})$ denotes the decision problem that asks, for a program $P \in \mathcal{P}$ and a family $\mathcal{F} \in \mathcal{C}$, whether P is parallel-bounded *and* parallel-correct with respect to $\mathcal{F}$. Parallel-boundedness for programs that are not parallel-correct does not seem meaningful.

Our objective in this section is to decide parallel-boundedness for frontier-guarded Datalog programs and families of policy pairs in Hash-MConstraints.

▶ **Theorem 21.** Para-Bound(*fg-Datalog*, *Hash-MConstraints*) *is* 2ExpTime*-complete.*

Similarly to the lower bound proof for parallel-correctness (cf. Theorem 12), the lower bound is by a reduction from the containment problem for monadic Datalog, which is known to be 2ExpTime-hard [11].

[10] Recall that non-connected occurrences of a variable in t correspond to different variables in $q[t]$.

Towards the upper bound, let in the following P be a fg-Datalog program, Z a hash policy scheme and Σ a set of data-moving distribution constraints, such that P and Σ have the polynomial communication property. The starting point for the upper bound is the rewriting of P, Z, Σ into a fg-Datalog program P' from Lemma 17 which simulates the distributed evaluation of P on scattered instances. We show first that the consideration of scattered instances suffices also for parallel-boundedness.

▶ **Lemma 22.** *Let P be a Datalog program, Z a hash policy scheme, and Σ be a set of data-moving distribution constraints such that P is parallel-correct w.r.t. $\mathcal{F}(Z, \Sigma)$. Then P is* parallel-bounded *w.r.t. $\mathcal{F}(Z, \Sigma)$ if and only if there is an $r \in \mathbb{N}$ such that for all policy pairs $(\delta_S, \rho_S) \in \mathcal{F}$ and all distributed instances D_S scattered by δ_S, no new* output facts *are computed on any server after r rounds in the distributed evaluation of P on D_S w.r.t. ρ_S.*

The program P' has two kinds of rules: *communication-prone* rules that incorporate communication steps and *communication-free* rules. The body of each communication-prone rule consists of the leaves of some partial proof tree. With each body atom A, the number of communication steps in which A is sent in this proof tree can be computed when the rule is generated and symbolic proof trees corresponding to P' can be extended to carry these numbers as weights.

Then P is parallel-bounded w.r.t. $\mathcal{F}(Z, \Sigma)$ if and only if there is some constant r such that for each symbolic proof tree $t \in T_P$, there is a tree $t' \in T_{P'}$ such that $q[t] \subseteq q[t']$ and along each (root-to-leaf) path of t' the sum of the communication weights is at most r. Thus, testing parallel-boundedness asks for an extension of the approach of Section 5 which constructed an alternating automaton $\mathcal{A}_{P \sqsubseteq P'}$ for the set of trees $t \in T_P$ that are "captured" by trees $t' \in T_{P'}$. Now we are asking for a tree t' that respects the path bound. Since r is not known in advance, the "communication count" can not be incorporated into the states of $\mathcal{A}_{P \sqsubseteq P'}$. However, there is an extension of tree automata that was invented for exactly this kind of situation: cost automata. For our purposes, we need a cost automaton that has one counter which can be incremented but neither be decremented nor reset to zero. A cost automaton $\mathcal{A}$ is *limited* if there is some number s such that in each run of $\mathcal{A}$ the counter stays below s.

▶ **Proposition 23.** *For each frontier-guarded Datalog program P, hash policy scheme Z, and set Σ of modest distribution constraints such that P is parallel-correct w.r.t. $\mathcal{F}(Z, \Sigma)$, one can construct an alternating two-way tree cost automaton $\mathcal{A}$ that uses only a single counter, which is never reset, such that $\mathcal{A}$ is limited if and only if P is parallel-bounded w.r.t. $\mathcal{F}(Z, \Sigma)$. Furthermore, $\mathcal{A}$ has size (and can be constructed in time) exponential in P.*

The final step is then to decide whether $\mathcal{A}$ is limited which is decidable in exponential time thanks to the following result by [12]. In particular, Propositions 23 and 24 imply the upper bound of Theorem 21.

▶ **Proposition 24** ([12])**.** *The limitedness problem for alternating two-way tree cost automata with a single counter that is never reset, is decidable in exponential time.*

Proof idea for Proposition 23. Let P' be the program constructed in Lemma 17 for P, Z, and Σ. Similarly as in Section 5, we describe the cost automaton $\mathcal{A}$ in terms of a 2-player game on symbolic proof trees t for P'.

The rules of the game are almost exactly as in the game underlying $\mathcal{A}_{P' \sqsubseteq P'}$. We assign a value to each play, which is the sum of the weights of the IDB-atoms of communication-prone rules in t' that are visited in that play.[11] Morgana's task is to minimize the value of the

[11] We recall that a play can move arbitrarily in t but evolves along a path of t'.

play, while Arthur's task is to maximize it. Correspondingly, in the semantics of $\mathcal{A}$ the value for t is maximized over Arthur's moves and minimized over Morgana's moves. There is one difference to the rules for $\mathcal{A}_{P' \sqsubseteq P'}$: in situations where Morgana would lose immediately, Arthur is allowed to increment the counter arbitrarily. Altogether, the constructed cost automaton $\mathcal{A}$ is limited if and only if P is parallel-bounded.

The size of $\mathcal{A}$ is the same as for $\mathcal{A}_{P \sqsubseteq P'}$, i.e. polynomial in the number of rules of P' and at most exponential in the number of variables of P' and the maximal length of a rule in P'. Thus, it has size exponential in the original program P (cf. Lemma 17). ◀

7 Variants of the Framework

The precise formalisation of our framework leaves a lot of freedom and it is a fair question how "stable" our results are. To shed some light on this question, we consider in this section two slightly different settings for distributed evaluation strategies for Datalog. The answers that we get are not entirely unambiguous.

We call the first alternative the *locally restrained* setting. It was actually used in [21]. It requires that during each local fixpoint computation, every server only derives facts that it can communicate to other servers according to the communication policy.[12] It turns out that our results on parallel-correctness are not affected by this difference in the setting.

▶ **Proposition 25.**

(a) *In the locally restrained setting,* Para-Correct*(fg-Datalog, Hash-Hash) and* Para-Correct*(mon-Datalog, Hash-Hash) are in* 2ExpTime*.*

(b) *In the locally restrained setting,* Para-Correct*(fg-Datalog, Hash-MConstraints) is in* 2ExpTime*.*

The second variant attempts to restrict communication by disallowing servers to send facts that they did not derive themselves through a local computation. We refer to this variant as the *non-transitive communication* setting.

Interestingly, although this variant does not alter our result for frontier-guarded Datalog, it has a huge effect on parallel-correctness for monadic Datalog.

▶ **Proposition 26.**

(a) *In the non-transitive communication setting* Para-Correct*(fg-Datalog, Hash-Constraints) is in* 2ExpTime*.*

(b) *In the non-transitive communication setting* Para-Correct*(mon-Datalog, Hash-Constraints) is undecidable.*

In a nutshell, the difference between frontier-guarded Datalog and monadic Datalog in the non-transitive communication setting can be explained as follows. The reduction in the proof of Proposition 26b (stemming from the proof of Theorem 7) simulates a Minsky machine by repeated redistribution of facts. Although non-transitive communication only allows to send facts that were derived locally, for mon-Datalog this can be overcome by copy rules. Although copy rules are also allowed in fg-Datalog, they need to be guarded by EDB-atoms there, and this might change the semantics in a distributed setting.

In particular this setting shows that in a parallel setting it can happen that the usual "semantical inclusion" of monadic Datalog in frontier-guarded Datalog by rewriting programs

[12] Although it might seem useless to restrict local computation, this setting allows a more fine-grained control over the workload for each server if for example all servers start with the complete database.

of the former kind into programs of the latter kind fails to hold and monadic Datalog can indeed behave worse than frontier-guarded Datalog. The next example illustrates why this rewriting does not always work in a distributed setting.

▶ **Example 27.** Every monadic Datalog program can be rewritten into an equivalent (w.r.t. evaluation over a global database) frontier-guarded one by adding suitable guards [13]. Consider the program P_{mon} from Example 1. Then the rule $\text{Out}(x) \leftarrow R(x), S(x)$ is not frontier-guarded but can be replaced by the following set of frontier-guarded rules:

$$\text{Out}(x) \leftarrow R(x), S(x), \text{Start}(x) \qquad \text{Out}(x) \leftarrow R(x), S(x), E_s(x, y)$$
$$\text{Out}(x) \leftarrow R(x), S(x), E_r(x, y) \qquad \text{Out}(x) \leftarrow R(x), S(x), E_s(y, x)$$
$$\text{Out}(x) \leftarrow R(x), S(x), E_r(y, x)$$

We denote the such obtained program by P'_{mon}. Then for every global database G, $P_{\text{mon}}(G) = P'_{\text{mon}}(G)$, but $[P_{\text{mon}}, \rho](D_{\text{mon}}) \neq [P'_{\text{mon}}, \rho](D_{\text{mon}})$ for D_{mon} and ρ as defined in Example 2. Indeed, I_3 can never output $\text{Out}(3)$ as that server contains none of the required EDB-facts. The difference between the global and distributed evaluation is that in the former *all* guards are present in the global database while this does not necessarily hold true for all servers in the distributed evaluation.

8 Conclusion

While most effort has been devoted to study correctness properties in the simpler setting of the single-round MPC model (e.g., [4, 18, 22]), to date only Ketsman, Albarghouthi, and Koutris [21] considered correctness w.r.t. the multi-round MPC model (for Datalog). We continue in their footsteps by presenting a more general framework in terms of communication policies for the distributed evaluation of Datalog. We provide decidable instances for the parallel-correctness and parallel-boundedness problem for Datalog. Even though the obtained complexities are far from tractable, we do feel that they contribute to not only the study of reasoning for parallel Datalog but also for parallel queries in the multi-round MPC model in general. Indeed, while a most natural direction for future work is to investigate more tractable settings for Datalog, we are eager to explore communication policies to study the evaluation of non-recursive queries, like join queries, in the multi-round setting. In addition, we plan to investigate data-moving distribution constraints further, trying to characterize the class of distributed evaluations that extensions and restrictions of those can define. We also would like to explore how they can be efficiently evaluated, which is not obvious as such constraints can refer to multiple servers.

An important insight resulting from our investigations is the central role scattered instances play w.r.t. parallel-correctness and parallel-boundedness, and how they can be encoded into Datalog programs if they are defined through hash partitioning. It allowed to translate our reasoning problems for the distributed setting to static analysis problems for traditional Datalog, a topic that has received quite some attention in the literature.

References

1 Serge Abiteboul, Meghyn Bienvenu, Alban Galland, and Émilien Antoine. A rule-based language for web data management. In *Principles of Database Systems*, pages 293–304, 2011.

2 Foto N. Afrati, Manas R. Joglekar, Christopher Ré, Semih Salihoglu, and Jeffrey D. Ullman. GYM: A multiround distributed join algorithm. In *International Conference on Database Theory, ICDT 2017*, pages 4:1–4:18, 2017.

3 Foto N. Afrati and Jeffrey D. Ullman. Optimizing Multiway Joins in a Map-Reduce Environment. *IEEE Transactions on Knowledge and Data Engineering*, 23(9):1282–1298, 2011.

4 Tom J. Ameloot, Gaetano Geck, Bas Ketsman, Frank Neven, and Thomas Schwentick. Parallel-Correctness and Transferability for Conjunctive Queries. *Journal of the ACM*, 64(5):36:1–36:38, 2017. doi:10.1145/3106412.

5 Apache Hadoop. https://hadoop.apache.org/.

6 Apache Spark. https://spark.apache.org/.

7 Molham Aref, Balder ten Cate, Todd J. Green, Benny Kimelfeld, Dan Olteanu, Emir Pasalic, Todd L. Veldhuizen, and Geoffrey Washburn. Design and Implementation of the LogicBlox System. In *ACM SIGMOD International Conference on Management of Data, SIGMOD 2015*, pages 1371–1382, 2015.

8 Vince Bárány, Balder ten Cate, and Martin Otto. Queries with Guarded Negation. *Proceedings of the Very Large Database Endowment*, 5(11):1328–1339, 2012.

9 Vince Bárány, Balder ten Cate, and Luc Segoufin. Guarded Negation. In *International Colloquium on Automata, Languages, and Programming, ICALP 2011*, pages 356–367, 2011.

10 Paul Beame, Paraschos Koutris, and Dan Suciu. Communication Steps for Parallel Query Processing. *Journal of the ACM*, 64(6):40:1–40:58, 2017.

11 Michael Benedikt, Pierre Bourhis, and Pierre Senellart. Monadic Datalog Containment. In *International Colloquium on Automata, Languages, and Programming, ICALP 2012*, pages 79–91, 2012.

12 Michael Benedikt, Balder Ten Cate, Thomas Colcombet, and Michael Vanden Boom. The complexity of boundedness for guarded logics. In *Logic in Computer Science, LICS 2015*, pages 293–304, 2015.

13 Pierre Bourhis, Markus Krötzsch, and Sebastian Rudolph. Reasonable Highly Expressive Query Languages. In *International Joint Conference on Artificial Intelligence, IJCAI 2015*, pages 2826–2832, 2015.

14 Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, pages 77–90, 1977.

15 Surajit Chaudhuri and Moshe Y Vardi. On the equivalence of recursive and nonrecursive datalog programs. *Journal of Computer and System Sciences*, 54(1):61–78, 1997.

16 Stavros S. Cosmadakis, Haim Gaifman, Paris C. Kanellakis, and Moshe Y. Vardi. Decidable Optimization Problems for Database Logic Programs (Preliminary Report). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 477–490, 1988. doi:10.1145/62212.62259.

17 Sumit Ganguly, Avi Silberschatz, and Shalom Tsur. A Framework for the Parallel Processing of Datalog Queries. In *ACM SIGMOD International Conference on Management of Data, SIGMOD 1990*, pages 143–152, 1990.

18 Gaetano Geck, Bas Ketsman, Frank Neven, and Thomas Schwentick. Parallel-Correctness and Containment for Conjunctive Queries with Union and Negation. In *International Conference on Database Theory, ICDT 2016*, pages 9:1–9:17, 2016.

19 Gaetano Geck, Frank Neven, and Thomas Schwentick. Distribution Constraints: a Declarative Framework for Reasoning about Data Distributions. Manuscript, 2018.

20 Daniel Halperin, Victor Teixeira de Almeida, Lee Lee Choo, Shumo Chu, Paraschos Koutris, Dominik Moritz, Jennifer Ortiz, Vaspol Ruamviboonsuk, Jingjing Wang, Andrew Whitaker, Shengliang Xu, Magdalena Balazinska, Bill Howe, and Dan Suciu. Demonstration of the Myria Big Data Management Service. In *ACM SIGMOD International Conference on Management of Data, SIGMOD 2014*, pages 881–884, 2014.

21 Bas Ketsman, Aws Albarghouthi, and Paraschos Koutris. Distribution Policies for Datalog. In *International Conference on Database Theory, ICDT 2018*, pages 17:1–17:22, 2018.

22 Bas Ketsman, Frank Neven, and Brecht Vandevoort. Parallel-Correctness and Transferability for Conjunctive Queries under Bag Semantics. In *International Conference on Database Theory, ICDT 2018*, pages 18:1–18:16, 2018.

23 Bas Ketsman and Dan Suciu. A Worst-Case Optimal Multi-Round Algorithm for Parallel Computation of Conjunctive Queries. In *Principles of Database Systems, PODS 2017*, pages 417–428, 2017.

24 Paraschos Koutris, Paul Beame, and Dan Suciu. Worst-Case Optimal Algorithms for Parallel Query Processing. In *International Conference on Database Theory, ICDT 2016*, pages 8:1–8:18, 2016.

25 M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems, Third Edition*. Springer, 2011.

26 Yehoshua Sagiv and Mihalis Yannakakis. Equivalences Among Relational Expressions with the Union and Difference Operators. *J. ACM*, 27(4):633–655, 1980. doi:10.1145/322217.322221.

27 Mark V. Sapir. Minsky Machines and Algorithmic Problems. In *Fields of Logic and Computation II - Essays Dedicated to Yuri Gurevich on the Occasion of His 75th Birthday*, volume 9300 of *Lecture Notes in Computer Science*, pages 273–292. Springer, 2015.

28 Alexander Shkapsky, Mohan Yang, Matteo Interlandi, Hsuan Chiu, Tyson Condie, and Carlo Zaniolo. Big Data Analytics with Datalog Queries on Spark. In *ACM SIGMOD International Conference on Management of Data, SIGMOD 2016*, pages 1135–1149, 2016.

29 Jingjing Wang, Magdalena Balazinska, and Daniel Halperin. Asynchronous and Fault-Tolerant Recursive Datalog Evaluation in Shared-Nothing Engines. *Proceedings of the Very Large Database Endowment*, 8(12):1542–1553, 2015.

30 Reynold S. Xin, Josh Rosen, Matei Zaharia, Michael J. Franklin, Scott Shenker, and Ion Stoica. Shark: SQL and Rich Analytics at Scale. In *ACM SIGMOD International Conference on Management of Data, SIGMOD 2013*, pages 13–24, 2013.

31 Erfan Zamanian, Carsten Binnig, and Abdallah Salama. Locality-aware Partitioning in Parallel Database Systems. In *ACM SIGMOD International Conference on Management of Data, SIGMOD 2015*, pages 17–30, 2015.

The First Order Truth Behind Undecidability of Regular Path Queries Determinacy

Grzegorz Głuch
Institute of Computer Science, University of Wrocław, Poland

Jerzy Marcinkowski
Institute of Computer Science, University of Wrocław, Poland

Piotr Ostropolski-Nalewaja
Institute of Computer Science, University of Wrocław, Poland

Abstract

In our paper [Głuch, Marcinkowski, Ostropolski-Nalewaja, LICS ACM, 2018] we have solved an old problem stated in [Calvanese, De Giacomo, Lenzerini, Vardi, SPDS ACM, 2000] showing that query determinacy is undecidable for Regular Path Queries. Here a strong generalisation of this result is shown, and – we think – a very unexpected one. We prove that no regularity is needed: determinacy remains undecidable even for finite unions of conjunctive path queries.

2012 ACM Subject Classification Information systems → Graph-based database models

Keywords and phrases database theory, query, view, determinacy, recursive path queries

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.15

Related Version `https://arxiv.org/abs/1808.07767`

Funding Supported by the Polish National Science Centre (NCN) grant 2016/23/B/ST6/01438.

1 Introduction

Query determinacy problem (QDP)

Imagine there is a database $\mathbb{D}$ we have no direct access to, and there are views of this $\mathbb{D}$ available to us, defined by some set of queries $\mathcal{Q} = \{Q_1, Q_2, \ldots Q_k\}$ (where the language of queries from $\mathcal{Q}$ is a parameter of the problem). And we are given another query Q_0. Will we be able, regardless of $\mathbb{D}$, to compute $Q_0(\mathbb{D})$ only using the views $Q_1(\mathbb{D}), \ldots Q_k(\mathbb{D})$? The answer depends on whether the queries in $\mathcal{Q}$ *determine*[1] query Q_0. Stating it more precisely, the **Query Determinacy Problem** is[2]:

> The instance of the problem is a set of queries $\mathcal{Q} = \{Q_1, \ldots Q_k\}$, and another query Q_0. The question is whether $\mathcal{Q}$ determines Q_0, which means that for (♣) each two structures (database instances) $\mathbb{D}_1$ and $\mathbb{D}_2$ such that $Q(\mathbb{D}_1) = Q(\mathbb{D}_2)$ for each $Q \in \mathcal{Q}$, it also holds that $Q_0(\mathbb{D}_1) = Q_0(\mathbb{D}_2)$.

QDP is seen as a very natural static analysis problem in the area of database theory. It is important for privacy (when we don't want the adversary to be able to compute the query) and for (query evaluation plans) optimisation (we don't need to access again the database as the given views already provide enough information). And, as a very natural static analysis problem, it has a 30 years long history as a research subject – the oldest paper we were able

[1] Or, using the language of [6], [5] [9] and [8], whether $\mathcal{Q}$ are *lossless* with respect to Q_0.
[2] More precisely, the problem comes in two different flavors, "finite" and "unrestricted", depending on whether the (♣) "each" ranges over finite structures only, or all structures, including infinite.

to trace, where QDP is studied, is [20], where decidability of QDP is shown for the case where Q_0 is a conjunctive query (CQ) and also the set $\mathcal{Q}$ consists of a single CQ.

But this is not a survey paper, so let us just point a reader interested in the history of QDP to Nadime Francis's thesis [14], which is a very good read indeed.

1.1 The context

As we said, this is a technical paper not a survey paper. But still, we need to introduce the reader to the technical context of our results. And, from the point of view of this introduction, there are two lines of research which are interesting: decidability problems of QDP for positive fragments of SQL (conjunctive queries and their unions) and for fragments of the language of Regular Path Queries (RPQs) – the core of most navigational graph query languages.

QDP for fragments of SQL

A lot of progress was done in this area in two past decades. The paper [21] was the first to present a negative result. QDP was shown there to be undecidable if unions of conjunctive queries are allowed in $\mathcal{Q}$ and Q_0. The proof is moderately hard, but the queries are high arity (by arity of a query we mean the number of free variables) and hardly can be seen as living anywhere close to database practice.

In [22] it was proved that determinacy is also undecidable if the elements of $\mathcal{Q}$ are conjunctive queries and Q_0 is a first order sentence (or the other way round). Another somehow related (although no longer contained in the first order/SQL paradigm) negative result is presented in [12]: determinacy is shown there to be undecidable if $\mathcal{Q}$ is a DATALOG program and Q_0 is a conjunctive query. Finally, closing the classification for the traditional relational model, it was shown in [17] and [18] that QDP is undecidable for Q_0 and the queries in $\mathcal{Q}$ being conjunctive queries. The queries in [17] and [18] are quite complicated (the Turing machine there is encoded in the arities of the queries), and again hardly resemble anything practical.

On the positive side, [22] shows that the problem is decidable for conjunctive queries if each query from $\mathcal{Q}$ has only one free variable.

Then, in [2] decidability was shown for $\mathcal{Q}$ and Q_0 being respectively a set of conjunctive path queries and a path query. (see Section 3 for the definition). This is an important result from the point of view of the current paper, and the proof in [2], while not too difficult, is very nice – it gives the impression of deep insight into the real reasons why a set of conjunctive path queries determines another conjunctive path query.

The result from [2] begs for generalisations, and indeed it was generalised in [23] to the scenario where $\mathcal{Q}$ is a set of conjunctive path queries but Q_0 is any conjunctive query.

QDP for Regular Path Queries

A natural extension of QDP to the graph database scenario is considered here. In this scenario, the underlying data is modelled as graphs, in which nodes are objects, and edge labels define relationships between those objects. Querying such graph-structured data has received much attention recently, due to numerous applications, especially for social networks.

There are many more or less expressive query languages for such databases (see [3]). The core of all of them (the SQL of graph databases) is RPQ – the language of Regular Path Queries. RPQ queries ask for all pairs of objects in the database that are connected by a specified path, where the natural choice of the path specification language, as [26]

elegantly explains, is the language of regular expressions. This idea is at least 30 years old (see for example [11], [10]) and considerable effort was put to create tools for reasoning about regular path queries, analogous to the ones we have in the traditional relational databases context. For example [1] and [4] investigate decidability of the implication problem for path constraints, which are integrity constraints used for RPQ optimisation. Containment of conjunctions of regular path queries has been proved decidable in [7] and [13], and then, in more general setting, in [19] and [24].

Naturally, query determinacy problem has also been stated, and studied, for Regular Path Queries. This line of research was initiated in [6], [5], [9] and [8], and it was in [9] where the central problem of this area – decidability of QDP for RPQ – was first stated (called there "losslessness for exact semantics").

On the positive side, the previously mentioned result of Afrati [2] can be seen as a special case, where each of the regular languages defining the queries only consists of one word (conjunctive path queries considered in [2] constitute in fact the intersection of CQ and RPQ). Another positive result is presented in [15], where "approximate determinacy" is shown to be decidable if the query Q_0 is (defined by) a single-word regular language (a conjunctive path query), and the languages defining the queries in Q_0 and $\mathcal{Q}$ are over a single-letter alphabet. See how difficult the analysis is here – despite a lot of effort (the proof of the result in [15] invokes ideas from [2] but is incomparably harder) even a subcase (for a single-word regular language) of a subcase (unary alphabet) was only understood "approximately".

On the negative side, in [16], we showed (solving the problem from [9]), that QDP is undecidable for full RPQ.

1.2 Our contribution

The main result of this paper, and – we think – quite an unexpected one, is the following strong generalisation of the main result from [16]:

▶ **Theorem 1.1.** *Answering Determinacy question for Finite Regular Path Queries is undecidable in unrestricted and finite case (for necessary definitions see Section 3).*

To be more precise, we show that the problem, both in the "finite" and the "unrestricted" versions, is undecidable.

It is, we believe, interesting to see that this negative result falls into both lines of research outlined above. Finite Regular Path Queries are of course a subset of RPQ, where star is not allowed in the regular expressions (only concatenation and plus are), but on the other hand they are also Unions of Conjunctive Path Queries, so unlike general RPQs are first order queries and they also fall into the SQL category.

Our result shows that the room for generalising the positive result from [2] is quite limited. What we however find most surprising is the discovery that it was possible to give a negative answer to the question from [9], which had been open for 15 years, without talking about RPQs at all – undecidability is already in the intersection of RPQs and (positive) SQL.

▶ Remark. [3] makes a distinction between "simple paths semantics" for Recursive Path Queries and "all paths semantics". As all the graphs we produce in this paper are acyclic (DAGs), all our results hold for both semantics.

Organization of the paper.

In short Section 3 we introduce (very few) notions and some notations we need to use. Sections 4–14 of this paper are devoted to the proof of Theorem 1.1.

In Section 4 we first follow the ideas from [16] defining the red-green signature. Then we define the game of Escape and state a crucial lemma (Lemma 4.2), asserting that this game really fully characterises determinacy for Regular Path Queries. This part follows in the footsteps of [16], but with some changes: in [16] Escape is a solitary game, and here we prefer to see it as a two-player one.

At this point we will have the tools ready for proving Theorem 1.1. In Section 5 we explain what is the undecidable problem we use for our reduction, and in Section 6 we present the reduction. In Sections 7 – 14 we use the characterisation provided by Lemma 4.2 to prove correctness of this reduction. Due to the space constraint, we try to explain the main ideas and formulate all the crucial lemmas in the main body of the paper, but the proofs of the lemmas can be found in the full paper.

2 How this paper relates to [16]

This paper builds on top of the technique developed in [16] to prove undecidability of QDP-RPQ for any languages, including infinite.

From the point of view of the high-level architecture the two papers do not differ much. In both cases, in order to prove that if some computational device rejects its input then the respective instance of QDP-RPQ (or QDP-FRPQ) is positive (there is determinacy) we use a game argument. In [16] this game is solitary. The player, called *Fugitive*, constructs a structure/graph database (a DAG, with source a and sink b). He begins the game by choosing a path $\mathbb{D}_0$ from a to b, which represents a word from some regular language $G(Q_0)$. Then, in each step he must "satisfy requests"– if there is a path from some v to w in the current structure, representing a word from some (*) regular language Q then he must add a path representing a word from another language Q' connecting these v and w. He loses when, in this process, a path from a to b from yet another language $R(Q_0)$ is created. In this paper this game is replaced by a two-player game. But this is a minor difference. There are however two reasons why the possibility of using infinite languages is crucial in [16]. Due to these reasons, while, as we said, the general architecture of the proof of the negative result in this paper is the same as in [16], the implementation of this architecture is almost completely different here.

The first reason is as follows. Because of the symmetric nature of the constraints, the language Q (in (*) above) is always almost the same as language Q' (they only have different "colors", but otherwise are equal). For this reason it is not at all clear how to force *Fugitive* to build longer and longer paths. This is a problem for us, as to be able to encode something undecidable we need to produce structures of unbounded size. One can think that paths of unbounded length translate to potentially unbounded length of Turing machine tape.

In order to solve this problem we use – in [16] – a language $G(Q_0)$. It is an infinite language and – in his initial move – *Fugitive* could choose/commit to a path of any length he wished so that the length of the path did not need to increase in the game. But now we only have finite languages, so also $G(Q_0)$ must be finite and we needed to invent something completely different.

The second reason is in $R(Q_0)$. This – one can think – is the language of "forbidden patterns" – paths from a to b that *Fugitive* must not construct. If he does, it means that he "cheats". But now again, $R(Q_0)$ is finite. So how can we use it to detect *Fugitive's* cheating on paths no longer than the longest one in $R(Q_0)$? This at first seemed to us to be an impossible task.

But it wasn't impossible. The solution to both aforementioned problems is in the complicated machinery of languages producing edges labelled with x and y.

3 Preliminaries and notations

Determination

For a set of queries $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_k\}$ and another query Q_0, we say that $\mathcal{Q}$ *determines* Q_0 if and only if: $\forall_{\mathbb{D}_1,\mathbb{D}_2}\ \mathcal{Q}(\mathbb{D}_1) = \mathcal{Q}(\mathbb{D}_2) \to Q_0(\mathbb{D}_1) = Q_0(\mathbb{D}_2)$, where $\mathcal{Q}(\mathbb{D}_1) = \mathcal{Q}(\mathbb{D}_2)$ is defined as $\forall_{Q\in\mathcal{Q}}\ Q(\mathbb{D}_1) = Q(\mathbb{D}_2)$. *Query determinacy* comes in two versions: *unrestricted* and *finite*, depending on whether we allow or disallow infinite structures $\mathbb{D}$ to be considered. When we speak about *determinacy* without specifying explicitly it's version, we assume the *unrestricted* case of the problem.

Structures

When we say "structure" we always mean a directed graph with edges labelled with letters from some signature/alphabet Σ. In other words every structure we consider is a relational structure $\mathbb{D}$ over some signature Σ consisting of binary predicate names. Letters $\mathbb{D}$, $\mathbb{M}$, $\mathbb{G}$ and $\mathbb{H}$ are used to denote structures. Ω is used for a set of structures. Every structure we consider will contain two distinguished constants a and b. For two structures $\mathbb{G}$ and $\mathbb{G}'$ over Σ, with sets of vertices V and V', a function $h : V \to V'$ is called a homomorphism if for each two vertices $\langle u, v\rangle$ connected by an edge with label $e \in \Sigma$ in $\mathbb{G}$ there is an edge connecting $\langle h(u), h(v)\rangle$, with the same label e, in $\mathbb{G}'$.

Conjunctive path queries

Given a set of binary predicate names Σ and a word $w = a_1a_2\ldots a_n$ over Σ^* we define a (conjunctive) path query $w(v_0, v_n)$ as a conjunctive query:

$$\exists_{v_1,\ldots,v_{n-1}} a_1(v_0, v_1) \wedge a_2(v_1, v_2) \wedge \ldots a_n(v_{n-1}, v_n).$$

We use the notation $w[v_0, v_n]$ to denote the canonical structure ("frozen body") of query $w(v_0, v_n)$ – the structure consisting of elements $v_0, v_1, \ldots v_n$ and atoms $a_1(v_0, v_1)$, $a_2(v_1, v_2)$, $\ldots$ $a_n(v_{n-1}, v_n)$.

Regular path queries

For a regular language Q over Σ we define a query, which is also denoted by Q, as $Q(u, v) = \exists_{w\in Q} w(u, v)$

In other words such a query Q looks for a path in the given graph labelled with any word from Q and returns the endpoints of that path. Clearly, if Q is a finite regular language (finite regular path query), then $Q(u, v)$ is a union of conjunctive queries.

We use letters Q and L to denote regular languages and $\mathcal{Q}$ and $\mathcal{L}$ to denote sets of regular languages. The notation $Q(\mathbb{D})$ has the natural meaning: $Q(\mathbb{D}) = \{\langle u, v\rangle \mid \mathbb{D} \models Q(u, v)\}$.

4 Red-Green Structures and Escape

In this section we will provide crucial tool for our proof. First we will introduce red-green structures. Such a structure will consist of two structures each with distinct colour. One can think that we take two databases and then colour one green and another red and then look at them as a whole. This notion is very useful for two coloured *Chase* technique from [17] and [18] that has evolved into Game of Escape in [16] and is also present in this paper.

4.1 Red-green signature and Regular Constraints

For a given alphabet (signature) Σ let Σ_G and Σ_R be two copies of Σ one written with "green ink" and another with "red ink". Let $\bar{\Sigma} = \Sigma_G \cup \Sigma_R$.

For any word w from Σ^* let $G(w)$ and $R(w)$ be copies of this word written in green and red respectively. For a regular language L over Σ let $G(L)$ and $R(L)$ be copies of this same regular language but over Σ_G and Σ_R respectively. Also for any structure $\mathbb{D}$ over Σ let $G(\mathbb{D})$ and $R(\mathbb{D})$ be copies of this same structure $\mathbb{D}$ but with labels of edges recolored to green and red respectively. For a pair of regular languages L over Σ and L' over Σ' we define the *Regular Constraint (RC)* $L \to L'$ as a formula $\forall_{u,v} L(u,v) \Rightarrow L'(u,v)$.

We use the notation $\mathbb{D} \models t$ to say that an RC t is satisfied in $\mathbb{D}$. Also, we write $\mathbb{D} \models T$ for a set T of RCs when for each $t \in T$ it is true that $\mathbb{D} \models t$.

For a graph $\mathbb{D}$ and an RC $t = L \to L'$ let $rq(t, \mathbb{D})$ (as "requests") be the set of all triples $\langle u, v, L \to L' \rangle$ such that $\mathbb{D} \models L(u,v)$ and $\mathbb{D} \not\models L'(u,v)$. For a set T of RCs by $rq(T, \mathbb{D})$ we mean the union of all sets $rq(t, \mathbb{D})$ such that $t \in T$. Requests are there in order to be satisfied:

Algorithm 1 Function Add used to satisfy requests.

function *Add*
arguments:
- Structure $\mathbb{D}$
- RC $L \to L'$
- pair $\langle u, v \rangle$ such that $\langle u, v, L \to L' \rangle \in rq(L \to L', \mathbb{D})$

body:

1: Take a word $w = a_0 a_1 \dots a_n$ from L' and create a new path $w[u,v] = a_0(u, s_1), a_1(s_1, s_2), \dots, a_n(s_{n-1}, v)$ where $s_1, s_2, \dots, s_{n-1}$ are **new** vertices
2: **return** $\mathbb{D} \cup w[u,v]$.

Notice that the result $Add(D, L \to L', \langle u, v \rangle)$ depends on the choice of $w \in L'$. So the procedure is non-deterministic.

For a regular language L we define $L^{\rightarrow} = G(L) \to R(L)$ and $L^{\leftarrow} = R(L) \to G(L)$. All regular constraints we are going to consider are either $L^{\rightarrow}$ or $L^{\leftarrow}$. For a regular language L we define $L^{\leftrightarrow} = \{L^{\rightarrow}, L^{\leftarrow}\}$ and for a set $\mathcal{L}$ of regular languages we define: $\mathcal{L}^{\leftrightarrow} = \bigcup_{L \in \mathcal{L}} L^{\leftrightarrow}$.

Requests of the form $\langle u, v, t \rangle$ for some RC t of the form $L^{\rightarrow}$ ($L^{\leftarrow}$) are *generated by* $G(L)$ (resp. *by* $R(L)$). Requests that are generated by $G(L)$ or $R(L)$ are said to be *generated by* L.

The following lemma is straightforward to prove and characterises determinacy in terms of regular constraints:

▶ **Lemma 4.1.** *A set $\mathcal{Q}$ of regular path queries over Σ does not determine (does not finitely determine) a regular path query Q_0, over the same alphabet, if and only if there exists a structure $\mathbb{M}$ (resp. a finite structure) and a pair of vertices $u, v \in \mathbb{M}$ such that $\mathbb{M} \models \mathcal{Q}^{\leftrightarrow}$ and $\mathbb{M} \models (G(Q_0))(u,v)$ but $\mathbb{M} \not\models (R(Q_0))(u, v-)$.*

Any structure $\mathbb{M}$, as above, will be called a *counterexample*. One can think of $\mathbb{M}$ as a pair of structures, being green and red parts of $\mathbb{M}$. Note that those two structures both agree on $\mathcal{Q}$ but don't on Q_0, thus proving that $\mathcal{Q}$ does not determine Q_0.

4.2 The game of Escape

Here we present the essential tool for our proof. One can note that the game of Escape is very simmilar to the well known *Chase* technique. This is indeed the case, as one can think about RCs as of *Tuple Generating Dependencies* (TGDs) from the *Chase*. Divergence from

standard *Chase* comes from "nondeterminism" that is inherent part of RCs (request can be satisfied by any word from a language) phenomenon not present in TGDs.

An instance Escape(Q_0, $\mathcal{Q}$) of a game called *Escape*, played by two players called *Fugitive* and *Crocodile*, is:

- A finite regular language Q_0 of *forbidden paths* over Σ.
- A set $\mathcal{Q}$ of finite regular languages over Σ,

The rules of the game are:

- First *Fugitive* picks the *initial position* of the game as $\mathbb{D}_0 = (G(w))[a, b]$ for some $w \in Q_0$.
- Suppose $\mathbb{D}_\beta$ is the current position of some play before move $\beta+1$ and let $S_\beta = rq(\mathcal{Q}^{\leftrightarrow}, \mathbb{D}_\beta)$. Then, in move $\beta + 1$, *Crocodile* picks one request $\langle u, v, t\rangle \in S_\beta$ and then *Fugitive* can move to any position of the form:

$$\mathbb{D}_{\beta+1} := Add(\mathbb{D}_\beta, t, \langle u, v\rangle)$$

- For a limit ordinal λ the position $\mathbb{D}_\lambda$ is defined as $\bigcup_{\beta<\lambda} \mathbb{D}_\beta$.
- If $rq(\mathcal{Q}^{\leftrightarrow}, \mathbb{D}_i)$ is empty then for each $j > i$ the structures $\mathbb{D}_j$ and $\mathbb{D}_i$ are equal.
- *Fugitive* loses when for a *final position* $\mathbb{D}_{\omega^2} = \bigcup_{\beta<\omega^2} \mathbb{D}_\beta$ it is true that $\mathbb{D}_{\omega^2} \models (R(Q_0))(a, b)$, otherwise he wins. Obviously if there is some $\beta < \omega^2$ such that $\mathbb{D}_\beta \models (R(Q_0))(a, b)$ then the result of the game is already known (*Fugitive* loses), but technically the game still proceeds.

Notice that we want the game to last ω^2 steps. This is not really crucial (if we were careful ω steps would be enough) but costs nothing and will simplify presentation in Section 10.

Obviously, different strategies of both players may lead to different final positions.

Now we can state the crucial Lemma, that connects the game of Escape and QDP-RPQ:

▶ **Lemma 4.2.** *For an instance of QDP-RPQ consisting of regular language Q_0 over Σ and a set of regular languages $\mathcal{Q}$ over Σ the two conditions are equivalent:*

(i) *$\mathcal{Q}$ does not determine Q_0,*

(ii) Fugitive *has a winning strategy in Escape(Q_0, $\mathcal{Q}$).*

The proof of this Lemma can be found in the full paper.

We should mention here that all the notions of Section 4 are similar to those of [16] but are not identical. The most notable difference is in the definition of the game of Escape, as it is no longer a solitary game, as it was in [16].

This makes the analysis slightly harder here, but pays off in Sections 7 – 14.

5 Source of undecidability

In this section we will define tiling problem that we will reduce to QDP-FRPQ. In order to prove undecidability for both finite and unrestricted case we will build our tiling problem upon notion of *recursively inseparable sets*.

▶ **Definition 5.1 (Recursively inseparable sets).** *Sets A and B are called recursively inseparable when each set C, called a separator, such that $A \subseteq C$ and $B \cap C = \emptyset$, is undecidable [25].*

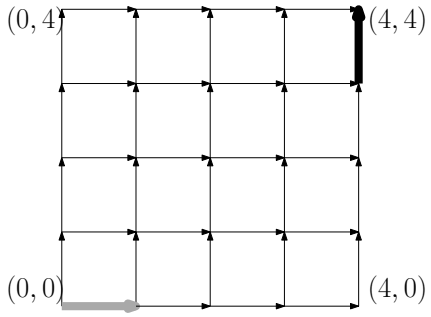


Figure 1 Finite square grid.

It is well known that:

▶ **Lemma 5.2.** *Let T be the set of all Turing Machines. Then sets $T_{acc} = \{\phi \in T | \phi(\varepsilon) = 1\}$ and $T_{rej} = \{\phi \in T | \phi(\varepsilon) = 0\}$ are recursively inseparable. By $\phi(\varepsilon)$ we mean the returned value of the Turing Machine ϕ that was run on an empty tape.*

▶ **Definition 5.3 (Square Grids).** *For a $k \in \mathbb{N}$ let $[k]$ be the set $\{i \in \mathbb{N} | 0 \leq i \leq k\}$. A square grid is a directed graph $\langle V, E\rangle$ where $V = [k] \times [k]$ for some natural $k > 0$ or $V = \mathbb{N} \times \mathbb{N}$. E is defined as $E(\langle i, j\rangle, \langle i + 1, j\rangle)$ and $E(\langle i, j\rangle, \langle i, j + 1\rangle)$ for each relevant $i, j \in \mathbb{N}$.*

▶ **Definition 5.4 (Our Grid Tiling Problem (OGTP)).** *An instance of this problem is a set of **shades** $\mathcal{S}$ having at least two elements ($\boldsymbol{gray}$,$\boldsymbol{black} \in \mathcal{S}$) and a set $\mathcal{F} \subseteq \{V, H\} \times \mathcal{S} \times \{V, H\} \times \mathcal{S}$ of forbidden pairs $\langle c, d\rangle$ where $c, d \in \{V, H\} \times \mathcal{S}$. Let the set of all these instances be called $\mathcal{I}$.*

▶ **Definition 5.5.** *A proper shading*[3] *is an assignment of shades to edges of some square grid $\mathbb{G}$ (see Figure 1) such that:*

(a1) *each horizontal edge of $\mathbb{G}$ has a label from $\{H\} \times \mathcal{S}$.*
(a2) *each vertical edge of $\mathbb{G}$ has a label from $\{V\} \times \mathcal{S}$.*
(b1) *bottom-left horizontal edge is shaded $\boldsymbol{gray}$*[4]*.*
(b2) *upper-right vertical edge (if it exists) is shaded $\boldsymbol{black}$.*
(b3) *$\boldsymbol{G}$ contains no forbidden paths of length 2 labelled by $\langle c, d\rangle \in \mathcal{F}$.*

We define two subsets of instances of OGTP:
$\boldsymbol{\mathcal{A}} = \{I \in \mathcal{I} | \text{there exists a proper shading of some finite square grid}\}$.
$\boldsymbol{\mathcal{B}} = \{I \in \mathcal{I} | \text{there is no proper shading of any square grid}\}$.

By a standard argument, using Lemma 5.2, one can show that:

▶ **Lemma 5.6.** *Sets $\mathcal{A}$ and $\mathcal{B}$ of instances of OGTP are recursively inseparable.*

In Section 6 we will construct a function $\mathfrak{R}$ ($\mathfrak{R}$ like $\mathfrak{R}$eduction) from $\mathcal{I}$ (instances of OGTP) to instances of QPD-FRPQ that will satisfy the following:

[3] We would prefer to use the term "coloring" instead, but we already have colors, red and green, and they shouldn't be confused with shades.
[4] We think of $(0, 0)$ as the bottom-left corner of a square grid. By 'right' we mean a direction of the increase of the first coordinate and by 'up' we mean a direction of increase of the second coordinate.

▶ **Lemma 5.7.** *For any instance $I = \langle \mathcal{S}, \mathcal{F} \rangle$ of OGTP and for $\langle \mathcal{Q}, Q_0 \rangle = \mathfrak{R}(I)$:*

(i) *If $I \in \mathcal{A}$ then $\mathcal{Q}$ does not finitely determine Q_0.*

(ii) *If $I \in \mathcal{B}$ then $\mathcal{Q}$ determines Q_0.*

Now the need for notion of recursive inseparability should be clear. Imagine, for the sake of contradiction, that we have an algorithm ALG deciding determinacy in either finite or unrestricted case. Then, in both cases, algorithm $ALG \circ \mathfrak{R}$ would separate $\mathcal{A}$ and $\mathcal{B}$, which contradicts the recursive inseparability of $\mathcal{A}$ and $\mathcal{B}$ (Lemma 5.6). That will be enough to prove Theorem 1.1.

6 The function $\mathfrak{R}$

Now we define a function $\mathfrak{R}$, as specified in Section 5, from the set of instances of OGTP to the set of instances of QDP-FRPQ. Suppose an instance $\langle \mathcal{S}, \mathcal{F} \rangle$ of OGTP is given. We will construct an instance $\langle \mathcal{Q}, Q_0 \rangle = \mathfrak{R}(\langle \mathcal{S}, \mathcal{F} \rangle)$ of QDP-FRPQ. The edge alphabet (signature) will be: $\Sigma = \{\alpha^C, \alpha^W, x^C, x^W, y^C, y^W, \$^C, \$^W, \omega\} \cup \Sigma_0$ where $\Sigma_0 = \{\mathbf{A}, \mathbf{B}\} \times \{H, V\} \times \{W, C\} \times \mathcal{S}$. We think of H and V as **orientations** – *Horizontal* and *Vertical*. W and C stand for *warm* and *cold*. It is worth reminding at this point that relations from $\bar{\Sigma}$ will – apart from shade, orientation and temperature – have also a **color**, red or green.

▶ **Notation 6.1.** *We will denote $(\mathbf{l}, o, t, s) \in \Sigma_0$ as $({}_s\mathbf{l}^t_o)$.*

Symbol $\bullet$ and empty space are to be understood as wildcards. This means, for example, that $({}_s\mathbf{A}_H)$ denotes the set $\{({}_s\mathbf{A}^W_H), ({}_s\mathbf{A}^C_H)\}$ and $({}_s\bullet^W_H)$ denotes $\{({}_s\mathbf{A}^W_H), ({}_s\mathbf{B}^W_H)\}$.

*Symbols from $(\bullet^W)$ and $\{\alpha^W, x^W, y^W, \$^W\}$ will be called **warm** and symbols from $(\bullet^C)$ and $\{\alpha^C, x^C, y^C, \$^C\}$ will be called **cold**.*

Now we define $\mathcal{Q}$ and Q_0. Let $\mathcal{Q}_{good}$ be a set of 15 languages:

1. ω
2. $\alpha^C + \alpha^W$
3. $x^C + x^W$
4. $y^C + y^W$
5. $\$^C + \$^W$
6. $(\mathbf{B}^C_V) + (\mathbf{B}^W_V)$
7. $(\mathbf{B}^W_H) + (\mathbf{B}^C_H)$
8. $(\mathbf{A}^W_V) + (\mathbf{A}^C_V)$
9. $(\mathbf{A}^C_H) + (\mathbf{A}^W_H)$
10. $(\mathbf{B}^W_H)(\mathbf{A}^W_V) + (\mathbf{B}^C_V)(\mathbf{A}^C_H)$
11. $(\mathbf{A}^C_H)(\mathbf{B}^C_V) + (\mathbf{A}^W_V)(\mathbf{B}^W_H)$
12. $x^C\big((\mathbf{A}^C_H)+(\mathbf{B}^C_H)+(\mathbf{A}^C_V)+(\mathbf{B}^C_V)\big)+x^C + x^W$
13. $\big((\mathbf{A}^C_H)+(\mathbf{B}^C_H)+(\mathbf{A}^C_V)+(\mathbf{B}^C_V)\big)\, y^C+y^C+y^W$
14. $x^W + x^C + x^C(\mathbf{A}^C_H)(\mathbf{B}^C_V)$
15. $y^W + \$^C + (\mathbf{A}^C_H)(\mathbf{B}^C_V)y^C + (\mathbf{B}^C_V)y^C$

Let $\mathcal{Q}_{bad}$ be a set of languages:

1. $\alpha^W x^W ({}_s\bullet^W_d)({}_{s'}\bullet^W_{d'})y^W\omega$ for each forbidden pair $\langle (d, s), (d', s') \rangle \in \mathcal{F}$.
2. $\alpha^W x^W ({}_{shade}\mathbf{B}^W_V)\$^W\omega$ for each $shade \in \mathcal{S} \setminus \{black\}$.

Finally, let $\mathcal{Q}_{ugly}$ be a set of languages: 1. $\alpha^C\Sigma^{\leq 4}(\bullet^W)\Sigma^{\leq 4}\omega$, 2. $\alpha^W\Sigma^{\leq 4}(\bullet^C)\Sigma^{\leq 4}\omega$ and 3. $\alpha^C x^C(\mathbf{B}^C_V)(\mathbf{B}^C_V)y^C\omega$. Where $\Sigma^{\leq 4}$ is language over Σ of words shorter than 5.

We write $Q^i_{good}, Q^i_{bad}, Q^i_{ugly}$ to denote the i-th language of the corresponding group. Now we can define: $\mathcal{Q} := \mathcal{Q}_{good} \cup \mathcal{Q}_{bad} \cup \mathcal{Q}_{ugly}$

The sense of the construction will (hopefully) become clear later, but we can already say something about the structure of the defined languages.

Languages from Q_{good} serve as a building blocks during the game of Escape, they are used solely to build a shaded square grid structure, that will correspond to some tiling of square grid. On the other hand the languages from Q_{bad} and Q_{ugly} serve as a "stick" during play. It will become clear later that whenever *Fugitive* has to answer a request generated by

one of languages in Q_{bad} or Q_{ugly} it is already too late for him to win. More precisely every request generated by Q_{bad} is due to *Fugitive* assigning shades to grid in a forbidden manner and leads to *Fugitive's* swift demise. Similarly requests generated by Q_{ugly} make *Fugitive* to use "right" building blocks from Q_{good}, by forcing that edge is red if and only if it is *warm* (this will be thoroughly explained in Section 8), or else he will loose. Third language from Q_{ugly} ensures that there won't be two consecutive $(\mathbf{B}_V^C)$ symbols in the structure, it is a feature used exclusively in proof of Lemma 11.1.

Finally, define $Q_{start} := \alpha^C x^C ({}_{gray}\mathbf{A}_H^C)(\mathbf{B}_V^C) y^C \omega$, and let:

$$Q_0 := Q_{start} + \bigoplus_{L \in \mathcal{Q}_{ugly}} L + \bigoplus_{L \in \mathcal{Q}_{bad}} L$$

Intuitively, we need to put the languages Q_{bad} and Q_{ugly} into Q_0 so that they can serve as aforementioned "sticks" such that whenever *Fugitive* makes a mistake he loses. Recall that *Fugitive* loses when he writes a red word from Q_0 connecting the beginning and the end of the starting structure. On the other side, this creates a problem regarding the starting structure (now *Fugitive* can pick a word from Q_{bad} or Q_{ugly} to start from instead of a word from Q_{start}), but fortunately it can be dealt with and will be resolved in Section 8.

7 Structure of the proof of Lemma 5.7

The rest of the paper will be devoted to the proof of Lemma 5.7 (restated):

▶ **Lemma 7.1.** *For any instance $I = \langle \mathcal{S}, \mathcal{F} \rangle$ of OGTP and for $\langle \mathcal{Q}, Q_0 \rangle = \mathfrak{R}(I)$:*

(i) *If $I \in \mathcal{A}$ then $\mathcal{Q}$ does not finitely determine Q_0.*

(ii) *If $I \in \mathcal{B}$ then $\mathcal{Q}$ determines Q_0.*

Proof of claim (i) – which will be presented in the end of Section 14 – will be straightforward once the reader grasps the (slightly complicated) constructions that will emerge in the proof of claim (ii).

For the proof of claim (ii) we will employ Lemma 4.2, showing that if the instance $\langle \mathcal{S}, \mathcal{F} \rangle$ has no proper shading then *Crocodile* **does have** a winning strategy in Escape($\mathcal{Q}$, Q_0) (where $\langle \mathcal{Q}, Q_0 \rangle = \mathfrak{R}(\langle \mathcal{S}, \mathcal{F} \rangle)$). As we remember from Section 4.2, in such a game *Fugitive* will first choose, as the initial position of the game, a structure $\mathbb{D}_0 = w[a, b]$ for some $w \in G(Q_0)$. Then, in each step, *Crocodile* will pick a request in the current structure (current position of the game) $\mathbb{D}$ and *Fugitive* will satisfy this request, creating a new (slightly bigger) current $\mathbb{D}$. *Fugitive* will win if he will be able to play forever (by which, formally speaking, we mean ω^2 steps, for more details see Section 4.2), or until all requests are satisfied, without satisfying (in the constructed structure) the query $(R(Q_0))(a, b)$. While talking about the strategy of *Fugitive* we will use the words "must not" and "must" as shorthands for "or otherwise he will quickly lose the game". The expression "*Fugitive* is forced to" will also have this meaning.

Analysing a two-player game (proving that certain player has a winning strategy) sounds like a complicated task: there is this (infinite) alternating tree of positions, whose structure somehow needs to be translated into a system of lemmas. In order to prune this game tree our plan is first to notice that in his strategy *Fugitive* must obey the following principles:

(I) The structure $\mathbb{D}_0$ resulting from his initial move must be $(G(w))[a, b]$ for some $w \in Q_{start}$.

(II) He must not allow any green edge with warm label and any red edge with cold label to appear in $\mathbb{D}$.

(III) He must never allow any path labelled by a word from the language $G(\mathcal{Q}_{bad}) \cup R(\mathcal{Q}_{bad})$ to occur between vertices a and b.

Then we will assume that *Fugitive's* play indeed follows the three principles and we will present a strategy for *Crocodile* which will be winning against *Fugitive*. From the point of view of *Crocodile's* operational objectives this strategy comprises three stages.

In each of these stages *Crocodile's* operational goal will be to force *Fugitive* to build some specified structure (where, of course all the specified structures will be superstructures of $\mathbb{D}_0$). In the first stage *Fugitive* will be forced to build a structure called $\mathbb{P}_1$ (defined in Section 9). In the second stage the specified structures will be called $\mathbb{P}_m$ and ${}^{\$}\mathbb{P}_m$ (each defined in Section 9) and in the third stage *Fugitive* will be forced to construct one of the structures $\mathbb{G}_m$ or $\mathbb{L}^k_m$ (defined in Section 12)

During the three stages of his play *Crocodile* will only pick requests from the languages in $\mathcal{Q}_{good}$. These languages, as we said before, are shade-insensitive, so we can imagine *Crocodile* playing in a sort of shade filtering glasses. Of course *Fugitive*, when responding to *Crocodile's* requests, will need to commit on the shades of the symbols he will use, but *Crocodile's* actions will not depend on these shades.

The shades will however play their part after the end of the third stage. Assuming that the original instance of OGTP has no proper shading, we will get that, at this moment, $R(\mathcal{Q}_{bad})(a, b)$ already holds true in the structure *Fugitive* was forced to construct. This will end the proof of (ii).

8 Principles of the Game

The rules of the game of Escape are such that *Fugitive* loses when he builds a path (from a to b) labelled with $w \in R(Q_0)$. So – when trying to encode something – one can think of words in Q_0 as of some sort of forbidden patterns. And thus one can think of Q_0 as of a tool detecting that *Fugitive* is cheating and not really building a valid computation of the computing device we encode. Having this in mind the reader can imagine why the words from languages from the sets $\mathcal{Q}_{bad}$ and $\mathcal{Q}_{ugly}$, which clearly are all about suspiciously looking patterns, are all in Q_0.

But another rule of the game is that at the beginning *Fugitive* picks his initial position $\mathbb{D}_0$ as a path (from a to b) labelled with some $w \in G(Q_0)$, so it would be nice to think of Q_0 as of initial configurations of this computing device. The fact that the same object is playing the set of forbidden patterns and, at the same time, the set of initial configurations is a problem. We solved it by having languages from $\mathcal{Q}_{ugly} \cup \mathcal{Q}_{bad}$ both in $\mathcal{Q}$ and in Q_0:

▶ **Lemma 8.1 (Principle I).** Fugitive *must choose to start the Escape game from* $\mathbb{D}_0 = G(w)[a, b]$ *for some* $w \in Q_{start}$.

Notice that, from the point of view of the shades-blind *Crocodile* the words in Q_{start} are indistinguishable and thus *Fugitive* only has one possible choice of $\mathbb{D}_0$. Proof of this Lemma can be found in the full paper.

From now on we assume that *Fugitive* obeys Principle I. This implies that for some $w \in Q_{start}$ structure $G(w)[a, b]$ is contained in all subsequent structures $\mathbb{D}$ at each step of the game.

Now we will formalise the intuition about languages from $\mathcal{Q}_{ugly}$ as forbidden patterns. We start with an observation that simplifies reasoning in the proof of Principle II.

▶ **Observation 8.2.** *For some vertices* u, v *in the current structure* $\mathbb{D}$ *if there is a green (red) edge between them then* Crocodile *can force* Fugitive *to draw a red (green) edge between* u *and* v.

Proof. It is possible due to languages $1 - 9$ in $\mathcal{Q}_{good}$. ◀

▶ **Definition 8.3.** *A P2-ready*[5] *structure* $\mathbb{D}$ *is a structure satisfying the following:*

- $\mathbb{D}_0$ *is a substructure of* $\mathbb{D}$*;*
- *there are only two edges incident to* a*:* $\langle a, a'\rangle$ *with label* $G(\alpha^C)$ *and* $\langle a, a'\rangle$ *with label* $R(\alpha^W)$*;*
- *all edges labeled with* α^C *and* α^W *are between* a *and* a'*;*
- *there are only two edges incident to* b*:* $\langle b', b\rangle$ *with label* $G(\omega)$ *and* $\langle b', b\rangle$ *with label* $R(\omega)$*;*
- *all edges labeled with* ω *are between* b' *and* b*;*
- *for each* $v \in \mathbb{D} \setminus \{a, b\}$ *there is a directed path in* $\mathbb{D}$*, of length at most* 4*, from* a' *to* v *and there is a directed path in* $\mathbb{D}$*, of length at most* 4*, from* v *to* b'*.*

▶ **Lemma 8.4 (Principle II).** *Suppose that, after* Fugitive's *move, the current structure* $\mathbb{D}$ *is a P2-ready structure. Then neither a green edge with label from* $(\bullet^W)$ *nor a red edge with label from* $(\bullet^C)$ *may appear in* $\mathbb{D}$*.*

The proof of this Lemma can be found in the full paper.

▶ **Lemma 8.5 (Principle III).** Fugitive *must not allow any path labelled with a word from* $R(\mathcal{Q}_{bad}) \cup G(\mathcal{Q}_{bad})$ *to occur in the current structure* $\mathbb{D}$ *between vertices* a *and* b*.*

The proof of this Lemma can be found in the full paper.

9 The paths $\mathbb{P}_m$ and ${}^{\$}\mathbb{P}_m$

▶ **Definition 9.1.** *(See Figure 2, please use a color printer if you can)* $\mathbb{P}_m$*, for* $m \in \mathbb{N}_+$*, is a directed graph* (V, E) *where* $V = \{a, a', b', b\} \cup \{v_i : i \in [0, 2m]\}$ *and the edges* E *are labelled with symbols from* $\Sigma \setminus \Sigma_0$ *or with symbols of the form* $(\mathbf{l}^t_o)$*, where – like before –* $\boldsymbol{l} \in \{A, B\}$*,* $o \in \{H, V\}$ *and* $t \in \{W, C\}$*. Each label has to also be either red or green. Notice that there is no* $s \in \mathcal{S}$ *here: the labels we now use are sets of symbols from* $\bar{\Sigma}$ *like in Notation 6.1: we watch the play in* Crocodile's *shade filtering glasses.*

The edges of $\mathbb{P}_m$ *are as follows:*

- *Vertex* a' *is a successor of* a *and vertex* b *is a successor of* b'*. For each* $i \in [2m]$ *the successors of* v_i *are* v_{i+1} *(if it exists) and* b' *and the predecessors of* v_i *are* v_{i-1} *(if it exists) and* a'*. From each node there are two edges to each of its successors, one red and one green, and there are no other edges.*
- *Each Cold edge (labelled with a symbol in* $(\bullet^C)$*) is green.*
- *Each Warm edge (labelled with a symbol in* $(\bullet^W)$*) is red.*
- *Each edge* $\langle v_{2i}, v_{2i+1}\rangle$ *is from* $(\mathbf{A}_H)$*.*
- *Each edge* $\langle v_{2i+1}, v_{2i+2}\rangle$ *is from* $(\mathbf{B}_V)$*.*
- *Each edge* $\langle a', v_i\rangle$ *is labelled by either* x^C *or* x^W*.*
- *Each edge* $\langle v_i, b'\rangle$ *is labelled by either* y^C *or* y^W*.*
- *Edges* $\langle a, a'\rangle$ *with label* $G(\alpha^C)$ *and* $\langle a, a'\rangle$ *with label* $R(\alpha^W)$ *are in* E*.*
- *Edges* $\langle b', b\rangle$ *with label* $G(\omega)$ *and* $\langle b', b\rangle$ *with label* $R(\omega)$ *are in* E*.*

▶ **Definition 9.2.** ${}^{\$}\mathbb{P}_m$ *for* $m \in \mathbb{N}_+$ *is* $\mathbb{P}_m$ *with two additional edges:* $\langle v_{2m}, b'\rangle \in E$*, with label* $G(\$^C)$*, and* $\langle v_{2m}, b'\rangle \in E$*, with label* $R(\$^W)$*.*

[5] Meaning "ready for Principle II".

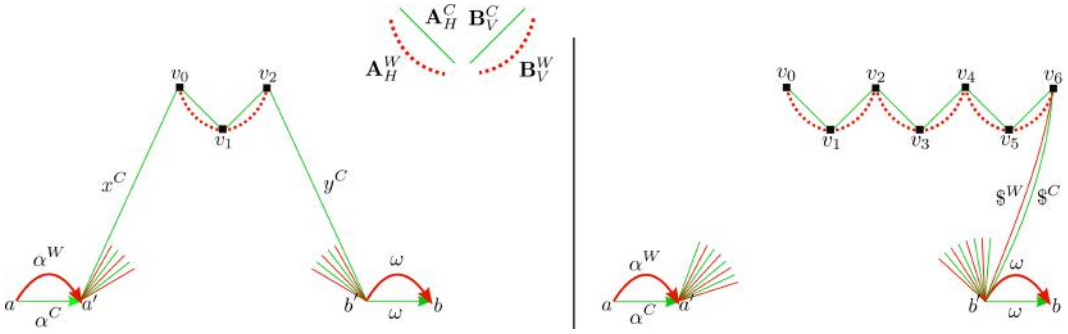


Figure 2 $\mathbb{P}_1$ (left) and ${}^{\$}\mathbb{P}_3$ (right).

One may notice[6] that $\mathbb{D}_0$ is a substructure of both $\mathbb{P}_m$ and ${}^{\$}\mathbb{P}_m$, and that:

▶ **Exercise 9.3.** *For any m, the only requests generated by $\mathcal{Q}_{good}$ in ${}^{\$}\mathbb{P}_m$ are those generated by Q^{10}_{good} and Q^{11}_{good}.*

▶ **Exercise 9.4.** *Each $\mathbb{P}_m$ and each ${}^{\$}\mathbb{P}_m$ is a P2-ready structure.*

10 Stage I

Recall that untill the end of Section 13 we watch, analyse *Fugitive's* and *Crocodile's* play in shade filtering glasses. And we assume that *Fugitive* obeys Principle I and III.

▶ **Definition 10.1 (Crocodile's strategy).** *The sequence of languages $S = (l_1, l_2, \ldots, l_n)$, for some $n \in \mathbb{N}$, defines a strategy for Crocodile as follows: If $S = (l) +\!\!+ S'$ (where $+\!\!+$ denotes sequence concatenation) then Crocodile demands Fugitive to satisfy requests generated by l one by one (in any order) until (it can take infinitely many steps) there are no more requests generated by l in the current structure. Then*[7] *Crocodile proceeds with strategy S'.*

Now we define a set of strategies for *Crocodile*. All languages that will appear in these strategies are from $\mathcal{Q}_{good}$ so instead of writing Q^i_{good} we will just write i. Let:

- $S_{color} := (3, 4, 5, 6, 7, 8, 9)$,
- $S_{cycle} := (15, 14) +\!\!+ S_{color} +\!\!+ (12, 13) +\!\!+ S_{color}$,
- $S_{start} := (1, 2) +\!\!+ S_{cycle}$.

Recall that $\mathbb{D}_0$ is *Fugitive's* initial structure (consisting of green edges only), as demanded by Principle I.

▶ **Lemma 10.2.** Crocodile's *strategy* $(1, 2)$ *applied to the current structure* $\mathbb{D}_0$ *forces* Fugitive *to add* $R(\alpha^W)[a, a']$ *and* $R(\omega)[b', b]$.

The proof of this Lemma can be found in full paper.

A careful reader could ask here: "Why did we need to work so hard to prove that the newly added red edge must be *warm*. Don't we have Principle II which says that red edges

[6] Not all anonymous reviewers equally appreciated our decision to use the "exercise" environment in this paper. In our opinion proving some simple facts themselves, rather than skipping the proofs, can help the readers to develop intuitions needed to understand what is to come in the paper. We discussed this issue with several colleagues and none of them felt that using this environment is arrogant.

[7] For this "then" to make sense we need the total number of moves of the game to be ω^2 rather than ω.

must always be *warm* and green must be *cold*?". But we cannot use Principle II here: the structure is not *P2-ready* yet. Read the proof of Principle II again to notice that this red α^W between a and a' is crucial there. And this is what Stage I is all about: it is here where *Crocodile* forces *Fugitive* to construct a structure which is *P2-ready*. From now on all the current structures will be *P2-ready* and *Fugitive* will indeed be a slave of Principle II.

The following Lemma explains the role of S_{color} and is a first cousin of Observation 8.2:

▶ **Lemma 10.3 ($\mathbf{S_{color}}$).** *Strategy S_{color} applied to a P2-ready $\mathbb{D}$ forces* Fugitive *to create a P2-ready $\mathbb{D}'$ such that:*

- *Sets of vertices of $\mathbb{D}$ and $\mathbb{D}'$ are equal.*
- *There are no requests generated by Q^{1-9}_{good} in $\mathbb{D}'$, which means that each edge has its counterpart (incident to the same vertices) of the opposite color and temperature.*

The proof of this Lemma can be found in the full paper.

▶ **Lemma 10.4.** *Strategy S_{start} applied to $\mathbb{D}_0$ forces* Fugitive *to build $\mathbb{P}_1$.*

The proof of this Lemma can be found in the full paper.

11 Stage II

Note that from now on *Fugitive* must obey all Principles.

Now we imagine that $\mathbb{P}_1$ has already been created and we proceed with the analysis to the later stage of the Escape game where either $\mathbb{P}_{m+1}$ or ${}^{\$}\mathbb{P}_k$ for some $k \leq m$ will be created.

Let us define $\{S_k\}$ inductively for $k \in \mathbb{N}_+$ in the following fashion:

- $S_1 \coloneqq S_{start}$,
- $S_k \coloneqq S_{k-1} +\!\!+ S_{cycle}$ for $k > 1$.

▶ **Lemma 11.1.** *For all $m \in \mathbb{N}_+$ strategy S_m applied to $\mathbb{D}_0$ forces* Fugitive *to build a structure isomorphic, depending on his choice, either $\mathbb{P}_{m+1}$ or ${}^{\$}\mathbb{P}_k$ for some $k \leq m$.*

The proof of this Lemma can be found in the full paper.

12 The grids $\mathbb{G}_m$ and partial grids $\mathbb{L}^k_m$

▶ **Definition 12.1.** *$\mathbb{G}_m$, for $m \in \mathbb{N}_+$, is a directed graph (V, E) where:*

$V = \{a, a', b', b\} \cup \{v_{i,j} : i, j \in [0, m]\}$ and the edges E are labelled (as in $\mathbb{P}_m$) with $\Sigma \setminus \Sigma_0$ or one of the symbols of the form $(\mathbf{l}^t_o)$, which means that the shade filtering glasses are still on.

The edges of $\mathbb{G}_m$ are as follows:

- *Vertex a' is a successor of a, b is a successor of b'. All $v_{i,j}$ are successors of a' and the successors of each $v_{i,j}$ are $v_{i+1,j}, v_{i,j+1}$ (when they exist) and b'. From each node there are two edges to each of its successors, one red and one green. There are no other edges.*
- *Each* cold *edge, labelled with a symbol in $(\bullet^C)$, is green.*
- *Each* warm *edge, labelled with a symbol in $(\bullet^W)$, is red.*
- *Each edge $\langle v_{i,j}, v_{i+1,j}\rangle$ is horizontal – its label is from $(\bullet_H)$.*
- *Each edge $\langle v_{i,j}, v_{i,j+1}\rangle$ is vertical – its label is from $(\bullet_V)$.*
- *The label of each edge leaving $v_{i,j}$, with $i + j$ even, is from $(\mathbf{A})$, the label of each edge leaving $v_{i,j}$, with $i + j$ odd, is from $(\mathbf{B})$.*
- *Each edge $\langle a', v_i\rangle$ is labeled by either x^C or x^W.*
- *Each edge $\langle v_i, b'\rangle$ is labeled by either y^C or y^W.*

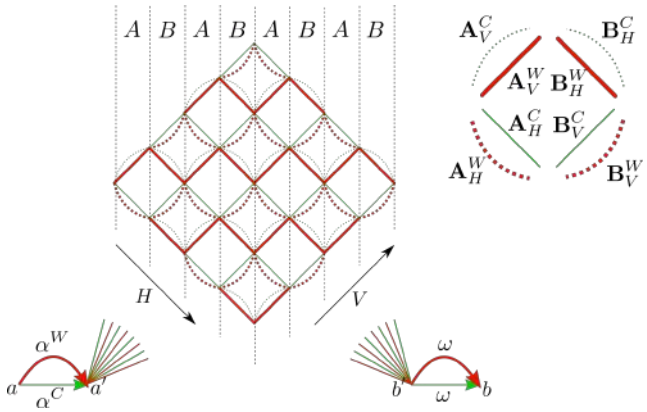


Figure 3 $\mathbb{G}_4$ (left). Smaller picture in the top-right corner explains how different line styles on the main picture map to Σ_0 (please use a color printer if you can).

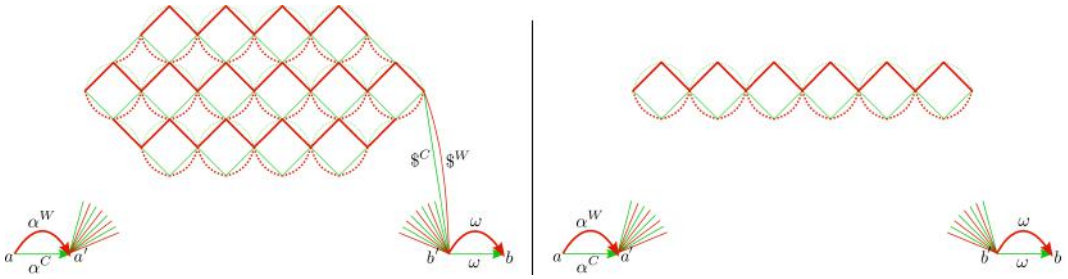


Figure 4 ${}^{\$}\mathbb{L}_6^3$ (left) and $\mathbb{L}_6^1$ (right).

- *Edges* $\langle a, a' \rangle$ *with label* $G(\alpha^C)$ *and* $\langle a, a' \rangle$ *with label* $R(\alpha^W)$ *are in* E.
- *Edges* $\langle b', b \rangle$ *with label* $G(\omega)$ *and* $\langle b', b \rangle$ *with label* $R(\omega)$ *are in* E.

▶ **Definition 12.2.** *Let* $\mathbb{L}_m^k = (V', E')$*, for* $m, k \in \mathbb{N}_+$ *where* $k \leq m$*, is a subgraph of* $\mathbb{G}_m = (V, E)$ *induced by the set* $V' \subseteq V$ *of vertices defined as* $V' = \{a, a', b', b\} \cup \{v_{i,j} : i, j \in [0, m]; \mid i - j \mid \leq k\}$.

▶ **Definition 12.3.** *Let* ${}^{\$}\mathbb{G}_m$ *for* $m \in \mathbb{N}_+$ *is* $\mathbb{G}_m$ *with two edges added:* $\langle v_{m,m}, b' \rangle$ *with label* $G(\$^C)$ *and* $\langle v_{m,m}, b' \rangle$ *with label* $R(\$^W)$.

▶ **Definition 12.4.** *Let* ${}^{\$}\mathbb{L}_m^k$ *for* $m \in \mathbb{N}_+, k \in \mathbb{N}_+ \cup \{0\}, k \leq m$ *is* $\mathbb{L}_m^k$ *with two edges added:* $\langle v_{m,m}, b' \rangle$ *with label* $G(\$^C)$ *and* $\langle v_{m,m}, b' \rangle$ *with label* $R(\$^W)$.

▷ Fact 12.5. For all m: $\mathbb{L}_m^m$ is equal to $\mathbb{G}_m$ and ${}^{\$}\mathbb{L}_m^m$ is equal to ${}^{\$}\mathbb{G}_m$.

▶ **Exercise 12.6.** *Languages from* $\mathcal{Q}_{good}$ *or* $\mathcal{Q}_{ugly}$ *do not generate requests in any* ${}^{\$}\mathbb{G}_m$.

13 Stage III

Now we imagine that either $\mathbb{P}_{m+1}$ or ${}^{\$}\mathbb{P}_k$ for some $k \leq m$ was created as the current position in a play of the game of Escape and we proceed with the analysis to the later stage of the play, where either $\mathbb{G}_{m+1}$ or ${}^{\$}\mathbb{G}_k$ will be created.

▶ **Lemma 13.1.** *For any* $m \in \mathbb{N}_+$ *Crocodile can force Fugitive to build a structure isomorphic, depending on Fugitive's choice, to either* $\mathbb{G}_{m+1}$ *or to* ${}^{\$}\mathbb{G}_k$ *for some* $k \leq m$.

Notice that by Exercise 12.5, in order to prove Lemma 13.1 it is enough to prove that for any $m \in \mathbb{N}_+$ *Crocodile* can force *Fugitive* to build a structure isomorphic to either $\mathbb{L}^{m+1}_{m+1}$ or to ${}^{\$}\mathbb{L}^k_k$ for some $k \leq m$.

As we said, we assume that *Crocodile* already forced *Fugitive* to build a structure isomorphic to either $\mathbb{P}_{m+1}$ or to ${}^{\$}\mathbb{P}_k$ for some $k \leq m$. Rename each v_i in this $\mathbb{P}_{m+1}$ (or ${}^{\$}\mathbb{P}_k$) as $v_{i,i}$. If the structure which was built is $\mathbb{P}_{m+1}$ we will show a strategy leading to $\mathbb{L}^{m+1}_{m+1}$ and when ${}^{\$}\mathbb{P}_k$ was built, we will show a strategy leading to ${}^{\$}\mathbb{L}^k_k$.

Now we define a sequence of strategies S^k_{layer}, which, similarly to strategies for building ${}^{\bullet}\mathbb{P}_{\bullet}$ consist only of languages from $\mathcal{Q}_{good}$, so instead of writing Q^i_{good} we will just write i.

Let:

- $S^{odd} \coloneqq (11) +\!\!+ S_{color} +\!\!+ (12, 13) +\!\!+ S_{color}$,
- $S^{even} \coloneqq (10) +\!\!+ S_{color} +\!\!+ (12, 13) +\!\!+ S_{color}$,
- $S^k_{layer} \coloneqq \begin{cases} [\,], & \text{if } k = 0 \\ S^{k-1}_{layer} +\!\!+ S^{odd} & \text{if } k \text{ odd} \\ S^{k-1}_{layer} +\!\!+ S^{even} & \text{otherwise} \end{cases}$

Proofs of the following four Lemmas can be found in the full paper.

▶ **Lemma 13.2.** *For all $k \in \mathbb{N}$ strategy S^1_{layer} applied to the current structure ${}^{\$}\mathbb{P}_k$ forces* Fugitive *to build ${}^{\$}\mathbb{L}^1_k$.*

▶ **Lemma 13.3.** *For all $k, m \in \mathbb{N}, k < m$ strategy S^{odd} (for $k+1$ odd) and S^{even} (for $k+1$ even) applied to ${}^{\$}\mathbb{L}^k_m$ forces* Fugitive *to build ${}^{\$}\mathbb{L}^{k+1}_m$.*

▶ **Lemma 13.4.** *For all $k, m \in \mathbb{N}, k < m$ strategy S^1_{layer} applied to $\mathbb{P}_k$ forces* Fugitive *to build $\mathbb{L}^1_k$, strategy S^{odd} (for $k+1$ odd) and S^{even} (for $k+1$ even) applied to $\mathbb{L}^k_m$ forces* Fugitive *to build $\mathbb{L}^{k+1}_m$.*

▶ **Lemma 13.5.** *For all $m \in \mathbb{N}$ strategy S^m_{layer} forces* Fugitive *to build ${}^{\$}\mathbb{L}^m_m$ from ${}^{\$}\mathbb{P}_m$ and $\mathbb{L}^m_m$ from $\mathbb{P}_m$.*

▶ **Observation 13.6.** *By Exercise 12.5 Lemma 13.5 proves Lemma 13.1.*

14 And now we finally see the shades again

Now we are ready to finish the proof of Lemma 5.7. First assume the original instance of Our Grid Tiling Problem has no *proper shading*. The following is straightforward from König's Lemma by noticing that if there were arbitrary grids with proper shading, then there would be an infinite one:

▶ **Lemma 14.1.** *If an instance I of OGTP has no proper shading then there exist natural m such that for any $k \geq m$ a square grid of size k has no shading that satisfies conditions (a1), (a2), (b1) and (b3) of proper shading.*

Let m be the value from Lemma 14.1. By Lemma 13.1 *Crocodile* can force *Fugitive* to build a structure isomorphic to either $\mathbb{G}_{m+1}$ or ${}^{\$}\mathbb{G}_k$ for some $k \leq m$. Now suppose the play ended, in some final position $\mathbb{H}$ isomorphic to one of these structures. We take off our glasses, and not only we still see this $\mathbb{H}$, but now we also see the shades, with each edge (apart from edges labeled with α, ω, x, y and \$) having one of the shades from $\mathcal{S}$. Now concentrate on the red edges labeled with $(\bullet^W)$ of $\mathbb{H}$. They form a grid, with each vertical edge labeled with V, each horizontal edge labeled with H, and with each edge labeled with a shade from $\mathcal{S}$. Now we consider two cases:

- If $\mathbb{G}_{m+1}$ was built then clearly condition (b3) of Definition 5.5 is unsatisfied. But this implies that a path labeled with a word from one of the languages Q_{bad} occurs in $\mathbb{H}$ between a and b, which is in breach with Principle III because of language Q^1_{bad}.
- If ${}^{\$}\mathbb{G}_k$ for $k \leq m$ was built then clearly condition (b2) or (b3) of Definition 5.5 is unsatisfied. This is because we assumed that there is no *proper shading*. But this implies that a path labeled with a word from one of the languages Q_{bad} occurs in $\mathbb{H}$ between a and b, which is in breach with Principle III because of language Q^1_{bad}.

This ends the proof of Lemma 5.7 (ii).

For the proof of Lemma 5.7 (i) assume the original instance $\langle \mathcal{S}, \mathcal{F} \rangle$ of Our Grid Tiling Problem has a *proper shading* – a labeled grid of side length m. Call this grid $\mathbb{G}$.

Recall that ${}^{\$}\mathbb{G}_m$ satisfies all regular constraints from $\mathcal{Q}^{\leftrightarrow}_{good}$ and from $\mathcal{Q}^{\leftrightarrow}_{ugly}$ (Exercise 12.6). Now copy the shades of the edges of $\mathbb{G}$ to the respective edges of ${}^{\$}\mathbb{G}_m$. Call this new structure (${}^{\$}\mathbb{G}_m$ with shades added) $\mathbb{M}$. It is easy to see that $\mathbb{M}$ constitutes a counterexample, as in Lemma 4.1.

References

1. Serge Abiteboul and Victor Vianu. Regular Path Queries with Constraints. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '97, pages 122–133, New York, NY, USA, 1997. ACM. doi:10.1145/263661.263676.
2. Foto N. Afrati. Determinacy and query rewriting for conjunctive queries and views. *Theoretical Computer Science*, 412(11):1005–1021, 2011. doi:10.1016/j.tcs.2010.12.031.
3. Pablo Barceló. Querying graph databases. In *PODS*, 2013.
4. Peter Buneman, Wenfei Fan, and Scott Weinstein. Path Constraints in Semistructured Databases. *Journal of Computer and System Sciences*, 61(2):146–193, 2000. doi:10.1006/jcss.2000.1710.
5. D. Calvanese, G. de Giacomo, M. Lenzerini, and M. Y. Vardi. View-based query processing and constraint satisfaction. In *Proceedings Fifteenth Annual IEEE Symposium on Logic in Computer Science (Cat. No.99CB36332)*, pages 361–371, June 2000. doi:10.1109/LICS.2000.855784.
6. D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Answering regular path queries using views. In *Proceedings of 16th International Conference on Data Engineering (Cat. No.00CB37073)*, pages 389–398, February 2000. doi:10.1109/ICDE.2000.839439.
7. Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the Decidability of Query Containment Under Constraints. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '98, pages 149–158, New York, NY, USA, 1998. ACM. doi:10.1145/275487.275504.
8. Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Rewriting of Regular Expressions and Regular Path Queries. In *Proceedings of the Eighteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '99, pages 194–204, New York, NY, USA, 1999. ACM. doi:10.1145/303976.303996.
9. Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Lossless Regular Views. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, pages 247–258, New York, NY, USA, 2002. ACM. doi:10.1145/543613.543646.
10. Mariano P. Consens and Alberto O. Mendelzon. GraphLog: a Visual Formalism for Real Life Recursion. In *PODS*, 1990.
11. Isabel F. Cruz, Alberto O. Mendelzon, and Peter T. Wood. A Graphical Query Language Supporting Recursion. In *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*, SIGMOD '87, pages 323–330, New York, NY, USA, 1987. ACM. doi:10.1145/38713.38749.

12 Wenfei Fan, Floris Geerts, and Lixiao Zheng. View determinacy for preserving selected information in data transformations. *Inf. Syst.*, 37:1–12, 2012.

13 Daniela Florescu, Alon Levy, and Dan Suciu. Query Containment for Conjunctive Queries with Regular Expressions. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '98, pages 139–148, New York, NY, USA, 1998. ACM. doi:10.1145/275487.275503.

14 Nadime Francis. *Vues et requetes sur les graphes de donnees: determinabilite et reecritures*. PhD thesis, Ecole Normale Superieure de Cacha, 2015.

15 Nadime Francis. Asymptotic Determinacy of Path Queries Using Union-of-Paths Views. *Theory of Computing Systems*, 61:156–190, 2016.

16 Grzegorz Gluch, Jerzy Marcinkowski, and Piotr Ostropolski-Nalewaja. Can One Escape Red Chains?: Regular Path Queries Determinacy is Undecidable. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pages 492–501, New York, NY, USA, 2018. ACM. doi:10.1145/3209108.3209120.

17 Tomasz Gogacz and Jerzy Marcinkowski. The Hunt for a Red Spider: Conjunctive Query Determinacy Is Undecidable. In *Proceedings of the 2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, LICS '15, pages 281–292, Washington, DC, USA, 2015. IEEE Computer Society. doi:10.1109/LICS.2015.35.

18 Tomasz Gogacz and Jerzy Marcinkowski. Red Spider Meets a Rainworm: Conjunctive Query Finite Determinacy Is Undecidable. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '16, pages 121–134, New York, NY, USA, 2016. ACM. doi:10.1145/2902251.2902288.

19 V. Juge and Moshe Vardi. On the containment of Datalog in Regular Datalog. Technical report, Rice University, 2009.

20 Per-Åke Larson and H. Z. Yang. Computing Queries from Derived Relations. In *Proceedings of the 11th International Conference on Very Large Data Bases - Volume 11*, VLDB '85, pages 259–269. VLDB Endowment, 1985. URL: http://dl.acm.org/citation.cfm?id=1286760.1286784.

21 Alan Nash, Luc Segoufin, and Victor Vianu. Determinacy and Rewriting of Conjunctive Queries Using Views: A Progress Report. In Thomas Schwentick and Dan Suciu, editors, *Database Theory – ICDT 2007*, pages 59–73, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

22 Alan Nash, Luc Segoufin, and Victor Vianu. Views and Queries: Determinacy and Rewriting. *ACM Trans. Database Syst.*, 35(3):21:1–21:41, July 2010. doi:10.1145/1806907.1806913.

23 Daniel Pasailă. Conjunctive Queries Determinacy and Rewriting. In *Proceedings of the 14th International Conference on Database Theory*, ICDT '11, pages 220–231, New York, NY, USA, 2011. ACM. doi:10.1145/1938551.1938580.

24 Juan L. Reutter, Miguel Romero, and Moshe Y. Vardi. Regular Queries on Graph Databases. *Theor. Comp. Sys.*, 61(1):31–83, July 2017. doi:10.1007/s00224-016-9676-2.

25 Hartley Rogers, Jr. *Theory of Recursive Functions and Effective Computability*. MIT Press, Cambridge, MA, USA, 1987.

26 Moshe Y. Vardi. A Theory of Regular Queries. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '16, pages 1–9, New York, NY, USA, 2016. ACM. doi:10.1145/2902251.2902305.

Datalog: Bag Semantics via Set Semantics

Leopoldo Bertossi
RelationalAI Inc., USA
Carleton University, Ottawa, Canada
Member of the "Millenium Institute for Foundational Research on Data" (IMFD, Chile)
bertossi@scs.carleton.ca

Georg Gottlob
University of Oxford, UK
TU Wien, Austria
georg.gottlob@cs.ox.ac.uk

Reinhard Pichler
TU Wien, Austria
pichler@dbai.tuwien.ac.at

Abstract

Duplicates in data management are common and problematic. In this work, we present a translation of Datalog under bag semantics into a well-behaved extension of Datalog, the so-called *warded Datalog*$^{\pm}$, under set semantics. From a theoretical point of view, this allows us to reason on bag semantics by making use of the well-established theoretical foundations of set semantics. From a practical point of view, this allows us to handle the bag semantics of Datalog by powerful, existing query engines for the required extension of Datalog. This use of Datalog$^{\pm}$ is extended to give a set semantics to duplicates in Datalog$^{\pm}$ itself. We investigate the properties of the resulting Datalog$^{\pm}$ programs, the problem of deciding multiplicities, and expressibility of some bag operations. Moreover, the proposed translation has the potential for interesting applications such as to Multiset Relational Algebra and the semantic web query language SPARQL with bag semantics.

2012 ACM Subject Classification Information systems → Query languages; Theory of computation → Logic; Theory of computation → Semantics and reasoning

Keywords and phrases Datalog, duplicates, multisets, query answering, chase, Datalog$^{\pm}$

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.16

Related Version A full version of the paper is available at `http://arxiv.org/abs/1803.06445`.

Acknowledgements Many thanks to Renzo Angles and Claudio Gutierrez for information on their work on SPARQL with bag semantics; and to Wolfgang Fischl for his help testing some queries in SQL DBMSs. We appreciate the useful comments received from the reviewers. Part of this work was done while L. Bertossi was spending a sabbatical at the DBAI Group of TU Wien with support from the "Vienna Center for Logic and Algorithms" and the Wolfgang Pauli Society. This author is grateful for their support and hospitality, and specially to G. Gottlob for making the stay possible. He was also supported by NSERC Discovery Grant #06148. The work of G. Gottlob was supported by the Austrian Science Fund (FWF):P30930 and the EPSRC programme grant EP/M025268/1 VADA. The work of R. Pichler was supported by the Austrian Science Fund (FWF):P30930.

1 Introduction

Duplicates are a common feature in data management. They appear, for instance, in the result of SQL queries over relational databases or when a SPARQL query is posed over RDF data. However, the semantics of data operations and queries in the presence of duplicates is not always clear, because duplicates are handled by bags or multisets, but common logic-based semantics in data management are set-theoretical, making it difficult to tell apart duplicates

through the use of sets alone. To address this problem, a bag semantics for Datalog programs was proposed in [21], what we refer to as the *derivation-tree bag semantics* (DTB semantics). Intuitively, two duplicates of the same tuple in an intentional predicate are accepted as such if they have syntactically different derivation trees. The DTB semantics was used in [2] to provide a bag semantics for SPARQL.

The DTB semantics follows a proof-theoretic approach, which requires external, meta-level reasoning over the set of all derivation trees rather than allowing for a query language that inherently collects those duplicates. The main goal of this paper is to identify a syntactic class of extended Datalog programs, such that: (a) it extends classical Datalog with stratified negation and has a classical model-based semantics, (b) for every program in the class with a bag semantics, another program in the same class can be built that has a set-semantics and fully captures the bag semantics of the initial program, (c) it can be used in particular to give a set-semantics for classical Datalog with stratified negation with bag semantics.

To this end, we show that the DTB semantics of a Datalog program can be represented by means of its transformation into a Datalog$^{\pm}$ program [8, 10], in such a way that the intended model of the former, including duplicates, can be characterized as the result of the duplicate-free chase instance for the latter. The crucial idea of our translation from bag semantics into set semantics (of Datalog$^{\pm}$) is the introduction of tuple ids (tids) via existentially quantified variables in the rule heads. Different tids of the same tuple will allow us to identify usual duplicates when falling back to a bag semantics for the original Datalog program. We establish the correspondence between the DTB semantics and ours. This correspondence is then extended to Datalog with stratified negation. We thus recover full relational algebra (including set difference) with bag semantics in terms of a well-behaved query language under set semantics.

The programs we use for this task belong to *warded* Datalog$^{\pm}$ [17]. This is a particularly well-behaved class of programs in that it properly extends Datalog, has a tractable conjunctive query answering (CQA) problem, and has recently been implemented in a powerful query engine, namely the VADALOG System [6, 7]. None of the other well-known classes of Datalog$^{\pm}$ share these properties: for instance, guarded [8], sticky and weakly-sticky [11] Datalog$^{\pm}$ only allow restricted forms of joins and, hence, do not cover Datalog. On the other hand, more expressive languages, such as weakly frontier guarded Datalog$^{\pm}$ [5], lose tractability of CQA. Warded Datalog$^{\pm}$ has been successfully applied to represent a core fragment of SPARQL under certain OWL 2 QL entailment regimes [16], with set semantics though [17] (see also [3, 4]), and it looks promising as a general language for specifying different data management tasks [6]. We then go one step further and also express the bag semantics of Datalog$^{\pm}$ by means of the set semantics of Datalog$^{\pm}$.

Structure and main results. In Section 2, we recall some basic notions. In Section 7, we conclude and discuss some future extensions. Our main results are detailed in Sections 3 – 6:

- Our translation of Datalog with bag semantics into warded Datalog$^{\pm}$ with set semantics, which will be referred to as program-based bag (PBB) semantics, is presented in Section 3. We also show how this translation can be extended to Datalog with stratified negation.
- In Section 4, we study the transformation from bag semantics into set semantics for Datalog$^{\pm}$ itself. We thus carry over both the DTB semantics and the PBB semantics to Datalog$^{\pm}$ with a form of stratified negation, and establish the equivalence of these two semantics also for this extended query language. Moreover, we verify that the Datalog$^{\pm}$ programs resulting from our transformation are warded whenever the programs to start

with belong to this class.
- In Section 5, we study crucial decision problems related to multiplicities. Above all, we are interested in the question if a given tuple has finite or infinite multiplicity. Moreover, in case of finiteness, we want to compute the precise multiplicity. We show that these tasks can be accomplished in polynomial time (data complexity).
- In Section 6, we apply our results on Datalog with bag semantics to Multiset Relational Algebra (MRA). We also discuss alternative semantics for multiset-intersection and multiset-difference, and the difficulties to capture them with our Datalog$^{\pm}$ approach.

2 Preliminaries

We assume familiarity with the relational data model, conjunctive queries (CQs), in particular Boolean conjunctive queries (BCQs); classical Datalog with minimal-model semantics, and Datalog with stratified negation with standard-model semantics, denoted Datalog$^{\neg s}$ (see [1] for an introduction). An n-ary relational predicate P has *positions*: $P[1], \ldots, P[n]$. With $Pos(P)$ we denote the set of positions of predicate P; and with $Pos(\Pi)$ the set of positions of (predicates in) a program Π.

2.1 Derivation-Tree Bag (DTB) Semantics for Datalog and Datalog$^{\neg s}$

We follow [21], where tuples are *colored* to tell apart duplicates of a same element in the extensional database (EDB), via an infinite, ordered list $\mathcal{C}$ of colors $c_1, c_2, \ldots$. For a multiset M, $e \in M$ if $mult(e, M) > 0$ holds, where $mult(e, M)$ denotes the multiplicity of e in M. In this case, the n copies of e are denoted by $e{:}1, \ldots, e{:}n$, indicating that they are colored with $c_1, \ldots, c_n$, respectively. So, $col(e) := \{e{:}1, \ldots, e{:}n\}$ becomes a set. A multiset M_1 is (multi)contained in multiset M_2, when $mult(e, M_1) \leq mult(e, M_2)$ for every $e \in M_1$. For a multiset M, $col(M) := \bigcup_{e \in M} col(e)$, which is a set. For a "colored" set S, $col^{-1}(S)$ produces a multiset by stripping tuples from their colors.

▶ **Example 1.** For $M = \{a, a, a, b, b, c\}$, $col(a) = \{a{:}1, a{:}2, a{:}3\}$, and $col(M) = S = \{a{:}1, a{:}2, a{:}3, b{:}1, b{:}2, c{:}1\}$. The inverse operation, the decoloration, gives, for instance: $col^{-1}(a{:}2) := a$; and $col^{-1}(S) := \{a, a, a, b, b, c\}$, a multiset.

We consider Datalog programs Π with multiset predicates and multiset EDBs D. A *derivation tree* (DT) for Π wrt. D is a tree with labeled nodes and edges, as follows:

1. For an EDB predicate P and $h \in col(P(D))$, a DT for $col^{-1}(h)$ contains a single node with label h.
2. For each rule of the form $\rho: \; H \leftarrow A_1, A_2, \ldots, A_k;$ with $k > 0$, and each tuple $\langle \mathcal{T}_1, \ldots, \mathcal{T}_k \rangle$ of DTs for the atoms $\langle d_1, \ldots, d_k \rangle$ that unify with $\langle A_1, A_2, \ldots, A_k \rangle$ with mgu θ, generate a DT for $H\theta$ with $H\theta$ as the root label, $\langle \mathcal{T}_1, \ldots, \mathcal{T}_k \rangle$ as the children, and ρ as the label for the edges from the root to the children. We assume that these children are arranged in the order of the corresponding body atoms in rule ρ.

For a DT $\mathcal{T}$, we define $Atoms(\mathcal{T})$ as col^{-1} of the root when $\mathcal{T}$ is a single-node tree, and the root-label of $\mathcal{T}$, otherwise. For a set of DTs $\mathfrak{T}$: $Atoms(\mathfrak{T}) := \biguplus\{Atoms(\mathcal{T}) \mid \mathcal{T} \in \mathfrak{T}\}$, which is a multiset containing D. Here, we write $\biguplus$ to denote the multi-union of (multi)sets that keeps all duplicates. If $DT(\Pi, col(D))$ is the set of (syntactically different) DTs, the *derivation-tree bag (DTB) semantics* for Π is the multiset $DTBS(\Pi, D) := Atoms(DT(\Pi, col(D)))$ (cf. [18] for an equivalent, provenance-based bag semantics for Datalog.)

▶ **Example 2.** Consider the program $\Pi = \{\rho_1, \rho_2, \rho_3\}$ and multiset EDB $D = \{Q(1,2,3), Q(1,2,5), Q(2,3,4), Q(2,3,4), T(4,1,2), T(4,1,2)\}$, where ρ_1, ρ_2, ρ_3 are defined as follows:

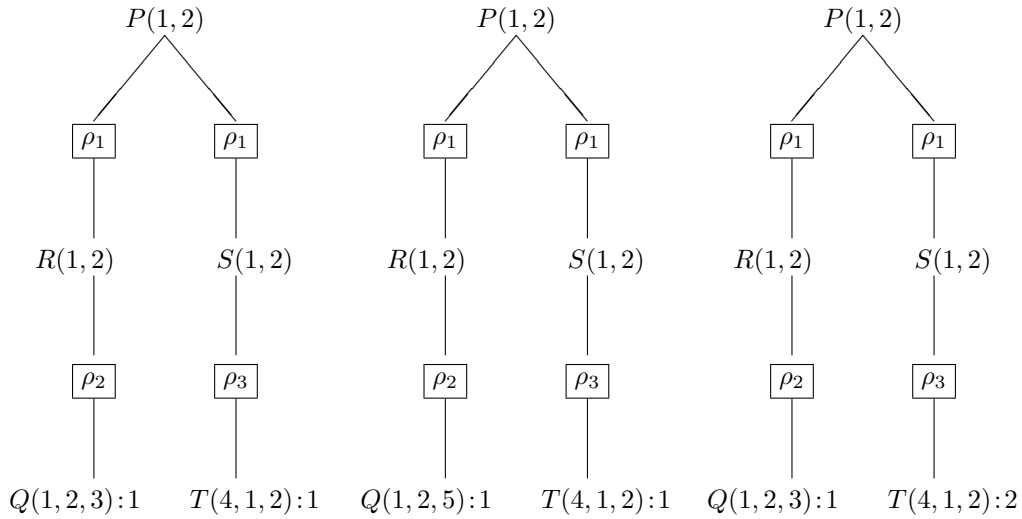


Figure 1 Three (out of four) derivation-trees for $P(1,2)$ in Example 2.

$\rho_1\colon\ P(x,y) \leftarrow R(x,y), S(x,y); \quad \rho_2\colon\ R(x,y) \leftarrow Q(x,y,z); \quad \rho_3\colon\ S(x,y) \leftarrow T(z,x,y).$

Here, $\mathit{col}(D) = \{Q(1,2,3){:}1, Q(1,2,5){:}1, Q(2,3,4){:}1, Q(2,3,4){:}2, T(4,1,2){:}1, T(4,1,2){:}2\}$. In total, we have 16 DTs: (a) 6 single-node trees with labels in $\mathit{col}(D)$ (b) 6 depth-two, linear trees (root to the left, i.e. rotated in $-90°$): $R(1,2) - \rho_2 - Q(1,2,3){:}1$. $R(1,2) - \rho_2 - Q(1,2,5){:}1$. $R(2,3) - \rho_2 - Q(2,3,4){:}1$. $R(2,3) - \rho_2 - Q(2,3,4){:}2$. $S(1,2) - \rho_3 - T(4,1,2){:}1$. $S(1,2) - \rho_3 - T(4,1,2){:}2$. (c) 4 depth-three trees for $P(1,2)$, three of which are displayed in Figure 1. The 16 different DTs in $\mathit{DT}(\Pi, \mathit{col}(D))$ give rise to $\mathit{DTBS}(\Pi, D) = D \cup \{R(1,2), R(1,2), R(2,3), R(2,3), S(1,2), S(1,2), P(1,2), P(1,2), P(1,2), P(1,2)\}$.

In [22], a bag semantics for Datalog$^{\neg s}$ was introduced via derivation-trees (DTs), extending the DTB semantics in [21] for (positive) Datalog. This extension applies to Datalog programs with stratified negation that are range-restricted and safe, i.e. a variable in a rule head or in a negative literal must appear in a positive literal in the body of the same rule. If Π is a Datalog$^{\neg s}$ program with multiset predicates and multiset EDB D, a *derivation tree* for Π wrt. D is as for Datalog programs, but with condition **2.** modified as follows:

2′. Now let ρ be a rule of the form $\rho\colon\ H \leftarrow A_1, A_2, \ldots, A_k, \neg B_1, \ldots \neg B_\ell$; with $k > 0$ and $\ell \geq 0$. Let the predicate of H be of some stratum i and let the predicates of $B_1, \ldots, B_\ell$ be of some stratum $< i$. Assume that we have already computed all derivation trees for (atoms with) predicates up to stratum $i-1$. Then, for each tuple $\langle \mathcal{T}_1, \ldots, \mathcal{T}_k\rangle$ of DTs for the atoms $\langle d_1, \ldots, d_k\rangle$ that unify with $\langle A_1, A_2, \ldots, A_k\rangle$ with mgu θ, such that there is no DT for any of the atoms $B_1\theta, \ldots, B_\ell\theta$, generate a DT for $H\theta$ with $H\theta$ as the root label and $\langle \mathcal{T}_1, \ldots, \mathcal{T}_k, \neg B_1\theta, \ldots, \neg B_\ell\theta\rangle$ as the children, in this order. Furthermore, all edges from the root to its children are labelled with ρ.

Analogously to the positive case, now for a range-restricted and safe program Π in Datalog$^{\neg s}$ and multiset EDB D, we write $\mathit{DTBS}(\Pi, D)$ to denote the derivation-tree based bag semantics.

▶ **Example 3** (ex. 2 cont.)**.** Consider now the EDB $D' = \{Q(1,2,3), Q(1,2,5), Q(2,3,4), Q(2,3,4), T(4,1,2)\}$ (with one duplicate of $T(4,1,2)$ removed from D), and modify Π to $\Pi' = \{\rho_1', \rho_2, \rho_3\}$ with $\rho_1'\colon\ P(x,y) \leftarrow R(x,y), \neg S(x,y)$ (i.e., ρ_1' now encodes multiset difference). Then, predicates Q, R, S, T are on stratum 0 and P is on stratum 1. The DTs

for atoms with predicates from stratum 0 are as in Example 2 with two exceptions: there is now only one single-node DT for $T(4,1,2)$ and only one DT for $S(1,2)$.

For ground atoms with predicate P, we now only get two DTs producing $P(2,3)$ – derived via two copies of $R(2,3)$, which in turn stem from the two copies of $Q(2,3,4)$ in the EDB. In contrast, $P(1,2)$ cannot be derived from either of the copies of $R(1,2)$ obtained from either $Q(1,2,3)$ or $Q(1,2,5)$ in the EDB because $S(1,2)$ has a DT.

In total, we have 12 different DTs in $DT(\Pi', \mathit{col}(D'))$ with $DTBS(\Pi', D') = D' \cup \{R(1,2), R(1,2), R(2,3), R(2,3), S(1,2), P(2,3), P(2,3)\}$. Notice that the DT semantics interprets the difference in rule ρ_1' as "all or nothing": when computing $P(x,y)$, a single DT for $S(x,y)$ "kills" all the DTs for $R(x,y)$ (cf. Section 6). For example, $P(1,2)$ is not derived despite the fact that we have two copies of $R(1,2)$ and only one of $S(1,2)$.

2.2 Warded Datalog$^{\pm}$

Datalog$^{\pm}$ was introduced in [9] as an extension of Datalog, where the "+" stands for the new ingredients, namely: tuple-generating dependencies (*tgds*), written as *existential rules* of the form $\exists \bar{z} P(\bar{x}', \bar{z}) \leftarrow P_1(\bar{x}_1), \ldots, P_n(\bar{x}_n)$, with $\bar{x}' \subseteq \bigcup_i \bar{x}_i$, and $\bar{z} \cap \bigcup_i \bar{x}_i = \emptyset$; as well as equality-generating dependencies (*egds*) and negative constraints. In this work we ignore egds and constraints. The "−" in Datalog$^{\pm}$ stands for syntactic restrictions on rules to ensure decidability of CQ answering.

We consider three sets of term symbols: $\mathbf{C}$, $\mathbf{B}$, and $\mathbf{V}$ containing constants, labelled nulls (also known as blank nodes in the semantic web context), and variables, respectively. Let T denote an atom or a set of atoms. We write $\mathit{var}(T)$ and $\mathit{nulls}(T)$ to denote the set of variables and nulls, respectively, occurring in T. In a DB, typically an EDB D, all terms are from $\mathbf{C}$. In an *instance*, we also allow terms to be from $\mathbf{B}$. For a rule ρ, $\mathit{body}(\rho)$ denotes the set of atoms in its body, and $\mathit{head}(\rho)$, the atom in its head. A homomorphism h from a set X of atoms to a set X' of atoms is a partial function $h\colon \mathbf{C} \cup \mathbf{B} \cup \mathbf{V} \to \mathbf{C} \cup \mathbf{B} \cup \mathbf{V}$ such that $h(c) = c$ for all $c \in \mathbf{C}$ and $P(h(t_1), \ldots, h(t_k)) \in X'$ for every atom $P(t_1, \ldots, t_k) \in X$.

We say that a rule $\rho \in \Pi$ is *applicable* to an instance I if there exists a homomorphism h from $\mathit{body}(\rho)$ to I. In this case, the result of applying ρ to I is an instance $I' = I \cup \{h'(\mathit{head}(\rho))\}$, where h' coincides with h on $\mathit{var}(\mathit{body}(\rho))$ and h' maps each existential variable in the head of ρ to a fresh labelled null not occurring in I. For such an application of ρ to I, we write $I\langle \rho, h \rangle I'$. Such an application of ρ to I is called a *chase step*. The chase is an important tool in the presence of existential rules. A *chase sequence* for a DB D and a Datalog$^{\pm}$ program Π is a sequence of chase steps $I_i\langle \rho_i, h_i \rangle I_{i+1}$ with $i \geq 0$, such that $I_0 = D$ and $\rho_i \in \Pi$ for every i (also denoted $I_i \rightsquigarrow_{\rho_i, h_i} I_{i+1}$). For the sake of readability, we sometimes only write the newly generated atoms of each chase step without repeating the old atoms. Also the subscript ρ_i, h_i is omitted if it is clear from the context. A chase sequence then reads $I_0 \rightsquigarrow A_1 \rightsquigarrow A_2 \rightsquigarrow A_3 \rightsquigarrow \ldots$ with $I_{i+1} \setminus I_i = \{A_{i+1}\}$.

The final atoms of all possible chase sequences for DB D and Π form an instance referred to as $\mathit{Chase}(D, \Pi)$, which can be infinite. We denote the result of all chase sequences up to length d for some $d \geq 0$ as $\mathit{Chase}^d(D, \Pi)$. The chase variant assumed here is the so-called *oblivious chase* [8, 19], i.e., if a rule $\rho \in \Pi$ ever becomes applicable with some homomorphism h, then $\mathit{Chase}(D, \Pi)$ contains exactly one atom of the form $h'(\mathit{head}(\rho))$ such that h' extends h to the existential variables in the head of ρ. Intuitively, each rule is applied exactly once for every applicable homomorphism.

Consider a DB D and a Datalog$^{\pm}$ program Π (the former the EDB for the latter). As a logical theory, $\Pi \cup D$ may have multiple models, but the model $M = \mathit{Chase}(D, \Pi)$ turns out to be a correct representative for the class of models: for every BCQ $\mathcal{Q}$, $\Pi \cup D \models \mathcal{Q}$

iff $M \models Q$ [14]. There are classes of Datalog$^{\pm}$ that, even with an infinite chase, allow for decidable or even tractable CQA in the size of the EDB. Much effort has been made in identifying and characterizing interesting syntactic classes of programs with this property (see [8] for an overview). In this direction, *warded* Datalog$^{\pm}$ was introduced in [3, 4, 17], as a particularly well-behaved fragment of Datalog$^{\pm}$, for which CQA is tractable. We briefly recall and illustrate it here, for which we need some preliminary notions.

A position p in Datalog$^{\pm}$ program Π is *affected* if: (a) an existential variable appears in p, or (b) there is $\rho \in \Pi$ such that a variable x appears in p in $\mathit{head}(\rho)$ and all occurrences of x in $\mathit{body}(\rho)$ are in affected positions. $\mathit{Aff}(\Pi)$ and $\mathit{NonAff}(\Pi)$ denote the sets of affected, resp. non-affected, positions of Π. Intuitively, $\mathit{Aff}(\Pi)$ contains all positions where the chase may possibly introduce a null.

▶ **Example 4.** Consider the following program:

$$\exists z_1\ R(y_1, z_1) \leftarrow P(x_1, y_1) \tag{1}$$

$$\exists z_2\ P(x_2, z_2) \leftarrow S(u_2, x_2, x_2), R(u_2, y_2) \tag{2}$$

$$\exists z_3\ S(x_3, y_3, z_3) \leftarrow P(x_3, y_3), U(x_3). \tag{3}$$

By the first case, $R[2], P[2], S[3]$ are affected. By the second case, $R[1], S[2]$ are affected. Now that $S[2], S[3]$ are affected, also $P[1]$ is. We thus have $\mathit{Aff}(\Pi) = \{P[1], P[2], R[1], R[2], S[2], S[3]\}$ and $\mathit{NonAff}(\Pi) = \{S[1], U[1]\}$.

For a rule $\rho \in \Pi$, and a variable x: (a) $x \in \mathit{body}(\rho)$ is *harmless* if it appears at least once in $\mathit{body}(\rho)$ at a position in $\mathit{NonAff}(\Pi)$. $\mathit{Harmless}(\rho)$ denotes the set of harmless variables in ρ. Otherwise, a variable is called *harmful*. Intuitively, harmless variables will always be instantiated to constants in any chase step, whereas harmful variables may be instantiated to nulls. (b) $x \in \mathit{body}(\rho)$ is *dangerous* if $x \notin \mathit{Harmless}(\rho)$ and $x \in \mathit{head}(\rho)$. $\mathit{Dang}(\rho)$ denotes the set of dangerous variables in ρ. These are the variables which may propagate nulls into the atom created by a chase step.

▶ **Example 5** (ex. 4 cont.)**.** x_1 and y_1 are both harmful but only y_1 is dangerous for (1); u_2 is harmless, x_2 is dangerous, y_2 is harmful but not dangerous for (2); x_3 is harmless and y_3 is dangerous for (3).

Now, a rule $\rho \in \Pi$ is *warded* if either $\mathit{Dang}(\rho) = \emptyset$ or there exists an atom $A \in \mathit{body}(\rho)$, the ward, such that (1) $\mathit{Dang}(\rho) \subseteq \mathit{var}(A)$ and (2) $\mathit{var}(A) \cap \mathit{var}(\mathit{body}(\rho) \setminus \{A\}) \subseteq \mathit{Harmless}(\rho)$. A program Π is *warded* if every rule $\rho \in \Pi$ is warded.

▶ **Example 6** (ex. 5 cont.)**.** Rule (1) is trivially warded with the single body atom as the ward. Rule (2) is warded by $S(u_2, x_2, x_2)$: variable x_2 is the only dangerous variable and u_2 (the only variable shared by the ward with the rest of the body) is harmless. Actually, the other body atom $R(u_2, y_2)$ contains the harmful variable y_2; but it is not dangerous and not shared with the ward. Finally, rule (3) is warded by $P(x_3, y_3)$; the other atom $U(x_3)$ contains no affected variable. Since all rules are warded, the program Π is warded.

Datalog$^{\pm}$ can be extended with safe, stratified negation in the usual way, similarly as stratified Datalog [1]. The resulting Datalog$^{\pm,\neg s}$ can also be given a chase-based semantics [10]. The notions of affected/non-affected positions and harmless/harmful/dangerous variables carry over to a Datalog$^{\pm,\neg s}$ program Π by considering only the program Π^{pos} obtained from Π by deleting all negated body atoms. For warded Datalog$^{\pm}$, only a restricted form of stratified negation is allowed – so-called *stratified ground* negation. This means that we

require for every rule $\rho \in \Pi$: if ρ contains a negated atom $P(t_1, \ldots, t_n)$, then every t_i must be either a constant (i.e, $t_i \in \mathbf{C}$) or a harmless variable. Hence, negated atoms can never contain a null in the chase. We write Datalog$^{\pm,\neg sg}$ for programs in this language.

The class of warded Datalog$^{\pm,\neg sg}$ programs extends the class of Datalog$^{\neg s}$ programs. Warded Datalog$^{\pm,\neg sg}$ is expressive enough to capture query answering of a core fragment of SPARQL under certain OWL 2 QL entailment regimes [16], and this task can actually be accomplished in polynomial time (data complexity) [3, 4]. Hence, Datalog$^{\pm,\neg sg}$ constitutes a very good compromise between expressive power and complexity. Recently, a powerful query engine for warded Datalog$^{\pm,\neg sg}$ has been presented [6, 7], namely the VADALOG system.

3 Datalog$^{\pm}$-Based Bag Semantics for Datalog$^{\neg s}$

We now provide a set-semantics that represents the bag semantics for a Datalog program Π with a multiset EDB D via the transformation into a Datalog$^{\pm}$ program Π^+ over a set EDB D^+ obtained from D. For this, we assume w.l.o.g., that the set of *nulls*, $\mathbf{B}$, for a Datalog$^{\pm}$ program is partitioned into two infinite ordered sets $\mathcal{I} = \{\iota_1, \iota_2, \ldots\}$, for unique, global tuple identifiers (tids), and $\mathcal{N} = \{\eta_1, \eta_2, \ldots\}$, for usual nulls in Datalog$^{\pm}$ programs. Given a multiset EDB D and a program Π, instead of using colors and syntactically different derivation trees, we will use elements of $\mathcal{I}$ to identify both the elements of the EDB and the tuples resulting from applications of the chase.

For every predicate $P(\ldots)$, we introduce a new version $P(.\,;\ldots)$ with an extra, first argument (its 0-th position) to accommodate a tid, which is a null from $\mathcal{I}$. If an atom $P(\bar{a})$ appears in D as n duplicates, we create the tuples $P(\iota'_1; \bar{a}), \ldots, P(\iota'_n; \bar{a})$, with the ι'_i pairwise different nulls from $\mathcal{I}$ as tids, and not used to identify any other element of D. We obtain a *set* EDB D^+ from the multiset EDB D. Given a rule in Π, we introduce tid-variables (i.e. appearing in the 0-th positions of predicates) and existential quantifiers in the rule head, to formally generate fresh tids when the rule applies. More precisely, a rule in Π of the form $\rho\colon\ H(\bar{x}) \leftarrow A_1(\bar{x}_1), A_2(\bar{x}_2), \ldots, A_k(\bar{x}_k)$, with $k > 0$, $\bar{x} \subseteq \cup_i \bar{x}_i$, becomes the Datalog$^{\pm}$ rule $\rho^+\colon\ \exists z\, H(z; \bar{x}) \leftarrow A_1(z_1; \bar{x}_1), A_2(z_2; \bar{x}_2), \ldots, A_k(z_k; \bar{x}_k)$, with fresh, different variables $z, z_1, \ldots, z_k$. The resulting Datalog$^{\pm}$ program Π^+ can be evaluated according to the usual *set semantics* on the set EDB D^+ via the chase: when the instantiated body $A_1(\iota'_1; \bar{a}_1)$, $A_2(\iota'_2; \bar{a}_2), \ldots, A_k(\iota'_k; \bar{a}_k)$ of rule ρ^+ becomes true, then the new tuple $H(\iota; \bar{a})$ is created, with ι the first (new) null from $\mathcal{I}$ that has not been used yet, i.e., the tid of the new atom.

▶ **Example 7** (ex. 2 cont.). The EDB D from Example 2 becomes

$$D^+ = \{Q(\iota_1; 1, 2, 3),\ Q(\iota_2; 1, 2, 5), Q(\iota_3; 2, 3, 4), Q(\iota_4; 2, 3, 4), T(\iota_5; 4, 1, 2), T(\iota_6; 4, 1, 2)\};^{1}$$

and program Π becomes $\Pi^+ = \{\rho_1^+, \rho_2^+, \rho_3^+\}$ with $\rho_1^+\colon\ \exists u P(u; x, y) \leftarrow R(u_1; x, y), S(u_2; x, y)$; $\rho_2^+\colon\ \exists u R(u; x, y) \leftarrow Q(u_1; x, y, z)$; $\rho_3^+\colon\ \exists u S(u; x, y) \leftarrow T(u_1; z, x, y)$.

The following is a 3-step chase sequence of D^+ and Π^+: $\{Q(\iota_1; 1, 2, 3), T(\iota_4; 4, 1, 2)\} \rightsquigarrow_{\rho_2^+} R(\iota_7; 1, 2) \rightsquigarrow_{\rho_3^+} S(\iota_{11}; 1, 2) \rightsquigarrow_{\rho_1^+} P(\iota_{13}; 1, 2)$.

Analogously to the depth-two and depth-three trees in Example 2, the chase produces 10 new atoms. In total, we get: $\mathit{Chase}(\Pi^+, D^+) = D^+ \cup \{R(\iota_7; 1, 2),\ R(\iota_8; 1, 2),\ R(\iota_9; 2, 3),\ R(\iota_{10}; 2, 3),\ S(\iota_{11}; 1, 2),\ S(\iota_{12}; 1, 2),\ P(\iota_{13}; 1, 2),\ P(\iota_{14}; 1, 2),\ P(\iota_{15}; 1, 2),\ P(\iota_{16}; 1, 2)\}$.

[1] Notice that this set version of D can also be created by means of Datalog$^{\pm}$ rules. For example, with the rule $\exists z Q(z; x, y, v) \leftarrow Q(x, y, v)$ for the EDB predicate Q.

In order to extend the PBB Semantics to Datalog$^{\neg s}$, we have to extend our transformation of programs Π into Π^+ to rules with negated atoms. Consider a rule ρ of the form:

$$\rho\colon\ H(\bar{x})\ \leftarrow\ A_1(\bar{x}_1),\ldots,A_k(\bar{x}_k),\neg B_1(\bar{x}_{k+1}),\ldots,\neg B_\ell(\bar{x}_{k+\ell}), \tag{4}$$

with, $\bar{x}_{k+1},\ldots,\bar{x}_{k+\ell}\subseteq\bigcup_{i=1}^{k}\bar{x}_i$; we transform it into the following two rules:

$$\begin{aligned}\rho^+\colon\ \exists z H(z;\bar{x}) \quad &\leftarrow \quad A_1(z_1;\bar{x}_1),\ldots,A_k(z_k;\bar{x}_k),\neg Aux_1^\rho(\bar{x}_{k+1}),\ldots,\neg Aux_\ell^\rho(\bar{x}_{k+\ell}),\\ Aux_i^\rho(\bar{x}_{k+i}) \quad &\leftarrow \quad B_i(z_i;\bar{x}_{k+i}),\quad i=1,\ldots,\ell.\end{aligned} \tag{5}$$

The introduction of auxiliary predicates Aux_i is crucial since adding fresh variables directly to the negated atoms would yield negated atoms of the form $\neg B_i(z_{k+i};\bar{x}_{k+i})$ in the rule body, which make the rule unsafe. The resulting Datalog$^{\pm,\neg s}$ program is from the desired class Datalog$^{\pm,\neg sg}$:

▶ **Theorem 8.** *Let Π be a Datalog$^{\neg s}$ program and let Π^+ be the transformation of Π into a Datalog$^{\pm,\neg s}$ program. Then, Π^+ is a warded Datalog$^{\pm,\neg sg}$ program.*

Operation col^{-1} of Section 2 inspires de-identification and multiset merging operations. Sometimes we use double braces, $\{\{\ldots\}\}$, to emphasize that the operation produces a *multiset*.

▶ **Definition 9.** *For a set D of tuples with tids, $\mathcal{DI}(D)$ and $\mathcal{SP}(D)$, for de-identification and set-projection, respectively, are: (a) $\mathcal{DI}(D) := \{\{P(\bar{c}) \mid P(t;\bar{c})\in D \text{ for some } t\}\}$, a multiset; and (b) $\mathcal{SP}(D) := \{P(\bar{c}) \mid P(t;\bar{c})\in D \text{ for some } t\}$, a set.*

▶ **Definition 10.** *Given a Datalog$^{\neg s}$ program Π and a multiset EDB D, the* program-based bag semantics *(PBB semantics) assigns to $\Pi\cup D$ the multiset:*

$$PBBS(\Pi,D) := \mathcal{DI}(Chase(\Pi^+,D^+)) = \{\{P(\bar{a}) \mid P(t;\bar{a})\in Chase(\Pi^+,D^+)\}\}.$$

The main results in this section are the correspondence of PBB semantics and DTB semantics and the relationship of both with classical set semantics of Datalog:

▶ **Theorem 11.** *For a Datalog$^{\neg s}$ program Π with a multiset EDB D, $DTBS(\Pi,D) = PBBS(\Pi,D)$ holds.*

Proof Idea. The theorem is proved by establishing a one-to-one correspondence between DTs in $DT(\Pi, col(D))$ with a fixed root atom $P(\bar{c})$ and (minimal) chase-derivations of $P(\bar{c})$ from D^+ via Π^+. This proof proceeds by induction on the depth of the DTs and length of the chase sequences. ◀

▶ **Corollary 12.** *Given a Datalog (resp. Datalog$^{\neg s}$) program Π and a multiset EDB D, the set $\mathcal{SP}(Chase(\Pi^+,D^+))$ is the minimal model (resp. the standard model) of the program $\Pi\cup\mathcal{SP}(D)$.*

4 Bag Semantics for Datalog$^{\pm,\neg sg}$

In the previous section, we have seen that warded Datalog$^{\pm}$ (possibly extended with stratified ground negation) is well suited to capture the bag semantics of Datalog in terms of classical set semantics. We now want to go one step further and study the *bag semantics* for Datalog$^{\pm,\neg sg}$ itself. Note that this question makes a lot of sense given the results from [4], where it is shown that warded Datalog$^{\pm,\neg sg}$ captures a core fragment of SPARQL under certain OWL2 QL entailment regimes and the official W3C semantics of SPARQL is a bag semantics.

4.1 Extension of the DTB Semantics to Datalog$^{\pm,\neg sg}$

The definition of a DT-based bag semantics for Datalog$^{\pm,\neg sg}$ is not as straightforward as for Datalog$^{\neg s}$, since atoms in a model of $D \cup \Pi$ for a (multiset) EDB D and Datalog$^{\pm,\neg sg}$ program Π may have labelled nulls as arguments, which correspond to existentially quantified variables and may be arbitrarily chosen. Hence, when counting duplicates, it is not clear whether two atoms differing only in their choice of nulls should be treated as copies of each other or not. We therefore treat multiplicities of atoms analogously to multiple answers to single-atom queries, i.e., the multiplicity of an atom $P(t_1, \ldots, t_k)$ wrt. EDB D and program Π corresponds to the multiplicity of answer $(t_1, \ldots, t_k)$ to the query $Q = P(x_1, \ldots, x_k)$ over the database D and program Π. In other words, we ask for all instantiations $(t_1, \ldots, t_k)$ of $(x_1, \ldots, x_k)$ such that $P(t_1, \ldots, t_k)$ is true in *every* model of $D \cup \Pi$. It is well known that only ground instantiations (on $\mathbf{C}$) can have this property (see e.g. [14]). Hence, below, we restrict ourselves to considering duplicates of ground atoms containing constants from $\mathbf{C}$ only (in this section we are not using tid-positions 0). In the rest of this section, unless otherwise stated, "ground atom" means instantiated on $\mathbf{C}$; and programs belong to Datalog$^{\pm,\neg sg}$.

In order to define the multiplicity of a ground atom $P(t_1, \ldots, t_k)$ wrt. a (multiset) EDB D and a warded Datalog$^{\pm,\neg sg}$ program Π, we adopt the notion of *proof tree* used in [4, 11], which generalizes the notion of derivation tree to Datalog$^{\pm,\neg sg}$. We consider first positive Datalog$^{\pm}$ programs. A proof tree (PT) for an atom A (possibly with nulls) wrt. (a set) EDB D and Datalog$^{\pm}$ program Π is a node- and edge-labelled tree with labelling function λ, such that: (1) The nodes are labelled by atoms over $\mathbf{C} \cup \mathbf{B}$. (2) The edges are labelled by rules from Π. (3) The root is labelled by A. (4) The leaf nodes are labelled by atoms from D. (5) The edges from a node N to its child nodes $N_1, \ldots, N_k$ are all labelled with the same rule ρ. (6) The atom labelling N corresponds to the result of a chase step where $\mathit{body}(\rho)$ is instantiated to $\{\lambda(N_1), \ldots, \lambda(N_k)\}$ and $\mathit{head}(\rho)$ becomes $\lambda(N)$ when instantiating the existential variables of $\mathit{head}(\rho)$ with fresh nulls. (7) If M' (resp. N') is the parent node of M (resp. N) such that $\mathit{nulls}(\lambda(M')) \setminus \mathit{nulls}(\lambda(M))$ and $\mathit{nulls}(\lambda(N')) \setminus \mathit{nulls}(\lambda(N))$ share at least one null, then the entire subtrees rooted at M' and at N' must be isomorphic (i.e., the same tree structure and the same labels). (8) If, for two nodes M and N, $\lambda(M)$ and $\lambda(N)$ share a null v, then there exist ancestors M' of M and N' of N such that M' and N' are siblings corresponding to two body atoms A and B of rule ρ with $x \in \mathit{var}(A)$ and $x \in \mathit{var}(B)$ for some variable x and ρ is applied with some substitution γ which sets $\gamma(x) = v$; moreover, v occurs in the labels of the entire paths from M' to M and from N' to N. A proof tree for Example 13 below is shown in Figure 2, left. As with derivation trees, we assume that siblings in the proof tree are arranged in the order of the corresponding body atoms in the rule ρ labelling the edge to the parent node (cf. Section 2.1).

Intuitively, a PT is a tree-representation of the derivation of an atom by the chase. The parent/child relationship in a PT corresponds to the head/body of a rule application in the chase. Condition (7) above refers to the case that a non-ground atom is used in two different rule applications within the chase sequence. In this case, the two occurrences of this atom must have identical proof sub-trees. A PT can be obtained from a chase-derivation by *reversing the edges and unfolding the chase graph into a tree by copying some of the nodes* [4]. By definition of the chase in Section 2.2, it can never happen that the same null is created by two different chase steps. Note that the nulls in $\mathit{nulls}(\lambda(M')) \setminus \mathit{nulls}(\lambda(M))$ (and, likewise in $\mathit{nulls}(\lambda(N')) \setminus \mathit{nulls}((\lambda(N)))$ are precisely the newly created ones. Hence, if $\lambda(M')$ and $\lambda(N')$ share such a null, then $\lambda(M')$ and $\lambda(N')$ are the same atom and the subtrees rooted at these nodes are obtained by unfolding the same subgraph of the chase graph. Condition (8) makes sure that we use the same null v in a PT only if this is required by a join condition of some

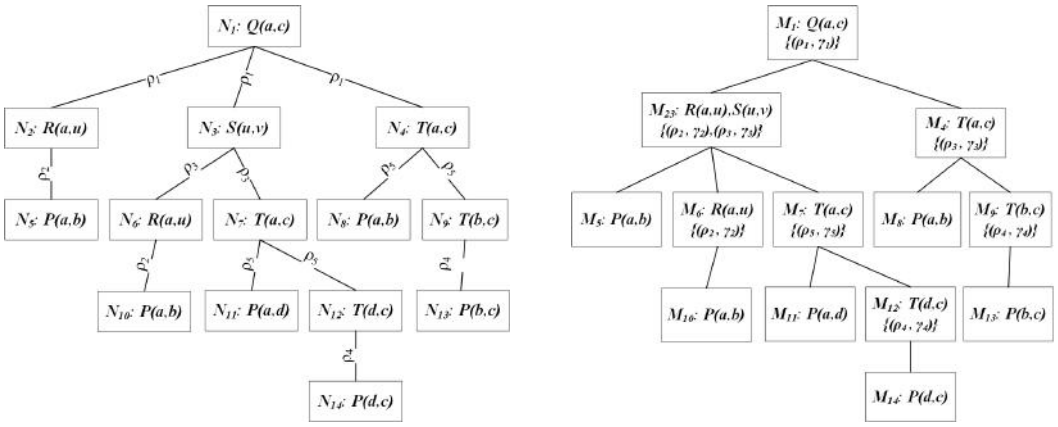


Figure 2 Proof tree (left) and witness (right) for atom $Q(a, c)$ in Examples 13 and 20.

rule ρ; otherwise nulls are renamed apart.

▶ **Example 13.** Let $\Pi = \{\rho_1, \ldots, \rho_5\}$ be the Datalog$^{\pm}$ program with ρ_1: $Q(x, w) \leftarrow R(x, y), S(y, z), T(x, w)$; ρ_2 : $\exists z\ R(x, z) \leftarrow P(x, y)$; ρ_3 : $\exists z\ S(x, z) \leftarrow R(w, x), T(w, y)$; ρ_4 : $T(x, y) \leftarrow P(x, y)$; ρ_5: $T(x, y) \leftarrow P(x, z), T(z, y)$. This program belongs to Datalog$^{\pm,\neg sg}$, and – although not necessary to build a proof tree for it – we notice that it is also warded: $\mathit{Aff}(\Pi) = \{R[2], S[2], S[1]\}$; and all other positions are not affected. Rule ρ_3 is warded with ward $R(w, x)$ (where x is the only dangerous variable in this rule). All other rules are trivially warded because they have no dangerous variables.

Now let $D = \{P(a, b), P(b, c), P(a, d), P(d, c)\}$. A possible proof tree for $Q(a, c)$ is shown in Figure 2 on the left. It is important to note that nodes N_2 and N_6 introducing labelled null u are labelled with the same atom and give rise to identical subtrees. Of course, $R(a, u)$ could also result from a chase step applying ρ_2 to $P(a, d)$. However, this would generate a null different from u and, subsequently, the nulls in $R(a, \_)$ and $S(\_, v)$ (in N_2 and N_3) would be different, and rule ρ_1 could not be applied anymore. In contrast, the nodes N_4 and N_7 with label $T(a, c)$ span different subtrees. This is desired: there are two possible derivations for each occurrence of atom $T(a, c)$ in the PT.

Here we deviate slightly from the definition of PTs in [4], in that we allow the same *ground* atom to have different derivations. This is needed to detect duplicates and to make sure that PTs in fact constitute a generalization of the derivation trees in Section 2.1. Moreover, condition (8) is needed to avoid non-isomorphic PTs by "unforced" repetition of nulls (i.e., identical nulls that are not required by a join condition further up in the tree). Analogous to the generalization in Section 2.1 of DTs for Datalog to DTs for Datalog$^{\neg s}$, it is easy to generalize proof trees to Datalog$^{\pm,\neg sg}$. Here it is important that we only allow stratified *ground* negation. Hence, analogously to DTs for Datalog$^{\neg s}$, we allow negated ground atoms $\neg A$ to occur as node labels of leaf nodes in a PT, provided that the positive atom A has no PT. Moreover, it is no problem to allow also multiset EDBs since, as in Section 2.1, we can keep duplicates apart by means of a coloring function *col*.

Finally, we can define proof trees $\mathcal{T}$ and $\mathcal{T}'$ as equivalent, denoted $\mathcal{T} \equiv \mathcal{T}'$, if one is obtained from the other by renaming of nulls. We can thus normalize PTs by assuming that nulls in node labels are from some fixed set $\{u_1, u_2 \ldots\}$ and that these nulls are introduced in the labels of the PT by some fixed-order traversal (e.g., top-down, left-to-right).

For a PT $\mathcal{T}$, we define $\mathit{Atoms}(\mathcal{T})$ as col^{-1} of the root when $\mathcal{T}$ is a single-node tree, and

the root-label of $\mathcal{T}$, otherwise. For a set of PTs $\mathfrak{T}$: $\mathit{Atoms}(\mathfrak{T}) := \biguplus\{\mathit{Atoms}(\mathcal{T}) \mid \mathcal{T} \in \mathfrak{T}\}$, which is a multiset that multi-contains D. If $\mathit{PT}(\Pi, \mathit{col}(D))$ is the set of normalized, *non-equivalent* PTs, the *proof-tree bag (PTB) semantics* for Π is the multiset $\mathit{PTBS}(\Pi, D) := \mathit{Atoms}(\mathit{PT}(\Pi, \mathit{col}(D)))$. For a ground atom A, $\mathit{mult}(A, \mathit{PTBS}(\Pi, D))$ denotes the multiplicity of A in the multiset $\mathit{PTBS}(\Pi, D)$.

▶ **Example 14** (ex. 13 cont.)**.** To compute $\mathit{PTBS}(\Pi, D)$ for Π and D from Example 13, we have to determine all proof trees of all ground atoms derivable from Π and D. In Figure 2, we have already seen one proof tree for $Q(a, c)$; and we have observed that the sub-proof tree of $R(a, u)$ rooted at nodes N_2 and N_6 could be replaced by a child node with label $P(a, d)$ (either both subtrees or none has to be changed). In total, the ground atom $Q(a, c)$ has 8 different proof trees wrt. Π and D (multiplying the 2 possible derivations of atom $R(a, u)$ with 4 derivations for the two occurrences of the atom $T(a, c)$ in nodes N_4 and N_7). The other ground atoms derivable from Π and D are $T(a, c)$ (with 2 possible PTs as discussed in Example 13) and the atoms $T(i, j)$ for each $P(i, j)$ in D. Hence, we have $\mathit{PTBS}(\Pi, D) = D \cup \{T(a, b), T(b, c), T(a, d), T(d, a), T(a, c), T(a, c), Q(a, c), \ldots, Q(a, c)\}$.

Clearly, every Datalog$^{\neg s}$ program is a special case of a warded Datalog$^{\pm,\neg sg}$ program. It is easy to verify that the PTB semantics indeed generalizes the DTB semantics:

▶ **Proposition 15.** Let Π be a Datalog$^{\neg s}$ program and D an EDB (possibly with duplicates). Then $\mathit{DTBS}(\Pi, D) = \mathit{PTBS}(\Pi, D)$ holds.

4.2 Extension of the PBB Semantics to Datalog$^{\pm,\neg sg}$

We now extend also the PBB semantics to Datalog$^{\pm,\neg sg}$ programs Π. First, in all predicates of Π, we add position 0 to carry tid-variables. Then every rule $\rho \in \Pi$ of the form $\rho:\ \exists\bar{y} H(\bar{y}, \bar{x}) \leftarrow A_1(\bar{x}_1), A_2(\bar{x}_2), \ldots, A_k(\bar{x}_k)$, with $k > 0$, $\bar{x} \subseteq \cup_i \bar{x}_i$, becomes the rule $\rho^+:\ \exists z \exists\bar{y}\ H(z; \bar{y}, \bar{x}) \leftarrow A_1(z_1; \bar{x}_1), A_2(z_2; \bar{x}_2), \ldots, A_k(z_k; \bar{x}_k)$, with fresh, different variables $z, z_1, \ldots, z_k$. Now consider a rule $\rho \in \Pi$ of the form $\rho:\ \exists\bar{y} H(\bar{y}\bar{x}) \leftarrow A_1(\bar{x}_1), \ldots, A_k(\bar{x}_k), \neg B_1(\bar{x}_{k+1}), \ldots, \neg B_\ell(\bar{x}_{k+\ell})$, with $\bar{x}_{k+1}, \ldots, \bar{x}_{k+\ell} \subseteq \bigcup_{i=1}^{k} \bar{x}_i$; we transform it into the following two rules:

$$\begin{aligned} \rho':\ \exists z \exists\bar{y} H(z; \bar{y}, \bar{x}) \quad &\leftarrow \quad A_1(z_1; \bar{x}_1), \ldots, A_k(z_k; \bar{x}_k), \neg \mathit{Aux}_1^\rho(\bar{x}_{k+1}), \ldots, \neg \mathit{Aux}_\ell^\rho(\bar{x}_{k+\ell}), \quad (6) \\ \mathit{Aux}_i^\rho(\bar{x}_{k+i}) \quad &\leftarrow \quad B_i(z_i; \bar{x}_{k+i}), \quad i = 1, \ldots, \ell. \end{aligned}$$

Analogously to Theorem 8, the resulting program also belongs to Datalog$^{\pm,\neg sg}$. Finally, as in Section 3, the ground atoms in the multiset EDB D are extended in D^+ by nulls from $\mathcal{I} = \{\iota_1, \iota_2, \ldots\}$ as tids in position 0 to keep apart duplicates. For an instance I, we write $I_\downarrow$ to denote the set of atoms in I such that all positions except for position 0 (the tid) carry a ground term (from $\mathbf{C}$). Analogously to Definition 10, we then define the *program-based bag semantics* (PBB semantics) of a Datalog$^{\pm,\neg sg}$ program Π and multiset EDB D as $\mathit{PBBS}(\Pi, D) := \mathcal{DI}(\mathit{Chase}(\Pi^+, D^+)_\downarrow) = \{\{P(\bar{a}) \mid P(\bar{a}) \text{ is ground and } P(t; \bar{a}) \in \mathit{Chase}(\Pi^+, D^+)\}\}$. For a ground atom A (here, without nulls or tids), $\mathit{mult}(A, \mathit{PBBS}(\Pi, D))$ denotes the multiplicity of A in the multiset $\mathit{PBBS}(\Pi, D)$.

▶ **Example 16** (ex. 13 and 14 cont.)**.** The transformations of Datalog$^{\pm,\neg sg}$ program Π and multiset EDB D from Example 13 are $D^+ = \{P(\iota_1, a, b), P(\iota_2, b, c), P(\iota_3, a, d), P(\iota_4, d, c)\}$ and $\Pi^+ = \{\rho_1^+, \rho_2^+, \rho_3^+, \rho_4^+, \rho_5^+\}$ with: $\rho_1^+:\ (\exists u) Q(u, x, w) \leftarrow R(v_1, r, y), S(v_2, y, z), T(v_3, x, w)$; $\rho_2^+:\ (\exists u, z) R(u, x, z) \leftarrow P(v, x, y)$; $\rho_3^+:\ (\exists u, z) S(u, x, z) \leftarrow R(v_1, w, x), T(v_2, w, y)$; $\rho_4^+:\ (\exists u) T(u, x, y) \leftarrow P(v, x, y)$; $\rho_5^+:\ (\exists u) T(u, x, y) \leftarrow P(v_1, x, z), T(v_2, z, y)$.

Applying program Π^+ to instance D^+ gives, for example, the following chase sequence for deriving the atoms $T(\iota_5, d, c), T(\iota_7, a, c), T(\iota_8, b, c), Q(\iota_{11}, a, c)$, which correspond to the ground atoms in the proof tree in Figure 2, left: $D^+ \rightsquigarrow_{\rho_4^+} T(\iota_5, d, c) \rightsquigarrow_{\rho_2^+} R(\iota_6, a, u) \rightsquigarrow_{\rho_5^+} T(\iota_7, a, c) \rightsquigarrow_{\rho_4^+} T(\iota_8, b, c) \rightsquigarrow_{\rho_3^+} S(\iota_9, u, v) \rightsquigarrow_{\rho_5^+} T(\iota_{10}, a, c) \rightsquigarrow_{\rho_1^+} Q(\iota_{11}, a, c)$. Implicitly, we thus also turn the PTs rooted at N_4, N_7, and N_9 into chase sequences. In total, we get in $PBBS(\Pi, D)$ precisely the atoms and multiplicities as in $PTBS(\Pi, D)$ for Example 14.

In principle, the above transformation Π^+ and the PBB semantics are applicable to any program Π in Datalog$^{\pm,\neg sg}$. However, in the first place, we are interested in *warded* programs Π. It is easy to verify that wardedness of Π carries over to Π^+:

▶ **Proposition 17.** If Π is a warded Datalog$^{\pm,\neg sg}$ program, then so is Π^+.

Proof Idea. The key observation is that the only additional affected positions in Π^+ are the tids at position 0. However, the variables at these positions occur only once in each rule and are never propagated from the body to the head. Hence, they do not destroy wardedness. ◀

We conclude this section with the analogous result of Theorem 11:

▶ **Theorem 18.** *For ground atoms A, $mult(A, PTBS(\Pi, D)) = mult(A, PBBS(\Pi, D))$. Hence, for a warded Datalog$^{\pm,\neg sg}$ program Π and multiset EDB D, $PTBS(\Pi, D) = PBBS(\Pi, D)$.*

5 Decidability and Complexity of Multiplicity

For Datalog$^{\neg s}$ programs, the following problems related to duplicates have been investigated:

- FFE: Given a program Π, a database D, and a predicate P, decide if every derivable ground P-atom has a finite number of DTs.
- FEE: Given a program Π and a predicate P, decide if every derivable ground P-atom has a finite number of DTs *for every* database D.

It has been shown in [22] that FFE is decidable (even in PTIME data complexity), whereas FEE is undecidable. We extend this study by considering warded Datalog$^{\pm,\neg sg}$ instead of Datalog$^{\neg s}$, and by computing the concrete multiplicities in case of finiteness. We thus study the following problems:

- FINITENESS: For fixed warded Datalog$^{\pm,\neg sg}$ program Π: Given a multiset database D and a ground atom A, does A have finite multiplicity, i.e., is $mult(A, PBBS(\Pi, D))$ finite?
- MULTIPLICITY: For fixed warded Datalog$^{\pm,\neg sg}$ program Π: Given a multiset database D and a ground atom A, compute the multiplicity of A, i.e. $mult(A, PBBS(\Pi, D))$.

We will show that both problems defined above can be solved in polynomial time (data complexity). In case of the FINITENESS problem, we thus generalize the result of [22] for the FFE problem from Datalog$^{\neg s}$ to Datalog$^{\pm,\neg sg}$. In case of the MULTIPLICITY problem, no analogous result for Datalog or Datalog$^{\neg s}$ has existed before. The following example illustrates that, even if Π is a Datalog program with a single rule, we may have exponential multiplicities. Hence, simply computing all DTs or (in case of Datalog$^{\pm}$) all PTs is not a viable option if we aim at polynomial time complexity.

▶ **Example 19.** Let $D = \{E(a_0, a_1), E(a_1, a_2), \ldots, E(a_{n-1}, a_n)\} \cup \{P(a_0, a_1), C(b_0), C(b_1)\}$ for $n \geq 1$ and let $\Pi = \{\rho\}$ with $\rho\colon P(x, y) \leftarrow P(x, z), E(z, y), C(w)$. Intuitively, E can be considered as an edge relation and P is the corresponding path relation for paths starting at a_0. The C-atom in the rule body (together with the two C-atoms in D) has the effect that there are always 2 possible derivations to extend a path. It can be verified by induction over

i, that atom $P(a_0, a_i)$ has 2^{i-1} possible derivations from D via Π. In particular, $P(a_0, a_n)$ has multiplicity 2^{n-1}.

Note that, if we add atom $E(a_1, a_1)$ to D (i.e., a self-loop, so to speak), then every atom $P(a_0, a_i)$ with $i \geq 1$ has infinite multiplicity. Intuitively, the infinitely many different derivation trees correspond to the arbitrary number of cycles through the self-loop $E(a_1, a_1)$ for a path from a_0 to a_i.

Our PTIME-membership results will be obtained by appropriately adapting the tractability proof of CQA for warded Datalog$^{\pm}$ in [4], which is based on the algorithm ProofTree for deciding if $D \cup \Pi \models P(\bar{c})$ holds for database D, warded Datalog$^{\pm}$ program Π, and ground atom $P(\bar{c})$. That algorithm works in ALOGSPACE (data complexity), i.e., alternating logspace, which coincides with PTIME [12]. It assumes Π to be normalized in such a way that each rule in Π is either *head-grounded* (i.e., each term in the head is a constant or a harmless variable) or *semi-body-grounded* (i.e., there exists at most one body atom with harmful variables). Algorithm ProofTree starts with ground atom $P(\bar{c})$ and applies resolution steps until the database D is reached. It thus proceeds as follows:

- If $P(\bar{c}) \in D$, then accept. Otherwise, guess a head-grounded rule $\rho \in \Pi$ whose head can be matched to $P(\bar{c})$ (denoted as $\rho \rhd P(\bar{c})$). Guess an instantiation γ on the variables in the body of ρ so that $\gamma(\mathit{head}(\rho)) = P(\bar{c})$. Let $\mathcal{S} = \gamma(\mathit{body}(\rho))$.
- Partition $\mathcal{S}$ into $\{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$, such that each null occurring in $\mathcal{S}$ occurs in exactly one $\mathcal{S}_i$, and each set $\mathcal{S}_i$ is chosen subset-minimal with this property. The purpose of these sets $\mathcal{S}_i$ of atoms is to keep together, in the parallel universal computations of ProofTree, the nulls in $\mathcal{S}$ until the atom in which they are created is known.
- Universally select each set $\mathcal{S}' \in \{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$ and "prove" it: If $\mathcal{S}'$ consists of a single ground atom $P'(\bar{c}')$, then call ProofTree recursively for D, Π, $P'(\bar{c}')$. Otherwise, do the following:
  (1) For each atom $A \in \mathcal{S}'$, guess rule ρ_A with $\rho_A \rhd A$ and guess variable instantiation γ_A on the variables in the body of ρ_A such that $A = \gamma_A(\mathit{head}(\rho_A))$.
  (2) The set $\bigcup_{A \in \mathcal{S}'} \gamma_A(\mathit{body}(\rho_A))$ is partitioned as above and each component of this partition is proved in a parallel universal computation.

The key to the ALOGSPACE complexity of algorithm ProofTree is that the data structure propagated by this algorithm fits into logarithmic space. This data structure is given by a pair $(\mathcal{S}, \mathcal{R}_\mathcal{S})$, where $\mathcal{S}$ is a set of atoms (such that $|\mathcal{S}|$ is bounded by the maximum number of body atoms of the rules in Π) and $\mathcal{R}_\mathcal{S}$ is a set of pairs (z, x), where z is a null occurring in $\mathcal{S}$ and x is either an atom A (meaning that null z was created when the application of some rule ρ generated the atom A containing this null z) or the symbol ε (meaning that we have not yet found such an atom A). A *witness* for a successful computation of ProofTree is then given by a tree with existential and universal nodes, with an existential node N consisting of a pair $(\mathcal{S}, \mathcal{R}_\mathcal{S})$; a universal node N indicates the guessed rule ρ_A together with the instantiation γ_A for each atom $A \in \mathcal{S}$ at the (existential) parent node of N. The child nodes of each universal node N are obtained by partitioning $\bigcup_{A \in \mathcal{S}} \gamma_A(\mathit{body}(A))$ as described above and computing the corresponding set $\mathcal{R}_\mathcal{S}$. At the root, we thus have an existential node labelled $(\mathcal{S}, \mathcal{R}_\mathcal{S}) = (\{P(\bar{c})\}, \emptyset)$. Each leaf node is a universal node labelled with $(B, \emptyset)$ for some ground atom $B \in D$. Hence, such a node corresponds to an accept-state.

▶ **Example 20.** Recall Π and D from Example 13. A proof tree of ground atom $Q(a, c)$ is shown in Figure 2 on the left. On the right, we display the witness of the corresponding successful computation of ProofTree. Note that witnesses have a strict alternation of existential and universal nodes. In the witness in Figure 2, we have merged each existential

node and its unique universal child node to make the correspondence between a PT and a witness yet more visible. In particular, on each depth level of the trees, we have exactly the same set of atoms (with the only difference, that in the witness these atoms are grouped together to sets $\mathcal{S}$ by null-occurrences). In this simple example, only node M_{23} contains such a group of atoms, namely the atoms from the nodes N_2 and N_3 in the PT.

In order not to overburden the figure, we have left out the sets $\mathcal{R}_\mathcal{S}$ carrying the information on the introduction of nulls. In this simple example, the only node with a non-empty set $\mathcal{R}_\mathcal{S}$ is M_6, with $\mathcal{R}_\mathcal{S} = \{(u, R(a, u)\}$, i.e. we have to pass on the information that the null u in node M_{23} was introduced by applying some rule (namely ρ_2) which generated the atom $R(a, u)$. We thus ensure (when proceeding from node M_6 to node M_{10}) that null u is introduced in the same way as before.

The information from the universal node below each existential node is displayed in the second line of each node: here we have pairs consisting of the guessed rule ρ plus the guessed instantiation γ for each atom in the corresponding set $\mathcal{S}$ (as mentioned above, $\mathcal{S}$ is a singleton in all nodes except for M_{23}). For instance, γ_1 in node M_1 denotes the substitution $\gamma_1 = \{x \leftarrow a, y \leftarrow u, z \leftarrow v, w \leftarrow c\}$. Only in node M_{23}, we have 2 pairs $(\rho_2, \gamma_2), (\rho_3, \gamma_3)$ with $\gamma_2 = \{x \leftarrow a, y \leftarrow b\}$ and $\gamma_3 = \{w \leftarrow a, x \leftarrow u, y \leftarrow c\}$. Note that the subscripts i of the γ_i's in Figure 2 are to be understood local to the node. For instance, the two different applications of rule ρ_4 in nodes M_9 and M_{12} are with the instantiations $\gamma_4 = \{x \leftarrow b, y \leftarrow c\}$ (in M_9) and $\gamma_4 = \{x \leftarrow d, y \leftarrow c\}$ (in M_{12}), respectively.

The above example illustrates the close relationship between PTs $\mathcal{T}$ and witnesses $\mathcal{W}$ of the ProofTree algorithm. However, our goal is a one-to-one correspondence, which requires further measures: for witness trees, we thus assume from now on (as in Example 20) that each existential node is merged with its unique universal child node and that nulls are renamed apart: that is, whenever nulls are introduced by a resolution step of ProofTree, then all nulls in $\mathit{nulls}(\gamma(\mathit{body}(\rho))) \setminus \mathit{nulls}(\gamma(\mathit{head}(\rho)))$ must be fresh (that is, they must not occur elsewhere in $\mathcal{W}$). Moreover, we eliminate redundant information from PTs and witnesses by pruning repeated subtrees: let N be a node in a PT $\mathcal{T}$ such that some null is introduced via atom A in $\lambda(N)$, and let N be closest to the root with this property, then prune all other subtrees below all nodes $M \neq N$ with $\lambda(M) = A$. Likewise, if a node in a witness $\mathcal{W}$ contains an atom A and the pair (z, A) in $\mathcal{R}_\mathcal{S}$, then we omit the resolution step for A. We refer to the reduced PT of $\mathcal{T}$ as $\mathcal{T}^*$ and to the reduced witness of $\mathcal{W}$ as $\mathcal{W}^*$. For instance, in the PT in Figure 2, we delete the node N_{10} because the information on how to derive atom $R(a, u)$ is already contained in the subtree rooted at node N_2. Likewise, in the witness in Figure 2, we omit the resolution step for atom $R(a, u)$ in node M_6, because, in this node, we have $\mathcal{R}_\mathcal{S} = \{(u, R(a, u)\}$. Hence, it is known from some resolution step "above" that u must be introduced via this atom and its derivation is checked elsewhere. We thus delete M_{10}.

It is straightforward to construct a reduced witness $\mathcal{W}^*$ from a given reduced PT $\mathcal{T}^*$, and vice versa. We refer to these constructions as $\mathcal{T}2\mathcal{W}$ and $\mathcal{W}2\mathcal{T}$, respectively: Given $\mathcal{T}^*$, we obtain $\mathcal{W}^*$ by a top-down traversal of $\mathcal{T}^*$ and merging siblings if they share a null. Moreover, if the label at a child node in $\mathcal{T}^*$ was created by applying rule ρ with substitution γ, then we add (ρ, γ) to the node label in $\mathcal{W}^*$. Finally, if such a rule application generates a new null z, then we add $(z, \gamma(\mathit{head}(\rho))$ to $\mathcal{R}_\mathcal{S}$ of every child node which still contains null z. Conversely, we obtain $\mathcal{T}^*$ from a reduced witness $\mathcal{W}^*$ by a bottom-up traversal of $\mathcal{W}^*$, where we turn every node labelled with k atoms into k siblings, each labelled with one of these atoms. The edge labels ρ in $\mathcal{T}^*$ are obtained from the corresponding (ρ, γ) labels in the node in $\mathcal{W}^*$. In summary, we have:

▶ **Lemma 21.** *There is a one-to-one correspondence between reduced proof trees $\mathcal{T}^*$ for a ground atom $P(\bar{c})$ and reduced witnesses $\mathcal{W}^*$ for its successful* ProofTree *computations. More precisely, given $\mathcal{T}^*$, we get a reduced witness as $\mathcal{T}2\mathcal{W}(\mathcal{T}^*)$; given $\mathcal{W}^*$, we get a reduced PT as $\mathcal{W}2\mathcal{T}(\mathcal{W}^*)$ with $\mathcal{W}2\mathcal{T}(\mathcal{T}2\mathcal{W}(\mathcal{T}^*)) = \mathcal{T}^*$ and $\mathcal{T}2\mathcal{W}(\mathcal{W}2\mathcal{T}(\mathcal{W}^*)) = \mathcal{W}^*$.*

So far, we have only considered warded Datalog$^{\pm}$, without negation. However, this restriction is inessential. Indeed, by the definition of warded Datalog$^{\pm,\neg sg}$, negated atoms can never contain a null in the chase. Hence, one can easily get rid of negation (in polynomial time data complexity) by computing for one stratum after the other the answer $Q(\Pi, D)$ to query Q with $Q \equiv P(x_1, \ldots, x_n)$, i.e., all ground atoms $P(a_1, \ldots, a_n)$ with $\Pi \cup D \models P(a_1, \ldots, a_n)$. We can then replace all occurrences of $\neg P(t_1, \ldots, t_n)$ in any rule of Π by the positive atom $P'(t_1, \ldots, t_n)$ (for a new predicate symbol P') and add to the instance all ground atoms $P'(a_1, \ldots, a_n)$ with $(a_1, \ldots, a_n) \notin Q(D, \Pi)$. Hence, all our results proved in this section for warded Datalog$^{\pm}$ also hold for warded Datalog$^{\pm,\neg sg}$.

Clearly, there is a one-to-one correspondence between proof trees PT $\mathcal{T}$ and their reduced forms $\mathcal{T}^*$. Hence, together with Lemma 21, we can compute the multiplicities by computing (reduced) witness trees. This allows us to obtain the results below:

▶ **Theorem 22.** *Let Π be a warded Datalog$^{\pm,\neg sg}$ program and D a database (possibly with duplicates). Then there exists a bound K which is polynomial in D, s.t. for every ground atom A:*

1. *A has finite multiplicity if and only if all reduced witness trees of A have depth $\leq K$.*
2. *If A has infinite multiplicity, then there exists at least one reduced witness tree of A whose depth is in $[K+1, 2K]$.*

Proof Idea. Recall that the data structure propagated by the ProofTree algorithm consists of pairs $(\mathcal{S}, \mathcal{R}_\mathcal{S})$. We call pairs $(\mathcal{S}, \mathcal{R}_\mathcal{S})$ and $(\mathcal{S}', \mathcal{R}_{\mathcal{S}'})$ *equivalent* if one can be obtained from the other by renaming of nulls. The bound K corresponds to the maximum number of non-equivalent pairs $(\mathcal{S}, \mathcal{R}_\mathcal{S})$ over the given signature and domain of D. By the logspace bound on this data structure, there can only be polynomially many (w.r.t. D) such pairs. For the first claim of the theorem, suppose that a (reduced) witness tree $\mathcal{W}^*$ has depth greater than K; then there must be a branch with two nodes N and N' with equivalent pairs $(\mathcal{S}, \mathcal{R}_\mathcal{S})$ and $(\mathcal{S}', \mathcal{R}_{\mathcal{S}'})$. We get infinitely many witness trees by arbitrarily often iterating the path between N and N'. ◀

In principle, Theorem 22 suffices to prove decidability of FINITENESS and design an algorithm for the MULTIPLICITY: just chase database D with the transformed warded Datalog program Π^+ up to depth $2K$. If the desired ground atom A extended by some tid is generated at a depth greater than K, then conclude that A has infinite multiplicity. Otherwise, the multiplicity of A is equal to the number of atoms of the form $(\iota; A)$ in the chase result. However, this chase of depth $2K$ may produce an exponential number of atoms and hence take exponential time. Below we show that we can in fact do significantly better:

▶ **Theorem 23.** *For warded Datalog$^{\pm,\neg sg}$ programs, both the FINITENESS problem and the MULTIPLICITY problem can be solved in polynomial time.*

Proof Idea. A decision procedure for the FINITENESS problem can be obtained by modifying the ALOGSPACE algorithm ProofTree from [4] in such a way that we additionally "guess" a branch in the witness tree with equivalent labels. The additional information thus needed also fits into logspace.

The MULTIPLICITY problem can be solved in polynomial time by a tabling approach to the ProofTree algorithm. We thus store for each (non-equivalent) value of (S, R_S) how many (reduced) witness trees it has and propagate this information upwards for each resolution step encoded in the (reduced) witness tree. ◀

6 Multiset Relational Algebra (MRA)

Following [20, 21], we consider multisets (or bags) M and elements e (from some domain) with non-negative integer multiplicities, $mult(e, M)$ (recall from Section 2.1 that, by definition, $e \in M$ iff $mult(e, M) \geq 1$). Now consider multiset relations R, S. Unless stated otherwise, we assume that R, S contain tuples of the same arity, say n. We define the following multiset operations of MRA: the *multiset union*, $\uplus$, is defined by $R \uplus S := T$, with $mult(e, T) := mult(e, R) + mult(e, S)$. *Multiset selection*, $\sigma^m_C(R)$, with a condition C, is defined as the multiset T containing all tuples in R that satisfy C with the same multiplicities as in R. For *multiset projection* $\pi^m_{\bar{k}}(R)$, we get the multiplicities $mult(e, \pi^m_{\bar{k}}(R))$ by summing up the multiplicities of all tuples in R that, when projected to the positions $\bar{k} = \langle i_1, \ldots, i_k \rangle$, produce e. For the *multiset (natural) join* $R \bowtie^m S$, the multiplicity of each tuple t is obtained as the product of multiplicities of tuples from R and of tuples from S that join to t.

For the *multiset difference*, two definitions are conceivable: Majority-based, or "monus" difference (see e.g. [15]), given by $R \ominus S := T$, with $mult(e, T) := max\{mult(e, R) - mult(e, S), 0\}$. There is also the "all-or-nothing" difference: $R \otimes S := T$, with $mult(e, T) := mult(e, R)$ if $e \notin S$ and $mult(e, T) := 0$, otherwise. Following [22], we have only considered $\otimes$ so far (implicitly, starting with Datalog$^{\neg s}$, in Section 2.1). The *multiset intersection* $\cap_m$ is not treated or used in any of [2, 20, 21, 22]. Extending the DTB semantics from [21] to $\cap_m$ would treat it as a special case of the join, which may be counter-intuitive, e.g., $\{a, a, b\} \cap_m \{a, a, a, c\} := \{a, a, a, a, a, a\}$. Alternatively, we could define "minority-based" intersection $R \cap_{mb} S$, which returns each element with its minimum multiplicity, e.g., $\{a, a, b\} \cap_{mb} \{a, a, a, c\} := \{a, a\}$.

We consider MRA with the following basic multiset operations: multiset union $\uplus$, multiset projection $\pi^m_{\bar{k}}$, multiset selection σ^m_C, with C a condition, multiset (natural) join $\bowtie^m$, and (all-or-nothing) multiset difference $\otimes$. We can also include *duplicate elimination* in MRA, which becomes operation $\mathcal{DI}$ of Definition 9 when using tids. Being just a projection in the latter case, it can be represented in Datalog. For the moment, we consider multiset-intersection as a special case of multiset (natural) join $\bowtie^m$. It is well known that the basic, set-oriented relational algebra operations can all be captured by means of (non-recursive) Datalog$^{\neg s}$ programs (cf. [1]). Likewise, one can capture MRA by means of (non-recursive) Datalog$^{\neg s}$ programs with multiset semantics (see e.g. [2]). Together with our transformation into set semantics of Datalog$^{\pm,\neg sg}$, we thus obtain:

▶ **Theorem 24.** *The Multiset Relational Algebra (MRA) can be represented by warded Datalog$^{\pm,\neg sg}$ with set semantics.*

As a consequence, the MRA operations can still be performed in polynomial-time (data complexity) via Datalog$^{\pm,\neg sg}$. This result tells us that MRA – applied at the level of an EDB with duplicates – can be represented in warded Datalog, and uniformly integrated under the same logical semantics with an ontology represented in warded Datalog.

We now retake multiset-intersection (and later also multiset-difference), which appears as $\cap_m$ and $\cap_{mb}$. The former does not offer any problem for our representation in Datalog$^{\pm}$ as above, because it is a special case of multiset join. In contrast, the *minority-based* intersection,

$\cap_{mb}$, is more problematic.[2] First, the DTB semantics does not give an account of it in terms of Datalog that we can use to build upon. Secondly, our Datalog$^{\pm}$-based formulation of duplicate management with MRA operations is set-theoretic. Accordingly, to investigate the representation of the bag-based operation $\cap_{mb}$ by means of the latter, we have to agree on a set-based reformulation $\cap_{mb}$. We propose for it a tid-based (set) representation, because tid creation becomes crucial to make it a deterministic operation. Accordingly, for multi-relations P and R with the same arity n (plus 1 for tids), we define:

$$P \cap_{mb} Q := \{(i; \bar{a}) \mid (i; \bar{a}) \in \begin{cases} P & \text{if } |\pi_0(\sigma_{1..n=\bar{a}}(P))| \leq |\pi_0(\sigma_{1..n=\bar{a}}(Q))| \\ Q & \text{otherwise} \end{cases} \}. \tag{7}$$

Here, we only assume that tids are *local* to a predicate, i.e. they act as values for a surrogate key. Intuitively, we keep for each tuple in the result the duplicates that appear in the relation that contains the minimum number of them. Here, π_0 denotes the projection on the 0-th attribute (for tids), and $\sigma_{1..n=\bar{a}}$ is the selection of those tuples which coincide with $\bar{a}$ on the next n attributes. This operation may be non-commutative when equality holds in the first case of (7) (e.g. $\{\langle 1; a\rangle\} \cap_{mb} \{\langle 2; a\rangle\} = \{\langle 1; a\rangle\} \neq \{\langle 2; a\rangle\} = \{\langle 2; a\rangle\} \cap_{mb} \{\langle 1; a\rangle\}$). Most importantly, it is non-monotonic: if any of the extensions of P or Q grows, the result may not contain the previous result,[3] e.g. $\{\langle 1; a\rangle\} \cap_{mb} \{\langle 2; a\rangle\} = \{\langle 1; a\rangle\} \not\supseteq \{\langle 2; a\rangle\} = (\{\langle 1; a\rangle\} \cup \{\langle 3; a\rangle\}) \cap_{mb} \{\langle 2; a\rangle\}$. (It is still non-monotonic under the DTB semantics.) We get the following inexpressibility results:

▶ **Proposition 25.** *The minority-based intersection with duplicates $\cap_{mb}$ as in (7) cannot be represented in Datalog, (positive) Datalog$^{\pm}$, or FO predicate logic (FOL). The same applies to the majority-based (monus) difference, $\ominus$.*

Proof Idea. By the non-monotonicity of $\cap_{mb}$, it is clear for Datalog and (positive) Datalog$^{\pm}$. For the inexpressibility in FOL, the key idea is that with the help of $\cap_{mb}$ or $\ominus$ we could express the *majority quantifier*, which is known to be undefinable in FOL [24, 25]. ◀

▶ **Proposition 26.** *The minority-based intersection with duplicates $\cap_{mb}$ as in (7) cannot be represented in Datalog$^{\neg s}$. The same applies to the majority-based difference, $\ominus$.*

Proof Idea. It can be shown that if a logic is powerful enough to express any of $\cap_{mb}$ or $\ominus$, then we could express in this logic – for sets A and B – that $|A| = |B \setminus A|$ holds. However, the latter property cannot even be expressed in the logic $L^{\omega}_{\infty\omega}$ under finite structures [13, sec. 8.4.2] and this logic extends Datalog$^{\neg s}$. ◀

7 Conclusions and Future Work

We have proposed the specification of the bag semantics of Datalog in terms of warded Datalog$^{\pm}$ with set semantics and we have also extended this specification to the bag semantics of warded Datalog$^{\pm,\neg sg}$ itself. Our work underlines that warded Datalog$^{\pm}$ is indeed a well-chosen fragment of Datalog$^{\pm}$: it provides a mild extension of Datalog by the restricted use of existentially quantified variables in the rule heads, which suffices to capture certain forms of ontological reasoning [3, 17] and, as we have seen here, the bag semantics of Datalog.

[2] The *majority-based union* operation on bags, that returns, e.g. $\{\{a, a, b, c\}\} \cup_{mab} \{\{a, b, b\}\} := \{\{a, a, b, b, c\}\}$ should be equally problematic.

[3] We could redefine (7) by introducing new tids, i.e. tuples $(f(i), \bar{a})$, for each tuple $(i, \bar{a})$ in the condition in (7), with some function f of tids. The $f(i)$ could be the next tid values after the last one used so far in a list of them. The operation defined in this would still be non-monotonic.

Further extensions of this system, above all to SPARQL with bag semantics based on previous Datalog rewritings [2, 23] are under way. Another interesting direction for future work is to investigate further inexpressibility issues such as, e.g., whether warded Datalog$^{\pm,\neg sg}$ is expressive enough to capture $\cap_{mb}$ and $\ominus$. This work will also include the development of tools to address (in)expressibility results in Datalog$^{\pm}$, with or without negation.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison Wesley, 1994.

2 Renzo Angles and Claudio Gutiérrez. The Multiset Semantics of SPARQL Patterns. In *Proc. ISWC 2016*, volume 9981 of *LNCS*, pages 20–36, 2016. doi:10.1007/978-3-319-46523-4_2.

3 Marcelo Arenas, Georg Gottlob, and Andreas Pieris. Expressive languages for querying the semantic web. In *Proc. PODS'14*, pages 14–26. ACM, 2014. doi:10.1145/2594538.2594555.

4 Marcelo Arenas, Georg Gottlob, and Andreas Pieris. Expressive languages for querying the semantic web. *ACM Trans. Database Syst. (to appear)*, 2018.

5 Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. On rules with existential variables: Walking the decidability line. *Artif. Intell.*, 175(9-10):1620–1654, 2011. doi:10.1016/j.artint.2011.03.002.

6 Luigi Bellomarini, Georg Gottlob, Andreas Pieris, and Emanuel Sallinger. Swift Logic for Big Data and Knowledge Graphs. In *Proc. IJCAI 2017*, pages 2–10. ijcai.org, 2017. doi:10.24963/ijcai.2017/1.

7 Luigi Bellomarini, Emanuel Sallinger, and Georg Gottlob. The Vadalog System: Datalog-based Reasoning for Knowledge Graphs. *PVLDB*, 11(9):975–987, 2018.

8 Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. *J. Artif. Intell. Res.*, 48:115–174, 2013.

9 Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. Datalog$^{\pm}$: a unified approach to ontologies and integrity constraints. In *Proc. ICDT 2009*, volume 361 of *ACM International Conference Proceeding Series*, pages 14–30. ACM, 2009.

10 Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general Datalog-based framework for tractable query answering over ontologies. *J. Web Sem.*, 14:57–83, 2012. doi:10.1016/j.websem.2012.03.001.

11 Andrea Calì, Georg Gottlob, and Andreas Pieris. Towards more expressive ontology languages: The query answering problem. *Artif. Intell.*, 193:87–128, 2012. doi:10.1016/j.artint.2012.08.002.

12 Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981. doi:10.1145/322234.322243.

13 Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory*. Springer, 2nd edition, 1999.

14 Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005. doi:10.1016/j.tcs.2004.10.033.

15 Floris Geerts and Antonella Poggi. On database query languages for K-relations. *J. Applied Logic*, 8(2):173–185, 2010. doi:10.1016/j.jal.2009.09.001.

16 Birte Glimm, Chimezie Ogbuji, Sandro Hawke, Ivan Herman, Bijan Parsia, Axel Polleres, and Andy Seaborne. SPARQL 1.1 Entailment Regimes. W3C Recommendation 21 march 2013, W3C, 2013. URL: https://www.w3.org/TR/sparql11-entailment/.

17 Georg Gottlob and Andreas Pieris. Beyond SPARQL under OWL 2 QL Entailment Regime: Rules to the Rescue. In *Proc. IJCAI 2015*, pages 2999–3007. AAAI Press, 2015. URL: http://ijcai.org/Abstract/15/424.

18 Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In *Proc. PODS'07*, pages 31–40. ACM, 2007. doi:10.1145/1265530.1265535.

19 David S. Johnson and Anthony C. Klug. Testing Containment of Conjunctive Queries under Functional and Inclusion Dependencies. *J. Comput. Syst. Sci.*, 28(1):167–189, 1984. doi:10.1016/0022-0000(84)90081-3.

20 Michael J. Maher and Raghu Ramakrishnan. Déjà Vu in Fixpoints of Logic Programs. In *Proc. NACLP 1989*, pages 963–980. MIT Press, 1989.

21 Inderpal Singh Mumick, Hamid Pirahesh, and Raghu Ramakrishnan. The Magic of Duplicates and Aggregates. In *Proc. VLDB 1990*, pages 264–277. Morgan Kaufmann, 1990. URL: http://www.vldb.org/conf/1990/P264.PDF.

22 Inderpal Singh Mumick and Oded Shmueli. Finiteness Properties of Database Queries. In *Proc. ADC '93*, pages 274–288. World Scientific, 1993.

23 Axel Polleres and Johannes Peter Wallner. On the relation between SPARQL1.1 and Answer Set Programming. *Journal of Applied Non-Classical Logics*, 23(1-2):159–212, 2013. doi:10.1080/11663081.2013.798992.

24 Johan van Benthem and Kees Doets. Higher-Order Logic. In Dov M. Gabbay and Franz Guenthner, editors, *Handbook of Philosophical Logic, Vol. I*, Synthese Library, Vol. 164, pages 275–329. D. Reidel Publishing Company, 1983.

25 Dag Westerståhl. Quantifiers in Formal and Natural. In Dov M. Gabbay and Franz Guenthner, editors, *Handbook of Philosophical Logic, Vol. 14*, pages 223–338. Springer, 2007.

Oblivious Chase Termination: The Sticky Case

Marco Calautti
School of Informatics, University of Edinburgh, UK
mcalautt@inf.ed.ac.uk

Andreas Pieris
School of Informatics, University of Edinburgh, UK
apieris@inf.ed.ac.uk

Abstract

The chase procedure is one of the most fundamental algorithmic tools in database theory. A key algorithmic task is uniform chase termination, i.e., given a set of tuple-generating dependencies (tgds), is it the case that the chase under this set of tgds terminates, for every input database? In view of the fact that this problem is undecidable, no matter which version of the chase we consider, it is natural to ask whether well-behaved classes of tgds, introduced in different contexts such as ontological reasoning, make our problem decidable. In this work, we consider a prominent decidability paradigm for tgds, called stickiness. We show that for sticky sets of tgds, uniform chase termination is decidable if we focus on the (semi-)oblivious chase, and we pinpoint its exact complexity: PSpace-complete in general, and NLogSpace-complete for predicates of bounded arity. These complexity results are obtained via graph-based syntactic characterizations of chase termination that are of independent interest.

2012 ACM Subject Classification Theory of computation → Database constraints theory; Theory of computation → Logic and databases

Keywords and phrases Chase procedure, tuple-generating dependencies, stickiness, termination, computational complexity

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.17

Funding Calautti and Pieris are funded by the EPSRC grant EP/S003800/1 "EQUID", and the EPSRC programme grant EP/M025268/ "VADA".

1 Introduction

The *chase procedure* (or simply chase) is a fundamental algorithmic tool that has been successfully applied to several database problems such as containment of queries under constraints [1], checking logical implication of constraints [3, 17], computing data exchange solutions [10], and query answering under constraints [5], to name a few. The chase procedure accepts as an input a database D and a set Σ of constraints and, if it terminates, its result is a finite instance D_Σ that is a *universal* model of D and Σ, i.e., is a model that can be homomorphically embedded into every other model of D and Σ. In other words, D_Σ acts as a representative of all the other models of D and Σ. This is the reason for the ubiquity of the chase in database theory, as discussed in [8]. Indeed, many key database problems can be solved by simply exhibiting a universal model.

A prominent class of constraints that can be naturally treated by the chase procedure is the class of *tuple-generating dependencies (tgds)*, i.e., sentences of the form

$$\forall \bar{x} \forall \bar{y} \, (\phi(\bar{x}, \bar{y}) \ \rightarrow \ \exists \bar{z} \, \psi(\bar{x}, \bar{z})) \, ,$$

where ϕ and ψ are conjunctions of atoms. Given a database D and a set Σ of tgds, the chase adds new atoms to D (possibly involving null values that act as witnesses for the existentially quantified variables) until the final result satisfies Σ. For example, given the database $D = \{R(c)\}$, and the tgd $\forall x(R(x) \rightarrow \exists y \, P(x, y) \wedge R(y))$, the database atom *triggers*

the tgd, and the chase will add in D the atoms $P(c, \perp_1)$ and $R(\perp_1)$ in order to satisfy it, where $\perp_1$ is a (labeled) null representing some unknown value. However, the new atom $R(\perp_1)$ triggers again the tgd, and the chase is forced to add the atoms $P(\perp_1, \perp_2), R(\perp_2)$, where $\perp_2$ is a new null. The result of the chase is the instance

$$\{R(c), P(c, \perp_1)\} \cup \bigcup_{i>0} \{R(\perp_i), P(\perp_i, \perp_{i+1})\},$$

where $\perp_1, \perp_2, \ldots$ are nulls.

The above example shows that the chase procedure may not terminate, even for very simple databases and sets of tgds. This fact motivated a long line of research on identifying subclasses of tgds that ensure the termination of the chase procedure, no matter how the input database looks like. A prime example is the class of *weakly-acyclic* sets of tgds [10], which is the standard language for data exchange purposes, and guarantees the termination of the semi-oblivious and restricted (a.k.a. standard) chase. A similar formalism, called *constraints with stratified-witness*, has been proposed in [9]. Inspired by weak-acyclicity, the notion of *rich-acyclicity* has been proposed in [16], which guarantees the termination of the oblivious chase. Many other sufficient conditions for chase termination can be found in the literature; see, e.g., [8, 9, 13, 15, 18, 19] – this list is by no means exhaustive, and we refer the reader to [14] for a comprehensive survey.

With so much effort spent on identifying sufficient conditions for the termination of the chase procedure, the question that immediately comes up is whether a sufficient condition that is also *necessary* exists. In other words, given a set Σ of tgds, is it possible to decide whether, for every database D, the chase on D and Σ terminates? This question has been addressed in [11], and has been shown that the answer is negative, no matter which version of the chase we consider, namely the oblivious, semi-oblivious and restricted chase. The problem remains undecidable even if the database is known; this has been established in [8] for the restricted chase, and it was observed in [18] that the same proof shows undecidability also for the (semi-)oblivious chase.

The undecidability proof given in [11] constructs a sophisticated set of tgds that goes beyond existing well-behaved classes of tgds that enjoy certain syntactic properties, which in turn ensure useful model-theoretic properties. This has been already observed in [4], where it is shown that the chase termination problem is decidable if we focus on the (semi-)oblivious version of the chase, and classes of tgds based on the notion of *guardedness*. Guardedness is one of the main decidability paradigms that gives rise to robust tgd-based languages [2, 5, 6] that capture important database constraints and lightweight description logics. The key model-theoretic property of guarded-based languages, which explains their robust behaviour, is the tree-likeness of the underlying universal models [5]. On the other hand, there are interesting statements that are inherently non-tree-like, and thus not expressible via guarded-based languages. Such a statement consists of the tgds

$$\forall x \forall y (R(x, y) \rightarrow \exists z\, R(y, z) \wedge P(z)) \qquad \forall x \forall y (P(x) \wedge P(y) \rightarrow S(x, y)),$$

which compute the cartesian product of a unary relation that stores infinitely many elements.

The inability of guarded-based tgds to express non-tree-like statements like the one above, has motivated a long line of research on isolating well-behaved classes of tgds that go beyond tree-like models and guardedness. The main decidability paradigm obtained from this effort is known as *stickiness* [7]. The key idea underlying stickiness can be described as follows: variables that appear more than once in the left-hand side of a tgd, known as the body of the tgd, should be inductively propagated (or "stick") to every atom in the right-hand side of

the tgd; more details are given in Section 2. It is easy to verify that the above non-tree-like statement is trivially sticky since none of the body variables occurs more than once. The crucial question that comes up is the following: *given a sticky set Σ of tgds, is it possible to decide whether the chase terminates for every input database?*

The main goal of this work is to study the chase termination problem for sticky sets of tgds, and give a definite answer to the above fundamental question. In fact, we focus on the (semi-)oblivious versions of the chase, and we show that deciding termination for sticky sets of tgds is decidable, and provide precise complexity results: PSpace-complete in general, and NLogSpace-complete for predicates of bounded arity. Although the (semi-)oblivious versions of the chase are considered as non-standard ones, they have certain advantages that classify them as important algorithmic tools, and thus they deserve our attention. In particular, unlike the restricted chase, the application of a tgd does not require checking if the right-hand side of the tgd is already satisfied by the instance, and this guarantees technical clarity and efficiency; for a more thorough discussion on the advantages of the oblivious and semi-oblivious chase see [5, 18].

Summary of Contributions. Our results can be summarized as follows:

- In Section 4, we provide a semantic characterization of non-termination of the oblivious and semi-oblivious chase under sticky sets of tgds via the existence of path-like infinite chase derivations, which forms the basis for our decision procedure.
- By exploiting the above semantic characterization, we then provide, in Section 5, a syntactic characterization of chase termination via graph-based conditions. To this end, we extend recent syntactic characterizations from [4] of the termination of the oblivious and semi-oblivious chase under *constant-free* linear tgds (tgds with one body atom), to linear tgds with constants. The transition from constant-free tgds to tgds with constants turned out to be more challenging than expected.
- Finally, in Section 6, by exploiting the graph-based syntactic characterization from the previous section, we establish the precise complexity of our problem: PSpace-complete in general, and NLogSpace-complete for predicates of bounded arity.

2 Preliminaries

We consider the disjoint countably infinite sets $\mathbf{C}$, $\mathbf{N}$, and $\mathbf{V}$ of *constants*, *(labeled) nulls*, and (regular) *variables* (used in dependencies), respectively. A fixed lexicographic order is assumed on $(\mathbf{C} \cup \mathbf{N})$ such that every null of $\mathbf{N}$ follows all constants of $\mathbf{C}$. We refer to constants, nulls and variables as *terms*. Let $[n] = \{1, \ldots, n\}$, for any integer $n \geq 1$.

Relational Databases. A *schema* $\mathbf{S}$ is a finite set of relation symbols (or predicates) with associated arity. We write R/n to denote that R has arity $n > 0$. A position $R[i]$ in $\mathbf{S}$, where $R/n \in \mathbf{S}$ and $i \in [n]$, identifies the i-th argument of R. An *atom* over $\mathbf{S}$ is an expression of the form $R(\bar{t})$, where $R/n \in \mathbf{S}$ and $\bar{t}$ is an n-tuple of terms. We write $\mathsf{var}(\alpha)$ for the set of variables occurring in an atom α; this notation naturally extends to sets of atoms. A *fact* is an atom whose arguments consist only of constants. An *instance* over $\mathbf{S}$ is a (possibly infinite) set of atoms over $\mathbf{S}$ that contain constants and nulls, while a *database* over $\mathbf{S}$ is a finite set of facts over $\mathbf{S}$. The *active domain* of an instance I, denoted $\mathsf{dom}(I)$, is the set of all terms, i.e., constants and nulls, occurring in I.

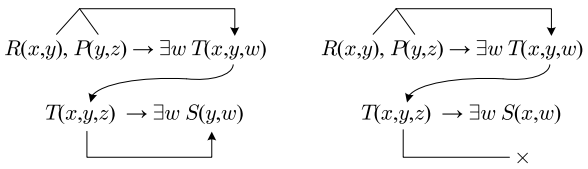
Substitutions and Homomorphisms. A *substitution* from a set T of terms to a set T' of terms is a function $h : T \to T'$ defined as follows: $\emptyset$ is a substitution (empty substitution), and if h is a substitution, then $h \cup \{t \mapsto t'\}$, where $t \in T$ and $t' \in T'$, is a substitution. The restriction of h to a subset S of T, denoted $h_{|S}$, is the substitution $\{t \mapsto h(t) \mid t \in S\}$. A *homomorphism* from a set A of atoms to a set B of atoms is a substitution h from the set of terms in A to the set of terms in B such that (i) $t \in \mathbf{C}$ implies $h(t) = t$, i.e., h is the identity on $\mathbf{C}$, and (ii) $R(t_1, \ldots, t_n) \in A$ implies $h(R(t_1, \ldots, t_n)) = R(h(t_1), \ldots, h(t_n)) \in B$.

Tuple-Generating Dependencies. A *tuple-generating dependency* σ is a sentence

$$\forall \bar{x} \forall \bar{y} \, (\phi(\bar{x}, \bar{y}) \ \rightarrow \ \exists \bar{z} \, \psi(\bar{x}, \bar{z})) \, ,$$

where $\bar{x}, \bar{y}, \bar{z}$ are tuples of variables of $\mathbf{V}$, while $\phi(\bar{x}, \bar{y})$ and $\psi(\bar{x}, \bar{z})$ are conjunctions of atoms (possibly with constants). For brevity, we write σ as $\phi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z} \, \psi(\bar{x}, \bar{z})$, and use comma instead of $\wedge$ for joining atoms. We refer to $\phi(\bar{x}, \bar{y})$ and $\psi(\bar{x}, \bar{z})$ as the *body* and *head* of σ, denoted $\mathsf{body}(\sigma)$ and $\mathsf{head}(\sigma)$, respectively. The *frontier* of the tgd σ, denoted $\mathsf{fr}(\sigma)$, is the set of variables $\bar{x}$, i.e., the variables that appear both in the body and the head of σ. The *schema* of a set Σ of tgds, denoted $\mathsf{sch}(\Sigma)$, is the set of predicates in Σ. We also write $\mathsf{const}(\Sigma)$ for the set of constants occurring in Σ. An instance I satisfies a tgd σ as the one above, written $I \models \sigma$, if the following holds: whenever there exists a homomorphism h such that $h(\phi(\bar{x}, \bar{y})) \subseteq I$, then there exists $h' \supseteq h_{|\bar{x}}$ such that $h'(\psi(\bar{x}, \bar{z})) \subseteq I$. Note that, by abuse of notation, we sometimes treat a conjunction of atoms as a set of atoms. The instance I satisfies a set Σ of tgds, written $I \models \Sigma$, if $I \models \sigma$ for each $\sigma \in \Sigma$.

One of the main syntactic paradigms for tgds is stickiness [7]. The key property underlying this condition is as follows: variables that appear more than once in the body of a tgd should be inductively propagated (or "stick") to every head atom. This is graphically illustrated as



where the first set of tgds is sticky, while the second is not. The formal definition is based on an inductive procedure that marks the variables that may violate the property described above. Roughly, during the base step of this procedure, a variable that appears in the body of a tgd but not in every head atom is marked. Then, the marking is inductively propagated from head to body. Stickiness requires every marked variable to appear only once in the body of a tgd. The formal definition follows. Let Σ be a set of tgds; w.l.o.g., we assume that the tgds in Σ do not share variables. Given an atom $R(\bar{t})$ and a variable x in $\bar{t}$, $\mathsf{pos}(R(\bar{t}), x)$ is the set of positions in $R(\bar{t})$ at which x occurs. Let $\sigma \in \Sigma$ and x a variable in the body of σ. We inductively define when x *is marked in* Σ:

- If x does not occur in every atom of $\mathsf{head}(\sigma)$, then x is marked in Σ.
- Assuming that $\mathsf{head}(\sigma)$ contains an atom of the form $R(\bar{t})$ and $x \in \bar{t}$, if there exists $\sigma' \in \Sigma$ that has in its body an atom of the form $R(\bar{t}')$, and each variable in $R(\bar{t}')$ at a position of $\mathsf{pos}(R(\bar{t}), x)$ is marked in Σ, then x is marked in Σ.

The set Σ is *sticky* if there is no tgd that contains two occurrences of a variable that is marked in Σ. We denote by $\mathbb{S}$ the class of sticky finite sets of tgds. Let us clarify that we work with finite sets of tgds only. Thus, in the rest of the paper, a set of tgds is always finite.

The Tgd Chase Procedure. The tgd chase procedure (or simply chase) takes as an input a database D and a set Σ of tgds, and constructs a (possibly infinite) instance I such that $I \supseteq D$ and $I \models \Sigma$. A crucial notion is that of trigger for a set of tgds on some instance. Consider a set Σ of tgds and an instance I. A *trigger* for Σ on I is a pair (σ, h), where $\sigma \in \Sigma$ and h is a homomorphism such that $h(\mathsf{body}(\sigma)) \subseteq I$. An application of (σ, h) to I returns the instance $J = I \cup h'(\mathsf{head}(\sigma))$, where $h' \supseteq h_{|\mathsf{fr}(\sigma)}$ is such that (i) for each existentially quantified variable z of σ, $h'(z) \in \mathbf{N}$ does not occur in I and follows lexicographically all nulls in I, and (ii) for each pair (z, w) of distinct existentially quantified variables of σ, $h'(z) \neq h'(w)$. Such a trigger application is denoted as $I\langle\sigma, h\rangle J$.

The main idea of the chase is, starting from a database D, to exhaustively apply triggers for the given set Σ of tgds on the instance constructed so far. However, the choice of the next trigger to be applied is crucial since it gives rise to different variations of the chase procedure. In this work, we focus on the *oblivious* [5] and the *semi-oblivious* [12, 18] chase.

Oblivious. A finite sequence $I_0, I_1, \ldots, I_n$ of instances, where $n \geq 0$, is said to be a *terminating oblivious chase sequence* of I_0 w.r.t. a set Σ of tgds if: (i) for each $0 \leq i < n$, there exists a trigger (σ, h) for Σ on I_i such that $I_i\langle\sigma, h\rangle I_{i+1}$; (ii) for each $0 \leq i < j < n$, assuming that $I_i\langle\sigma_i, h_i\rangle I_{i+1}$ and $I_j\langle\sigma_j, h_j\rangle I_{j+1}$, $\sigma_i = \sigma_j$ implies $h_i \neq h_j$, i.e., h_i and h_j are different homomorphisms; and (iii) there is no trigger (σ, h) for Σ on I_n such that $(\sigma, h) \notin \{(\sigma_i, h_i)\}_{0 \leq i < n}$. In this case, the result of the chase is the (finite) instance I_n. An infinite sequence $I_0, I_1, \ldots$ of instances is said to be a *non-terminating oblivious chase sequence* of I_0 w.r.t. Σ if: (i) for each $i \geq 0$, there exists a trigger (σ, h) for Σ on I_i such that $I_i\langle\sigma, h\rangle I_{i+1}$; (ii) for each $i, j > 0$ such that $i \neq j$, assuming that $I_i\langle\sigma_i, h_i\rangle I_{i+1}$ and $I_j\langle\sigma_j, h_j\rangle I_{j+1}$, $\sigma_i = \sigma_j$ implies $h_i \neq h_j$; and (iii) for each $i \geq 0$, and for every trigger (σ, h) for Σ on I_i, there exists $j \geq i$ such that $I_j\langle\sigma, h\rangle I_{j+1}$; this is the *fairness condition*, and guarantees that all the triggers eventually will be applied. The result of the chase is $\bigcup_{i \geq 0} I_i$.

Semi-oblivious. This is a refined version of the oblivious chase, which avoids the application of some superfluous triggers. Roughly, given a tgd σ, for the semi-oblivious chase, two homomorphisms h and g that agree on the frontier of σ, i.e., $h_{|\mathsf{fr}(\sigma)} = g_{|\mathsf{fr}(\sigma)}$, are indistinguishable. To formalize this, we first define the binary relation $\sim_\sigma$ on the set of all possible substitutions from the terms in $\mathsf{body}(\sigma)$ to $(\mathbf{C} \cup \mathbf{N})$, denoted S_σ, as follows: $h \sim_\sigma g$ iff $h_{|\mathsf{fr}(\sigma)} = g_{|\mathsf{fr}(\sigma)}$. It is easy to verify that $\sim_\sigma$ is an equivalence relation on S_σ. A (terminating or non-terminating) oblivious chase sequence $I_0, I_1, \ldots$ is called *semi-oblivious* if the following holds: for every $i, j \geq 0$ such that $i \neq j$, assuming that $I_i\langle\sigma_i, h_i\rangle I_{i+1}$ and $I_j\langle\sigma_j, h_j\rangle I_{j+1}$, $\sigma_i = \sigma_j = \sigma$ implies $h_i \not\sim_\sigma h_j$, i.e., h_i and h_j belong to different equivalence classes.

Henceforth, we write o-chase and so-chase for oblivious and semi-oblivious chase, respectively. A useful notion that we are going to use in our proofs is the so-called chase relation [7], which essentially describes how the atoms generated during the chase depend on each other. Fix a non-terminating $\star$-chase sequence $s = (I_i)_{i \geq 0}$, where $\star \in \{\mathsf{o}, \mathsf{so}\}$, of a database D w.r.t. a set Σ of tgds, and assume that for each $i \geq 0$, $I_i\langle\sigma_i, h_i\rangle I_{i+1}$, i.e., I_{i+1} is obtained from I_i via the application of the trigger (σ_i, h_i) to I_i. The *chase relation* of s, denoted $\prec_s$, is a binary relation over $\bigcup_{i \geq 0} I_i$ such that $\alpha \prec_s \beta$ iff there exists $i \geq 0$ such that $\alpha \in h_i(\mathsf{body}(\sigma_i))$ and $\beta \in I_{i+1} \setminus I_i$.

3 Chase Termination Problem

It is well-known that due to the existentially quantified variables, a $\star$-chase sequence, where $\star \in \{\mathsf{o}, \mathsf{so}\}$, may be infinite. This is true even for very simple settings: it is easy to verify that the only $\star$-chase sequence of $D = \{R(a, b)\}$ w.r.t. the set Σ consisting of the single tgd $R(x, y) \rightarrow \exists z\, R(y, z)$ is non-terminating. The question that comes up is, given a set Σ of tgds, whether we can check that, for every database D, all or some (semi-)oblivious chase sequences of D w.r.t. Σ are terminating. Before formalizing the above problem, let us recall the following central classes of tgds:

$$\mathbb{CT}^{\star}_{\forall\forall} = \left\{ \Sigma \;\middle|\; \begin{array}{l} \text{for } \textit{every} \text{ database } D, \\ \textit{every } \star\text{-chase sequence of } D \text{ w.r.t. } \Sigma \text{ is terminating} \end{array} \right\}$$

$$\mathbb{CT}^{\star}_{\forall\exists} = \left\{ \Sigma \;\middle|\; \begin{array}{l} \text{for } \textit{every} \text{ database } D, \\ \text{there } \textit{exists} \text{ a terminating } \star\text{-chase sequence of } D \text{ w.r.t. } \Sigma \end{array} \right\}$$

The main problems tackled in this work are defined as follows, where $\mathbb{C}$ is a class of tgds:

PROBLEM :	$\mathsf{CT}^{\star}_{\forall\forall}(\mathbb{C})$
INPUT :	A set $\Sigma \in \mathbb{C}$ of tgds.
QUESTION :	Is $\Sigma \in \mathbb{CT}^{\star}_{\forall\forall}$?

PROBLEM :	$\mathsf{CT}^{\star}_{\forall\exists}(\mathbb{C})$
INPUT :	A set $\Sigma \in \mathbb{C}$ of tgds.
QUESTION :	Is $\Sigma \in \mathbb{CT}^{\star}_{\forall\exists}$?

It is well-known that $\mathbb{CT}^{\mathsf{o}}_{\forall\forall} = \mathbb{CT}^{\mathsf{o}}_{\forall\exists} \subset \mathbb{CT}^{\mathsf{so}}_{\forall\forall} = \mathbb{CT}^{\mathsf{so}}_{\forall\exists}$ [12]. This immediately implies that, after fixing the version of the chase in consideration, i.e., oblivious or semi-oblivious, the above decision problems are equivalent. Henceforth, for a class $\mathbb{C}$ of tgds, we simply refer to the problem $\mathsf{CT}^{\star}_{\forall}(\mathbb{C})$, and we write $\mathbb{CT}^{\star}_{\forall}$ for the classes $\mathbb{CT}^{\star}_{\forall\forall}$ and $\mathbb{CT}^{\star}_{\forall\exists}$, where $\star \in \{\mathsf{o}, \mathsf{so}\}$.

We know that our main problem is undecidable if we consider arbitrary tgds. In fact, assuming that $\mathbb{TGD}$ denotes the class of arbitrary tgds, we have that:

▶ **Theorem 1.** *For $\star \in \{\mathsf{o}, \mathsf{so}\}$, $\mathsf{CT}^{\star}_{\forall}(\mathbb{TGD})$ is undecidable.*

The above result has been shown in [11]. However, the employed set of tgds for showing this result is far from being sticky. This led us to ask whether $\mathsf{CT}^{\star}_{\forall}(\mathbb{S})$ is decidable. This is a non-trivial problem, and pinpointing its complexity is the main goal of this work.

Some Useful Results. Before proceeding with the complexity analysis, let us recall a couple of technical results that would allow us to significantly simplify our later analysis.

It would be useful to have a special database that gives rise to a non-terminating chase sequence if it exists. Interestingly, such a database exists, which is known as the critical database for a set of tgds [18]. Formally, given a set Σ of tgds, the *critical database* for Σ is

$$\mathsf{cr}(\Sigma) = \begin{cases} \{R(c, \ldots, c) \mid R \in \mathsf{sch}(\Sigma)\}, \text{where } c \in \mathbf{C} \text{ is a fixed constant} & \text{if } \mathsf{const}(\Sigma) = \emptyset, \\ \{R(c_1, \ldots, c_n) \mid R \in \mathsf{sch}(\Sigma) \text{ and } (c_1, \ldots, c_n) \in \mathsf{const}(\Sigma)^n\} & \text{if } \mathsf{const}(\Sigma) \neq \emptyset. \end{cases}$$

In other words, $\mathsf{cr}(\Sigma)$ consists of all the atoms that can be formed using the predicates and the constants in Σ; if Σ is constant-free, then we consider an arbitrary constant of $\mathbf{C}$. The following result from [18] shows that $\mathsf{cr}(\Sigma)$ is indeed the desired database:

▶ **Proposition 2.** *Consider a set Σ of tgds. For $\star \in \{\mathsf{o}, \mathsf{so}\}$, $\Sigma \notin \mathbb{CT}^{\star}_{\forall}$ iff there exists a non-terminating $\star$-chase sequence of $\mathsf{cr}(\Sigma)$ w.r.t. Σ.*

Even though we can focus on the critical database and check whether it gives rise to a non-terminating chase sequence s, the main difficulty is to ensure that s enjoys the fairness condition. Interestingly, as it has been recently shown in [4], we can neglect the fairness condition, which significantly simplifies the required analysis. To formalize this result, we need to recall the notion of the infinite chase derivation, which is basically a non-terminating chase sequence without the fairness condition. Fix $\star \in \{\mathsf{o}, \mathsf{so}\}$. We define $\diamond^\star_\sigma$ as $\neq$, if $\star = \mathsf{o}$, and $\not\sim_\sigma$, if $\star = \mathsf{so}$. An *infinite $\star$-chase derivation* of a database D w.r.t. a set Σ of tgds is an infinite sequence $(I_i)_{i\geq 0}$ of instances, where $I_0 = D$, such that: (i) for each $i \geq 0$, there exists a trigger (σ_i, h_i) for Σ in I_i with $I_i \langle \sigma_i, h_i \rangle I_{i+1}$, and (ii) for each $i \neq j$, $\sigma_i = \sigma_j = \sigma$ implies $h_i \diamond^\star_\sigma h_j$. The following holds:

▶ **Proposition 3.** *Consider a database D and a set Σ of tgds. For $\star \in \{\mathsf{o}, \mathsf{so}\}$, the following are equivalent:*

1. *There is a non-terminating $\star$-chase sequence of D w.r.t. Σ.*
2. *There is an infinite $\star$-chase derivation of D w.r.t. Σ.*

By combining Propositions 2 and 3, we immediately get the following useful result:

▶ **Corollary 4.** *Consider a set Σ of tgds. For $\star \in \{\mathsf{o}, \mathsf{so}\}$, $\Sigma \notin \mathbb{CT}^\star_\forall$ iff there exists an infinite $\star$-chase derivation of $\mathsf{cr}(\Sigma)$ w.r.t Σ.*

4 Semantic Characterization of Chase Non-Termination

We proceed to characterize the non-termination of the (semi-)oblivious chase under sticky sets of tgds. In particular, we show that if a sticky set Σ of tgds does not belong to $\mathbb{CT}^\star_\forall$, for $\star \in \{\mathsf{o}, \mathsf{so}\}$, then we can always isolate a *linear* infinite $\star$-chase derivation δ_ℓ of $\mathsf{cr}(\Sigma)$ w.r.t. Σ. Roughly, linearity means that there is an infinite simple path $\alpha_0, \alpha_1, \alpha_2 \ldots$ in the chase relation of δ_ℓ such that $\alpha_0 \in \mathsf{cr}(\Sigma)$ and α_i is constructed during the i-th trigger application, while all the atoms that are needed to construct this path, and are not already on the path, are atoms of $\mathsf{cr}(\Sigma)$. Notice that the chase relation of a $\star$-chase derivation is defined in the same way as the chase relation of a $\star$-chase sequence.

▶ **Definition 5.** *Consider a set Σ of tgds. For $\star \in \{\mathsf{o}, \mathsf{so}\}$, an infinite $\star$-chase derivation $\delta = (I_i)_{i\geq 0}$ of $\mathsf{cr}(\Sigma)$ w.r.t. Σ, where $I_i \langle \sigma_i, h_i \rangle I_{i+1}$ for $i \geq 0$, is called* linear *if there exists an infinite sequence of distinct atoms $(\alpha_i)_{i\geq 0}$ such that the following hold:*

- $\alpha_0 \in \mathsf{cr}(\Sigma)$.
- *For each $i \geq 0$, $\alpha_{i+1} \in I_{i+1} \setminus I_i$, and there exists $\beta \in \mathsf{body}(\sigma_i)$ such that $h_i(\beta) = \alpha_i$ and $h_i(\mathsf{body}(\sigma_i) \setminus \{\beta\}) \subseteq \mathsf{cr}(\Sigma)$.*

A simple example that illustrates the notion of linear infinite o-chase derivation follows:

▶ **Example 6.** Let Σ be the sticky set consisting of the tgd

$$\sigma = P(x, y, z), R(y, w) \rightarrow \exists v\, P(z, y, v), R(y, v).$$

Consider the infinite o-chase derivation $\delta = (I_i)_{i\geq 0}$ of $\mathsf{cr}(\Sigma)$ w.r.t. Σ, where

$$\begin{aligned}
I_0 &= \{P(c,c,c), R(c,c)\} && \langle \sigma, h_0 = \{x \mapsto c, y \mapsto c, z \mapsto c, w \mapsto c\}\rangle \\
I_1 = I_0 \cup \{P(c,c,\bot_1), R(c,\bot_1)\} && \langle \sigma, h_1 = \{x \mapsto c, y \mapsto c, z \mapsto \bot_1, w \mapsto c\}\rangle \\
I_2 = I_1 \cup \{P(\bot_1,c,\bot_2), R(c,\bot_2)\} && \langle \sigma, h_2 = \{x \mapsto \bot_1, y \mapsto c, z \mapsto \bot_2, w \mapsto c\}\rangle \\
&\vdots \\
I_{i+1} = I_i \cup \{P(\bot_i,c,\bot_{i+1}), R(c,\bot_{i+1})\} && \langle \sigma, h_{i+1} = \{x \mapsto \bot_i, y \mapsto c, z \mapsto \bot_{i+1}, w \mapsto c\}\rangle \\
&\vdots
\end{aligned}$$

Let $\alpha_0 = P(c,c,c)$, $\alpha_1 = P(c,c,\bot_1)$, and $\alpha_i = P(\bot_{i-1},c,\bot_i)$ for $i > 1$. It is easy to verify that δ is linear due to $(\alpha_i)_{i\geq 0}$. Indeed, $\alpha_0 \in \mathsf{cr}(\Sigma)$, and for every $i \geq 0$, α_i belongs to $I_{i+1} \setminus I_i$, while $h_i(P(x,y,z)) = \alpha_i$ and $h_i(R(y,w)) = R(c,c) \in \mathsf{cr}(\Sigma)$.

We are now ready to present the main characterization of non-termination of the oblivious and semi-oblivious chase under sticky sets of tgds via linear infinite $\star$-chase derivations.

▶ **Theorem 7.** *Consider a set $\Sigma \in \mathbb{S}$ of tgds. For $\star \in \{\mathsf{o}, \mathsf{so}\}$, $\Sigma \notin \mathbb{CT}^{\star}_{\forall}$ iff there exists a linear infinite $\star$-chase derivation of $\mathsf{cr}(\Sigma)$ w.r.t. Σ.*

By Corollary 4, it suffices to show the following: the existence of an infinite $\star$-chase derivation of $\mathsf{cr}(\Sigma)$ w.r.t. Σ implies the existence of a linear infinite $\star$-chase derivation of $\mathsf{cr}(\Sigma)$ w.r.t. Σ. This is a rather involved result, which is established in two main steps:

1. We show that the existence of an infinite $\star$-chase derivation of $\mathsf{cr}(\Sigma)$ w.r.t. Σ implies the existence of an infinite $\star$-chase derivation δ of $\mathsf{cr}(\Sigma)$ w.r.t. Σ such that the chase relation of δ contains a special path rooted at an atom of $\mathsf{cr}(\Sigma)$, called *continuous*. Intuitively, continuity ensures the continuous propagation of a new null on the path in question.
2. By exploiting the existence of a continuous path, we construct a linear infinite $\star$-chase derivation of $\mathsf{cr}(\Sigma)$ w.r.t. Σ. In fact, due to stickiness, we can convert an infinite suffix P of the continuous path in $\prec_\delta$, together with all the atoms that are needed to generate the atoms on P via a single trigger application, into a linear infinite $\star$-chase derivation δ_ℓ of $\mathsf{cr}(\Sigma)$ w.r.t. Σ. As we shall see, stickiness helps us to ensure that δ_ℓ is linear, while continuity allows us to show that δ_ℓ is infinite.

We proceed to give some more details for the above two steps. Although we keep the following discussion informal, we give enough evidence for the validity of Theorem 7.

4.1 Existence of a Continuous Path

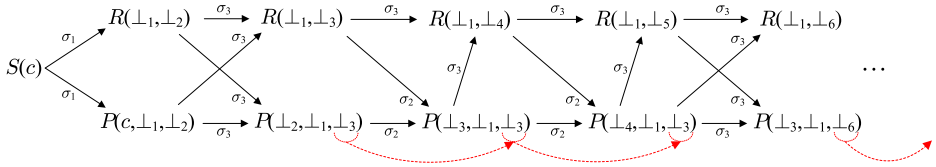
Let us first make the notion of the path in the chase relation of a derivation more precise. Given an infinite $\star$-chase derivation $\delta = (I_i)_{i\geq 0}$ of $\mathsf{cr}(\Sigma)$ w.r.t. Σ, a *finite δ-path* is a finite sequence of atoms $(\alpha_i)_{0\leq i\leq n}$ such that $\alpha_0 \in I_0$ and $\alpha_i \prec_\delta \alpha_{i+1}$. Analogously, we can define *infinite δ-paths*, which are infinite sequences of atoms rooted at an atom of I_0.

The intention underlying continuity is to ensure the continuous propagation of a new null on a path. Roughly, a δ-path $(\alpha_i)_{0\leq i\leq n}$ is continuous via a sequence of indices $(\ell_i)_{0\leq i\leq m}$, with $\ell_0 < \cdots < \ell_m$, if $\ell_0 = 1$, $\ell_m = n$, and, for each $i \in \{0,\ldots,m\}$, a new null is invented in α_{ℓ_i} that is *necessarily propagated* up to the atom $\alpha_{\ell_{i+1}}$ in case $\star = \mathsf{so}$ (resp., the atom before $\alpha_{\ell_{i+1}}$ in case $\star = \mathsf{o}$). An infinite δ-path $(\alpha_i)_{i\geq 0}$ is continuous if there exists an infinite sequence of indices $(\ell_i)_{i\geq 0}$, with $\ell_0 < \ell_1 < \cdots$, such that every finite δ-path $(\alpha_i)_{0\leq i\leq \ell_j}$, for $j \geq 0$, is continuous via $(\ell_i)_{0\leq i\leq \ell_j}$. Here is a simple example that illustrates this notion.

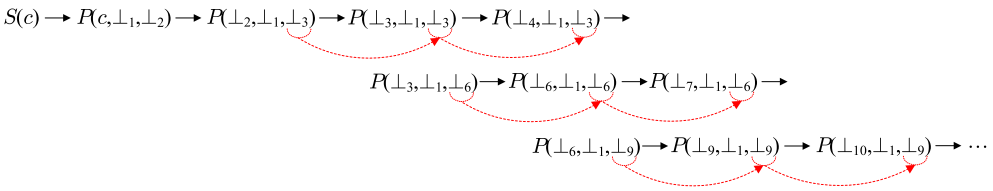
▶ **Example 8.** Consider the sticky set $\Sigma = \{\sigma_1, \sigma_2, \sigma_3\}$, where

$$\begin{aligned}
\sigma_1 &= S(x) \rightarrow \exists y \exists z\, P(x,y,z), R(y,z) \\
\sigma_2 &= P(x,y,z), R(y,w) \rightarrow P(w,y,z) \\
\sigma_3 &= P(x,y,z), R(y,w) \rightarrow \exists v\, P(z,y,v), R(y,v).
\end{aligned}$$

Notice that σ_3 is the tgd used in Example 6. It is easy to verify that there exists an infinite o-chase derivation δ of $\mathsf{cr}(\Sigma)$ w.r.t. Σ such that the following is part of $\prec_\delta$; a black edge from α to β labeled by σ means that (α, β) belongs to $\prec_\delta$ due to a trigger that involves the tgd σ:



It can be verified that the path with P-atoms in the figure is a continuous infinite δ-path. Let us explain the reason. The first atom in which a null is invented is $P(c, \bot_1, \bot_2)$, with $\bot_1, \bot_2$ being the new nulls, and continuity is satisfied since the next atom invents a null, that is, $\bot_3$. Now, since the null $\bot_3$ is propagated (this is indicated via the red dashed arrows) up to the atom before the next null generator $P(\bot_3, \bot_1, \bot_6)$, continuity is satisfied. In the rest of the path the same pattern is repeated, and thus continuity is globally satisfied. In fact, the pattern that we can extract is the following



where the continuous propagation of a new null (red arrows) can be easily observed.

We can show, via a graph-theoretic argument, that the existence of an infinite $\star$-chase derivation of $\mathsf{cr}(\Sigma)$ w.r.t. Σ implies the existence of an infinite $\star$-chase derivation of $\mathsf{cr}(\Sigma)$ w.r.t. Σ that admits a continuous infinite path. Let us briefly explain the key idea underlying this result. If we know that an infinite $\star$-chase derivation δ of $\mathsf{cr}(\Sigma)$ w.r.t. Σ exists, then we can construct an infinite $\star$-chase derivation $\delta' = (I_i)_{i \geq 0}$ of $\mathsf{cr}(\Sigma)$ w.r.t. Σ (by essentially rearranging the triggers of δ in order to obtain a derivation of a convenient form) such that the following statement holds:

there exists an infinite directed acyclic rooted graph $G = (N \cup \{\bullet\}, E, \lambda)$ of finite degree, where $\bullet$ is the root, $N \subseteq \bigcup_{i \geq 0} I_i$, every node of N is reachable from $\bullet$, and λ labels the edges of E with finite sequences of atoms from $\bigcup_{i \geq 0} I_i$, such that, for every finite path $\bullet, v_1, \ldots, v_n$, for $n \geq 1$, with $\lambda(\bullet, v_1) = (\alpha^i_j)_{0 \leq j \leq m_1 - 1}$, and $\lambda(v_{i-1}, v_i) = (\alpha^i_j)_{0 \leq j \leq m_i - 1}$,

$$\left((\alpha^i_j)_{0 \leq j \leq m_i - 1} \right)_{1 \leq i \leq n} = (\alpha_i)_{0 \leq i \leq (m_1 + \cdots + m_n) - 1}$$

is a continuous δ'-path via $(\ell_i)_{0 \leq i \leq n-1}$, where $\ell_0 = 1$ and $\ell_i = \ell_{i-1} + m_{i+1}$, for $i \in [n-1]$.

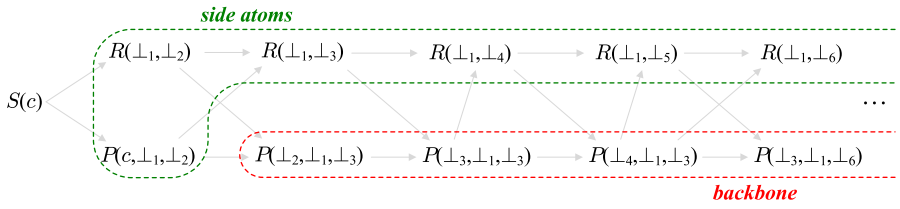
By applying König's lemma[1] on G, we get that G contains an infinite simple path $P = \bullet, v_1, v_2, \ldots$. We claim that $P' = \lambda(\bullet, v_1), \lambda(v_1, v_2), \ldots$ is a continuous δ'-path, which establishes the claim. By contradiction, assume that P' is not a continuous δ'-path. This implies that there exists a finite prefix $\bullet, v_1, \ldots, v_{n'}$ of P such that $\lambda(\bullet, v_1), \lambda(v_1, v_2), \ldots, \lambda(v_{n'-1}, v_{n'})$ is not a continuous δ'-path via $(\ell_i)_{0 \leq i \leq n'-1}$ – this is the sequence of indices used in the above statement – which contradicts the fact that the label of every finite path $\bullet, v_1, \ldots, v_n$, for $n \geq 1$, is a continuous finite δ'-path via $(\ell_i)_{0 \leq i \leq n-1}$.

4.2 From Continuous Paths to Linear Infinite Derivations

We now discuss that the existence of an infinite $\star$-chase derivation δ of $\mathsf{cr}(\Sigma)$ w.r.t. Σ such that a continuous infinite δ-path exists implies the existence of a linear infinite $\star$-chase derivation δ_ℓ of $\mathsf{cr}(\Sigma)$ w.r.t. Σ. Starting from δ, we are going to construct a sequence of instances that leads to the desired derivation δ_ℓ. The construction proceeds in three steps:

Useful part of δ. We first isolate a useful part of the $\star$-chase derivation $\delta = (I_i)_{i \geq 0}$. Recall that there exists a continuous infinite δ-path $P = (\alpha_i)_{i \geq 0}$. By stickiness, there exists $j \geq 0$ such that α_j is the last atom on P in which a term t becomes *sticky*. The latter means that the first time t participates in a join is during the trigger application that generates α_j, and thus t occurs in (or sticks to) every atom of $\{\alpha_i\}_{i \geq j}$. Let $k \geq j$ be the integer such that α_k is the first atom on P after α_j in which a new null is invented. The useful part of δ that we are going to focus on is the infinite sequence of atoms $(\alpha_i)_{i \geq k}$, which we call the *backbone*, and the atoms of $\bigcup_{i \geq 0} I_i$, which we call *side atoms*, that are needed to generate the atoms on the backbone via a single trigger application. In other words, for a backbone atom α, if α is obtained via the trigger (σ, h) for Σ on instance I_i, for some $i \geq 0$, then the atoms $h(\mathsf{body}(\sigma))$, excluding the backbone atoms, are side atoms.

▶ **Example 9.** Consider again the set $\Sigma \in \mathbb{S}$ from Example 8. As discussed above, there exists an infinite o-chase derivation δ of $\mathsf{cr}(\Sigma)$ w.r.t. Σ such that a continuous infinite δ-path exists (see the figures above). The useful part of δ is as shown below

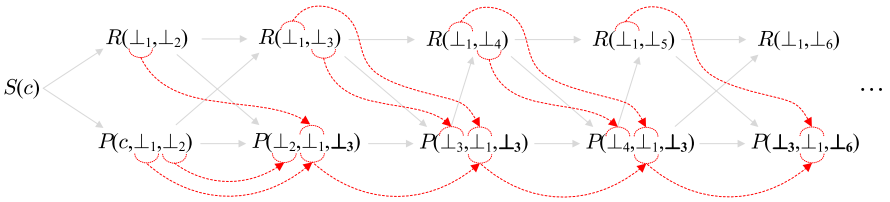


Observe that the last atom on the continuous path in which a term becomes sticky is $P(\bot_2, \bot_1, \bot_3)$; in fact, the sticky term is $\bot_1$, which is the only sticky term on the continuous path. It happened that $P(\bot_2, \bot_1, \bot_3)$ invents also a new null, that is, $\bot_3$, and therefore the suffix of the continuous path that starts at $P(\bot_2, \bot_1, \bot_3)$ is the backbone. It is now easy to verify that all the other atoms, apart from $S(c)$, indeed contribute in the generation of a backbone atom via a single trigger application.

[1] König's lemma is a well-known result from graph theory that states the following: for an infinite directed rooted graph, if every node is reachable from the root, and every node has finite out-degree, then there exists an infinite directed simple path from the root.

Renaming step. We proceed to rename some of the nulls that occur in backbone atoms or side atoms. In particular, for every null $\bot$ occurring in a side atom α, we apply the following renaming steps; fix a constant $c \in \mathsf{dom}(\mathsf{cr}(\Sigma))$: (i) every occurrence of $\bot$ in α is replaced by c, and (ii) every occurrence of $\bot$ in a backbone atom β that is propagated from α to β is replaced by c. For a backbone or side atom α, let $\rho(\alpha)$ be the atom obtained from α after globally applying the above renaming steps. We now define the sequence of instances $\delta' = (J_i)_{i \geq 0}$ as follows: $J_0 = \{\rho(\alpha) \mid \alpha \text{ is a side atom } \} \subseteq \mathsf{cr}(\Sigma)$ and $J_i = J_{i-1} \cup \{\rho(\alpha_{k+i-1})\} \cup H$, where H is the set of atoms that are generated together with α_{k+i-1} (since we can have a conjunction of atoms in the head of a tgd) after renaming the nulls that do not occur in $\rho(\alpha_{k+i-1})$ to c. Notice that H is empty in case of single-head tgds. It is crucial to observe that a new null generated in a backbone atom never participates in a join. This is because the first backbone atom α_k comes after the atom α_j, which is the last atom on P in which a term becomes sticky. This fact allows us to modify triggers from δ in order to construct, for every $i \geq 0$, a trigger (σ_i, h_i) such that $J_i \langle \sigma_i, h_i \rangle J_{i+1}$.

▶ **Example 10.** We consider again our running example. Before renaming the nulls that appear in side atoms, we first need to understand how nulls are propagated from side atoms to backbone atoms during the chase. This is depicted in the following figure



Notice that the boldfaced occurrences of the nulls $\bot_3, \bot_6, \ldots$ are not propagated from side atoms, but generated on the backbone, and thus will not be renamed. Let us recall that the existence of such nulls is guaranteed by continuity. By applying the renaming step, i.e., by replacing every null in a side atom with the constant c, and then propagating it to the backbone as indicated above, we get the sequence of instances $J_0 = \{R(c,c), P(c,c,c)\} \subseteq \mathsf{cr}(\Sigma)$, $J_1 = J_0 \cup \{P(c,c,\bot_3), R(c,\bot_3)\}$, $J_2 = J_1 \cup \{P(c,c,\bot_3)\}$, $J_3 = J_2 \cup \{P(c,c,\bot_3)\}$, $J_4 = J_3 \cup \{P(\bot_3,c,\bot_6), R(c,\bot_6)\}, \ldots$. Observe that, due to stickiness, none of the nulls $\bot_3, \bot_6, \ldots$ generated on the backbone participates in a join. This means that the renaming step preserves all the joins, and thus, by adapting triggers from δ, we can devise a valid trigger for each pair (J_i, J_{i+1}) of instances.

Pruning step. At this point, one may be tempted to think that $\delta' = (J_i)_{i \geq 0}$, with $J_i \langle \sigma_i, h_i \rangle J_{i+1}$ for $i \geq 0$, is the desired linear infinite $\star$-chase derivation of $\mathsf{cr}(\Sigma)$ w.r.t. Σ. It is easy to verify that we have the infinite sequence of atoms $(\rho(\alpha_i))_{i \geq k-1}$ such that $\rho(\alpha_{k-1}) \in \mathsf{cr}(\Sigma)$ since α_{k-1} is a side atom, and for each $i \geq k-1$, $J_{i-k+2} \supseteq J_{i-k+1} \cup \{\rho(\alpha_{i+1})\}$, and there exists $\beta \in \mathsf{body}(\sigma_i)$ such that $h_i(\beta_i) = \alpha_i$ and $h_i(\mathsf{body}(\sigma_i) \setminus \{\beta\}) \subseteq \mathsf{cr}(\Sigma)$. However, we cannot conclude yet that δ' is the desired derivation for the following two reasons:

1. triggers may repeat, i.e., we may have $i \neq j$ such that $\sigma_i = \sigma_j = \sigma$ and $h_i \diamond_\sigma^\star h_j$, where $\diamond_\sigma^\star$ is $=$ (resp., $\sim_\sigma$) if $\star = \mathsf{o}$ (resp., $\star = \mathsf{so}$), and
2. we may have $i \neq j$ such that $\rho(\alpha_i) = \rho(\alpha_j)$, i.e., the sequence of atoms $(\rho(\alpha_i))_{i \geq k-1}$ does not consist of distinct atoms.

This can be easily fixed by pruning the subderivation between the two repeated triggers or atoms. But since this pruning step may be applied infinitely many times, the question that comes up is whether the obtained $\star$-chase derivation δ'' is infinite. Interestingly, this is the case due to continuity. Since the backbone $(\alpha_i)_{i \geq k}$ is part of a continuous δ-path, we conclude that two repeated triggers or atoms are necessarily between two atoms α and β in which new nulls are invented. The fact that we have infinitely many pairs of such atoms on the backbone, we immediately conclude that after the pruning step the obtained $\star$-chase derivation is infinite. Thus, δ'' is a linear infinite $\star$-chase derivation of J_0 w.r.t. Σ. Since $J_0 \subseteq \mathsf{cr}(\Sigma)$, we can easily construct a linear infinite $\star$-chase derivation δ_ℓ of $\mathsf{cr}(\Sigma)$ w.r.t. Σ by simply adding to J_0 the set of atoms $\mathsf{cr}(\Sigma) \setminus J_0$, and the claim follows.

▶ **Example 11.** Coming back to our running example, it can be seen that the sequence of instances devised in Example 10 is not the desired linear derivation due to repeated triggers and atoms. However, after applying the pruning step, we get the sequence of instances

$$J'_0 = J_0,\ J'_1 = J_1,\ J'_2 = J'_1 \cup \{P(\bot_3, c, \bot_6), R(c, \bot_6)\},\ J'_3 = J'_2 \cup \{P(\bot_6, c, \bot_9), R(c, \bot_9)\}, \ldots.$$

Now, it is easy to verify that after adding the atom $S(c)$ in J'_0, we get (modulo null renaming) the linear infinite o-chase derivation of $\mathsf{cr}(\Sigma)$ w.r.t. Σ given in Example 6.

5 Graph-Based Characterization of Chase Termination

In this section, we characterize the termination of the (semi-)oblivious chase for sticky sets of tgds via graph-based conditions. More precisely, we show that a set $\Sigma \in \mathbb{S}$ belongs to $\mathbb{CT}^\star_\forall$ iff a linearized version of it, i.e., a set of linear tgds obtained from Σ, enjoys a condition similar to rich-acyclicity [16], if $\star = \mathsf{o}$, and weak-acyclicity [10], if $\star = \mathsf{so}$. Recall that linear tgds are tgds with only one body atom [6]; we write $\mathbb{L}$ for the class of linear tgds. The proof of the above result proceeds in two steps:

1. We first show that the given sticky set Σ of tgds can be rewritten into a set of linear tgds, while this rewriting preserves chase termination. This heavily relies on Theorem 7, which establishes that non-termination of the (semi-)oblivious chase coincides with the existence of a linear infinite chase derivation of $\mathsf{cr}(\Sigma)$ w.r.t. Σ.
2. We then extend recent characterizations from [4], which are based on extensions of rich-acyclicity and weak-acyclicity, of the termination of the (semi-)oblivious chase under constant-free linear tgds in order to deal with constants in the tgds. Although in other contexts, e.g., query answering under tgds, the transition from constant-free tgds to tgds with constants is relatively straightforward, in the context of chase termination the constants in the tgds cause additional complications that must be carefully treated.

We proceed to give more details for the above two steps.

5.1 Linearization

Before presenting the linearization procedure, we need to introduce some auxiliary notions. Given a tgd σ and an atom $\alpha \in \mathsf{body}(\sigma)$, let $V_{\alpha,\sigma} = \mathsf{var}(\mathsf{body}(\sigma) \setminus \{\alpha\})$, that is, the set of body variables of σ that do not occur only in α. Moreover, given a set Σ of tgds, a tgd $\sigma \in \Sigma$, and an atom $\alpha \in \mathsf{body}(\sigma)$, let $M^\Sigma_{\alpha,\sigma} = \{h \mid h : V_{\alpha,\sigma} \to \mathsf{dom}(\mathsf{cr}(\Sigma))\}$, i.e., the set of all possible mappings from the variables of $V_{\alpha,\sigma}$ to the constants occurring in $\mathsf{cr}(\Sigma)$.

▶ **Definition 12.** *Consider a set Σ of tgds. The* linearization *of a tgd $\sigma \in \Sigma$ (w.r.t. Σ) of the form $\phi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\, \psi(\bar{x}, \bar{z})$, denoted $\mathsf{Lin}(\sigma)$, is the set of linear tgds*

$$\bigcup_{\alpha \in \phi(\bar{x}, \bar{y})} \bigcup_{h \in M^{\Sigma}_{\alpha,\sigma}} \{h(\alpha) \rightarrow \exists \bar{z}\, h(\psi(\bar{x}, \bar{z}))\}.$$

The linearization of Σ is defined as $\mathsf{Lin}(\Sigma) = \bigcup_{\sigma \in \Sigma} \mathsf{Lin}(\sigma)$.

The linearization procedure converts a tgd σ into a set of linear tgds by keeping only one atom α from $\mathsf{body}(\sigma)$, while the variables in $\mathsf{body}(\sigma) \setminus \{\alpha\}$ are instantiated with constants from $\mathsf{cr}(\Sigma)$ in all the possible ways. Theorem 7 allows us to show that this procedure preserves the termination of the (semi-)oblivious chase whenever the input set of tgds is sticky.

▶ **Theorem 13.** *Consider a set $\Sigma \in \mathbb{S}$ of tgds. For $\star \in \{\mathsf{o}, \mathsf{so}\}$, $\Sigma \in \mathbb{CT}^{\star}_{\forall}$ iff $\mathsf{Lin}(\Sigma) \in \mathbb{CT}^{\star}_{\forall}$.*

The ($\Rightarrow$) direction is almost immediate. Let us focus on the ($\Leftarrow$) direction. Firstly, let us clarify that it suffices to show Theorem 13 for normalized sets of tgds, i.e., tgds with only one atom in the head. This holds since $\Sigma \in \mathbb{CT}^{\star}_{\forall}$ iff $\mathsf{Norm}(\Sigma) \in \mathbb{CT}^{\star}_{\forall}$, and $\mathsf{Lin}(\Sigma) \in \mathbb{CT}^{\star}_{\forall}$ iff $\mathsf{Lin}(\mathsf{Norm}(\Sigma)) \in \mathbb{CT}^{\star}_{\forall}$, where $\mathsf{Norm}(\Sigma)$ is the normalized version of Σ obtained by applying the standard normalization procedure; see, e.g., [7]. Assume that $\Sigma \notin \mathbb{CT}^{\star}_{\forall}$. By Theorem 7, there exists a linear infinite $\star$-chase derivation $\delta = (I_i)_{i \geq 0}$ of $\mathsf{cr}(\Sigma)$ w.r.t. Σ, where $I_i \langle \sigma_i, h_i \rangle I_{i+1}$ for $i \geq 0$. In other words, there exists an infinite sequence of distinct atoms $(\alpha_i)_{i \geq 0}$ such that: (i) $\alpha_0 \in \mathsf{cr}(\Sigma)$, and (ii) for each $i \geq 0$, $\alpha_{i+1} \in I_{i+1} \setminus I_i$, and there exists $\beta_i \in \mathsf{body}(\sigma_i)$ such that $h_i(\beta_i) = \alpha_i$ and $h_i(\mathsf{body}(\sigma_i) \setminus \{\beta_i\}) \subseteq \mathsf{cr}(\Sigma)$. Let Σ' be the set of linear tgds $\{g_i(\beta_i) \rightarrow g_i(\mathsf{head}(\sigma_i))\}_{i \geq 0}$, where g_i is the restriction of h_i over V_{β_i, σ_i}. Since $\Sigma' \subseteq \mathsf{Lin}(\Sigma)$, it suffices to show that $\Sigma' \notin \mathbb{CT}^{\star}_{\forall}$. To this end, we are going to construct an infinite $\star$-chase derivation δ' of $\mathsf{cr}(\Sigma') \subseteq \mathsf{cr}(\Sigma)$ w.r.t. Σ'. The derivation δ' is obtained from δ by replacing each trigger (σ_i, h_i), for $i \geq 0$, with (σ'_i, h'_i), where σ'_i is the linear tgd $g_i(\beta_i) \rightarrow g_i(\mathsf{head}(\sigma_i))$ that belongs to Σ', while h'_i is the restriction of h_i to the terms occurring in $\mathsf{body}(\sigma'_i)$. It remains to show that δ' is indeed a valid infinite $\star$-chase derivation of $\mathsf{cr}(\Sigma')$ w.r.t. Σ', which boils down to showing that, for every $i \neq j \geq 0$, $\sigma'_i = \sigma'_j = \sigma$ implies $h'_i \diamond^{\star}_{\sigma} h'_j$, where $\diamond^{\mathsf{o}}_{\sigma}$ is $\neq$, and $\diamond^{\mathsf{so}}_{\sigma}$ is $\not\sim_{\sigma}$. Towards a contradiction, assume that there exists $i \neq j$ such that $\sigma'_i = \sigma'_j = \sigma$ but $h'_i \diamond^{\star}_{\sigma} h'_j$ does not hold. This implies that $\alpha_i = \alpha_j$ since we consider single-head tgds. But this contradicts the fact that $\alpha_i \neq \alpha_j$.

5.2 Acyclicity Conditions

We proceed to extend the characterizations of the termination of the (semi-)oblivious chase under constant-free linear tgds established in [4]. The goal is to show that, given a set of tgds $\Sigma \in \mathbb{L}$, which may contain constants, $\Sigma \in \mathbb{CT}^{\mathsf{o}}_{\forall}$ iff Σ is critically-richly-acyclic, and $\Sigma \in \mathbb{CT}^{\mathsf{so}}_{\forall}$ iff Σ critically-weakly-acyclic, where critical-rich- and critical-weak-acyclicity are appropriate extensions of rich- and weak-acyclicity proposed in [4]. These notions rely on the dependency graph of a set of tgds, which we now recall. We assume a fixed order on the head-atoms of tgds. For a tgd σ with $\mathsf{head}(\sigma) = \alpha_1, \ldots, \alpha_k$, we write (σ, i) for the *single-head* tgd, i.e., the tgd with only one atom in its head, obtained from σ by keeping only the atom α_i, and the existentially quantified variables in α_i. Recall that $\mathsf{pos}(\alpha, x)$ is the set of positions in α at which x occurs. Analogously, we write $\mathsf{pos}(\mathsf{body}(\sigma), x)$ for the set of positions at which the variable x occurs in the body of σ. We also write $\mathsf{pos}(\mathsf{sch}(\Sigma))$ for the set of positions of $\mathsf{sch}(\Sigma)$, i.e., the set $\{R[i] \mid R/n \in \mathsf{sch}(\Sigma) \text{ and } i \in \{1, \ldots, n\}\}$.

▶ **Definition 14.** *The* dependency graph *of a set Σ of tgds is a labeled directed multigraph $\mathsf{dg}(\Sigma) = (N, E, \lambda)$, where $N = \mathsf{pos}(\mathsf{sch}(\Sigma))$, $\lambda : E \to \Sigma \times \mathbb{N}$, and E contains only the following edges. For each $\sigma \in \Sigma$ with $\mathsf{head}(\sigma) = \alpha_1, \ldots, \alpha_k$, for each $x \in \mathsf{fr}(\sigma)$, and for each $\pi \in \mathsf{pos}(\mathsf{body}(\sigma), x)$:*

- *For each $i \in [k]$, and for each $\pi' \in \mathsf{pos}(\alpha_i, x)$, there is a* normal *edge $e = (\pi, \pi') \in E$ with $\lambda(e) = (\sigma, i)$.*
- *For each existentially quantified variable z in σ, for each $i \in [k]$, and for each $\pi' \in \mathsf{pos}(\alpha_i, z)$, there is a* special *edge $e = (\pi, \pi') \in E$ with $\lambda(e) = (\sigma, i)$.*

A normal edge (π, π') keeps track of the fact that a term may propagate from π to π' during the chase. Moreover, a special edge (π, π'') keeps track of the fact that the propagation of a value from π to π' also creates a null at position π''. As we shall see, the dependency graph is appropriate when we consider the semi-oblivious chase. For the oblivious chase, we need an extended version of it. The *extended dependency graph* of Σ, denoted $\mathsf{edg}(\Sigma)$, is obtained from $\mathsf{dg}(\Sigma)$ by simply adding special labeled edges from the positions where non-frontier variables occur to the positions where existentially quantified variables occur.

Two well-known classes of tgds, introduced in the context of data exchange, that guarantee the termination of the oblivious and semi-oblivious chase are rich-acyclicity and weak-acyclicity, respectively. A set Σ is richly-acyclic (resp., weakly-acyclic) if there is no cycle in $\mathsf{edg}(\Sigma)$ (resp., $\mathsf{dg}(\Sigma)$) that contains a special edge. It would be very useful if, whenever we focus on linear tgds, rich- and weak-acyclicity are also necessary conditions for the termination of the oblivious and semi-oblivious chase, respectively. Unfortunately, this is not the case. A simple counterexample follows:

▶ **Example 15.** Consider the set Σ of linear tgds consisting of $R(x, x) \to \exists z\, R(z, x)$. In $\mathsf{dg}(\Sigma) = \mathsf{edg}(\Sigma)$ there is a cycle that contains a special edge. However, for $\star \in \{\mathsf{o}, \mathsf{so}\}$, there is only one $\star$-chase sequence of $\mathsf{cr}(\Sigma)$ w.r.t. Σ that is terminating; thus, $\Sigma \in \mathbb{CT}^{\star}_{\forall}$.

As it has been shown in [4], there is an extension of rich- and weak-acyclicity, called critical-rich- and critical-weak-acyclicity, that whenever we focus on linear tgds provides a necessary and sufficient condition for the termination of the oblivious and semi-oblivious chase, respectively. However, the analysis performed in [4] considers only tgds without constants, while after the linearization of a sticky set Σ of tgds, even if Σ is constant-free, the obtained set $\mathsf{Lin}(\Sigma)$ contains at least one constant. Thus, in order to be able to apply critical-rich- and critical-weak-acyclicity on $\mathsf{Lin}(\Sigma)$, we first need to appropriately extend these notions to linear tgds with constants.

A crucial notion underlying critical-rich- and critical-weak-acyclicity is the notion of compatibility among two single-head linear tgds. Intuitively, if a single-head linear tgd σ_1 is compatible with a single-head linear tgd σ_2, then the atom obtained during the chase by applying σ_1 may trigger σ_2. It is clear that the presence of constants in the tgds affects the way that we define compatibility. We assume the reader is familiar with the notion of unification. Given two atoms α, β, we write $\mathsf{mgu}(\alpha, \beta)$ for their most general unifier. For brevity, we write Π^{σ}_{t} for the set of positions $\mathsf{pos}(\mathsf{body}(\sigma), t)$, i.e., the set of positions at which the term t occurs in the body of σ. We also write $\mathsf{term}(\alpha, \Pi)$, where α is an atom, and Π a set of positions, for the set of terms occurring in α at positions of Π.

▶ **Definition 16.** *Consider two single-head linear tgds σ_1 and σ_2. We say that σ_1 is* compatible *with σ_2 if the following hold:*

1. $\mathsf{head}(\sigma_1)$ *and* $\mathsf{body}(\sigma_2)$ *unify.*
2. *For each $x \in \mathsf{var}(\mathsf{body}(\sigma_2))$, either $\mathsf{term}(\mathsf{head}(\sigma_1), \Pi^{\sigma_2}_{x}) = \{z\}$ for some existentially quantified variable z in σ_1, or $\mathsf{term}(\mathsf{head}(\sigma_1), \Pi^{\sigma_2}_{x}) \subseteq \mathsf{fr}(\sigma_1) \cup \{c\}$ for some constant c.*
3. *For each $c \in \mathsf{const}(\mathsf{body}(\sigma_2))$, $\mathsf{term}(\mathsf{head}(\sigma_1), \Pi^{\sigma_2}_{c}) \subseteq \mathsf{fr}(\sigma_1) \cup \{c\}$.*

Having the notion of compatibility among two single-head linear tgds in place, we can recall the resolvent of a sequence $\sigma_1, \ldots, \sigma_n$ of single-head linear tgds, which is in turn a single-head tgd. Roughly, such a resolvent mimics the behavior of the sequence $\sigma_1, \ldots, \sigma_n$ during the chase. Notice that the existence of such a resolvent is not guaranteed, but if it exists, this implies that we may have a sequence of trigger applications that involve the tgds $\sigma_1, \ldots, \sigma_n$ in this order. In such a case, we call the sequence $\sigma_1, \ldots, \sigma_n$ active.

▶ **Definition 17.** *The* resolvent *of a sequence $\sigma_1, \ldots, \sigma_n$ of single-head linear tgds, denoted $[\sigma_1, \ldots, \sigma_n]$, is inductively defined as follows; for brevity, we write ρ for $[\sigma_1, \ldots, \sigma_{n-1}]$:*

1. $[\sigma_1] = \sigma_1$*;*
2. $[\sigma_1, \ldots, \sigma_n] = \gamma(\mathsf{body}(\rho)) \to \gamma(\mathsf{head}(\sigma_n))$*, where* $\gamma = \mathsf{mgu}(\mathsf{head}(\rho), \mathsf{body}(\sigma_n))$*, if* $\rho \neq \Diamond$ *and ρ is compatible with σ_n; otherwise,* $[\sigma_1, \ldots, \sigma_n] = \Diamond$*.*

The sequence $\sigma_1, \ldots, \sigma_n$ is called active *if* $[\sigma_1, \ldots, \sigma_n] \neq \Diamond$*.*

At this point, one may think that the right extension of rich- and weak-acyclicity, which will provide a necessary condition for the termination of the oblivious and semi-oblivious chase under linear tgds, is to allow cycles with special edges in the underlying dependency graph as long as the corresponding sequence of single-head tgds, which can be extracted from the edge labels, is not active. This is not enough. If a cycle with a special edge is labeled with an active sequence, then we can conclude that it will be traversed at least once during the chase. However, it is not guaranteed that it will be traversed infinitely many times.

▶ **Example 18.** Consider the set Σ of linear tgds consisting of

$$\sigma_1 = R(x, y, z) \;\to\; P(x, y, z) \qquad \sigma_2 = P(x, y, x) \;\to\; \exists z\, R(y, z, x).$$

In $\mathsf{dg}(\Sigma) = \mathsf{edg}(\Sigma)$ there is an active cycle that contains a special edge; e.g., $C = R[2], P[2], R[2]$, which corresponds to the sequence of tgds σ_1, σ_2. It is easy to see that $[\sigma_1, \sigma_2] \neq \Diamond$, and thus C is active. Despite the existence of an active cycle that contains a special edge, we can show that $\Sigma \in \mathbb{CT}^{\star}_{\forall}$, where $\star \in \{\mathsf{o}, \mathsf{so}\}$.

A cycle that is labeled with an active sequence $\sigma_1, \ldots, \sigma_n$, and contains a special edge, will be certainly traversed infinitely many times if the resolvent of the sequence $\rho, \ldots, \rho$ of length k, where $\rho = [\sigma_1, \ldots, \sigma_n]$, exists, for every $k > 0$. Interestingly, for ensuring the latter condition, it suffices to consider sequences of length at most $(\omega + 1)$, where ω is the arity of the predicate of $\mathsf{body}(\sigma_1)$. This brings us to critical sequences. For brevity, we write σ^k for the sequence $\sigma, \ldots, \sigma$ of length k.

▶ **Definition 19.** *A sequence $\sigma_1, \ldots, \sigma_n$ of single-head linear tgds is* critical *if $\sigma_1, \ldots, \sigma_n$ is active, and $[\sigma_1, \ldots, \sigma_n]^{\omega+1}$ is active, where ω is the arity of the predicate of* $\mathsf{body}(\sigma_1)$*.*

We can now recall critical-rich- and critical-weak-acyclicity. They are essentially rich- and weak-acyclicity, with the difference that a cycle in the underlying graph is "dangerous", not only if it contains a special edge, but if it is also labeled with a critical sequence.

▶ **Definition 20.** *Consider a set $\Sigma \in \mathbb{L}$ of tgds, and let $G = (N, E, \lambda)$ be either* $\mathsf{edg}(\Sigma)$ *or* $\mathsf{dg}(\Sigma)$*. A cycle $v_0, v_1, \ldots, v_n, v_0$ in G is* critical *if $\lambda(v_0, v_1), \lambda(v_1, v_2), \ldots, \lambda(v_n, v_0)$ is critical. We say that Σ is* critically-richly-acyclic *(resp.,* critically-weakly-acyclic*), if no critical cycle in* $\mathsf{edg}(\Sigma)$ *(resp.,* $\mathsf{dg}(\Sigma)$*) contains a special edge.*

The desired result follows:

▶ **Theorem 21.** *Consider a set $\Sigma \in \mathbb{L}$ of tgds. The following hold:*

- $\Sigma \in \mathbb{CT}^{\mathsf{o}}_{\forall}$ *iff* Σ *is critically-richly-acyclic.*
- $\Sigma \in \mathbb{CT}^{\mathsf{so}}_{\forall}$ *iff* Σ *is critically-weakly-acyclic.*

The "if" directions of the above result are shown by giving proofs similar to the ones given in [16] and [10] for showing that rich-acyclicity and weak-acyclicity guarantees the termination of the oblivious and restricted chase, respectively. The interesting direction is the "only if" direction. By Corollary 4, it suffices to show that if Σ is not critically-richly-acyclic (resp., critically-weakly-acyclic), then there exists an infinite o-chase (resp., so-chase) derivation of $\mathsf{cr}(\Sigma)$ w.r.t. Σ. This is a non-trivial result that requires a couple of auxiliary lemmas.

The equality type of an atom is a set of equalities among positions, as well as among positions and constants, that describes its shape. Formally, for an atom $\alpha = R(t_1, \ldots, t_n)$, the *equality type* of α is $\mathsf{eqtype}(\alpha) = \{R[i] = R[j] \mid t_i = t_j\} \cup \{R[i] = c \mid c \in \mathbf{C} \text{ and } t_i = c\}$. For a linear tgd σ, we write $\mathsf{eqtype}(\sigma)$ for the equality type of the atom $\mathsf{body}(\sigma)$. The next result establishes a useful connection between active sequences and equality types:

▶ **Lemma 22.** *Consider a single-head linear tgd σ such that σ^i is active, for some $i > 1$, and $\mathsf{eqtype}([\sigma^{i-1}]) = \mathsf{eqtype}([\sigma^i])$. Then, σ^{i+1} is active, and $\mathsf{eqtype}([\sigma^i]) = \mathsf{eqtype}([\sigma^{i+1}])$.*

Despite the fact that the above lemma has been already shown in [4] for constant-free tgds, it turned out that the proof from [4] cannot be easily extended to tgds with constants. Thus, we had to devise a completely new proof that exploits further properties of the resolvent of a sequence $\sigma, \ldots, \sigma$. In fact, we show via an inductive argument that $[\sigma^i] = [[\sigma^{i-1}], \sigma]$ is the same (modulo variable renaming) as $[\sigma, [\sigma^{i-1}]]$, which in turn allows us to easily establish Lemma 22. Having the connection between active sequences and equality types provided by Lemma 22, we show the next lemma, which states that critical cycles can be traversed infinitely many times during the chase.

▶ **Lemma 23.** *Consider a critical sequence $\sigma_1, \ldots, \sigma_n$ of single-head linear tgds. For every $k > 0$, $[\sigma_1, \ldots, \sigma_n]^k$ is active.*

As for Lemma 22, even though the above result has been shown in [4] for constant-free tgds, we had to provide a new proof in order to deal with the constants in the tgds. Let us briefly explain how Lemma 23 is shown. For brevity, let $\rho = [\sigma_1, \ldots, \sigma_n]$. Since $\sigma_1, \ldots, \sigma_n$ is critical, by definition we get that $\rho^{\omega+1}$ is active, where ω is the arity of the predicate of $\mathsf{body}(\sigma_1)$. The crucial step is to also show that $\mathsf{eqtype}([\rho^\omega]) = \mathsf{eqtype}([\rho^{\omega+1}])$. Then, by iteratively applying Lemma 22, we obtain that ρ^k is active for every $k > \omega + 1$. Since $\rho^{\omega+1}$ is active, we can conclude that ρ^k is active for every $1 \leq k \leq \omega + 1$, and Lemma 23 follows.

We can show via an inductive argument that an active sequence $\sigma_1, \ldots, \sigma_n$ of single-head linear tgds mimics the sequence of trigger applications that involve the tgds $\sigma_1, \ldots, \sigma_n$ (in this order), starting from an atom in the critical instance; in particular, the ground version of $\mathsf{body}([\rho^{\omega+1}])$. This fact and Lemma 23 allow us to show that a critical cycle of minimal length in the (extended) dependency graph that contains a special edge, gives rise to an infinite o-chase (so-chase) derivation of $\mathsf{cr}(\Sigma)$ w.r.t. Σ, and Theorem 21 follows.

It is now easy to see that Theorems 13 and 21 establish the main result of this section:

▶ **Corollary 24.** *Consider a set $\Sigma \in \mathbb{S}$ of tgds. The following hold:*

- $\Sigma \in \mathbb{CT}^{\mathsf{o}}_{\forall}$ *iff* $\mathsf{Lin}(\Sigma)$ *is critically-richly-acyclic.*
- $\Sigma \in \mathbb{CT}^{\mathsf{so}}_{\forall}$ *iff* $\mathsf{Lin}(\Sigma)$ *is critically-weakly-acyclic.*

6 Complexity of Chase Termination

In this final section, we pinpoint the complexity of the $\star$-chase termination problem under sticky sets of tgds. In particular, we establish the following complexity result:

▶ **Theorem 25.** *For* $\star \in \{\mathsf{o}, \mathsf{so}\}$, $\mathsf{CT}^{\star}_{\forall}(\mathbb{S})$ *is* PSPACE*-complete, and* NLOGSPACE*-complete for predicates of bounded arity. The lower bounds hold even for tgds without constants.*

Upper Bounds. The problem $\mathsf{CT}^{\star}_{\forall}$ under constant-free linear tgds is PSPACE-complete, in general, and NLOGSPACE-complete for predicates of bounded arity [4]. However, despite the fact that, by Corollary 24, we can reduce $\mathsf{CT}^{\star}_{\forall}(\mathbb{S})$ to $\mathsf{CT}^{\star}_{\forall}(\mathbb{L})$, we cannot directly exploit the complexity results from [4] for two reasons: (i) the linearized version of Σ contains at least one constant, while the results from [4] apply only to constant-free tgds, and (ii) the linearization procedure takes exponential time, in general, and polynomial time in the case of bounded-arity predicates; thus, we cannot explicitly compute the set $\mathsf{Lin}(\Sigma)$, and then check for critical-rich- and critical-weak-acyclicity. Therefore, a more refined procedure is needed.

We focus on the complement of our problem, i.e., given a set $\Sigma \in \mathbb{S}$ of tgds, we want to check whether $\Sigma \notin \mathbb{CT}^{\star}_{\forall}$. By Corollary 24, it suffices to show that $\mathsf{Lin}(\Sigma)$ is not critically-richly-acyclic, if $\star = \mathsf{o}$, and not critically-weakly-acyclic, if $\star = \mathsf{so}$. The latter problems can be seen as a generalization of the standard graph reachability problem. Indeed, we need to check whether there exists a node v in the (extended) dependency graph of $\mathsf{Lin}(\Sigma)$ that is reachable from itself via a critical cycle that contains a special edge. However, as discussed above, we cannot explicitly construct $\mathsf{Lin}(\Sigma)$ and its (extended) dependency graph G. Instead, the above reachability check should be performed on a compact representation of G, which is the set Σ itself. We show that this check can be performed via a non-deterministic procedure that uses $O(\omega \log(\omega \cdot |\mathsf{sch}(\Sigma)|) + \omega \log(\omega \cdot m \cdot |\Sigma|))$ space, where ω is the maximum arity over all predicates in Σ, and m is the maximum number of atoms occurring in a tgd of Σ.

Lower Bounds. The PSPACE-hardness is shown by providing a polynomial time reduction from the acceptance problem of a deterministic polynomial space Turing machine M. Such a reduction can be easily devised if we are allowed to join a variable in the body of a tgd and then lose it, or if we can use constants in the body of a tgd. In this case, a configuration of M can be straightforwardly encoded in a single predicate *Config* of polynomial arity. However, if we want the set of tgds to be sticky and constant-free, then we need a more clever encoding for a configuration of M, which increases the arity of *Config*, but only polynomially.

The NLOGSPACE-hardness is inherited from [4], where it is shown that $\mathsf{CT}^{\star}_{\forall}(\mathbb{L})$ is NLOGSPACE-hard, even for tgds that are constant-free, each body variable occurs only once (i.e., stickiness is trivially satisfied), and only unary and binary predicates are used.

7 Conclusions

We have shown that the uniform (semi-)oblivious chase termination problem for sticky sets of tgds is decidable, and obtained precise complexity results. This is done by first characterizing the termination of the (semi-)oblivious chase for sticky sets of tgds via graph-based conditions that are of independent interest. In particular, to check whether the oblivious (resp., semi-oblivious) chase terminates for a sticky set Σ of tgds, we simply need to linearize it, i.e., convert it, via an easy procedure, into a set $\mathsf{Lin}(\Sigma)$ of linear tgds, and then check whether $\mathsf{Lin}(\Sigma)$ enjoys an acyclicity condition in the spirit of rich-acyclicity (resp., weak-acyclicity). The next natural step is to concentrate on the restricted (a.k.a. standard) version of the chase, which makes the problem even more challenging due to its non-deterministic behaviour that cannot be captured via static graph-based conditions.

References

1. Alfred V. Aho, Yehoshua Sagiv, and Jeffrey D. Ullman. Efficient Optimization of a Class of Relational Expressions. *ACM Trans. Database Syst.*, 4(4):435–454, 1979.
2. Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. On rules with existential variables: Walking the decidability line. *Artif. Intell.*, 175(9-10):1620–1654, 2011.
3. Catriel Beeri and Moshe Y. Vardi. A Proof Procedure for Data Dependencies. *J. ACM*, 31(4):718–741, 1984.
4. Marco Calautti, Georg Gottlob, and Andreas Pieris. Chase Termination for Guarded Existential Rules. In *PODS*, pages 91–103, 2015.
5. Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. *J. Artif. Intell. Res.*, 48:115–174, 2013.
6. Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general Datalog-based framework for tractable query answering over ontologies. *J. Web Sem.*, 14:57–83, 2012.
7. Andrea Calì, Georg Gottlob, and Andreas Pieris. Towards more expressive ontology languages: The query answering problem. *Artif. Intell.*, 193:87–128, 2012.
8. Alin Deutsch, Alan Nash, and Jeff B. Remmel. The Chase Revisisted. In *PODS*, pages 149–158, 2008.
9. Alin Deutsch and Val Tannen. Reformulation of XML Queries and Constraints. In *ICDT*, pages 225–241, 2003.
10. Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.
11. Tomasz Gogacz and Jerzy Marcinkowski. All-Instances Termination of Chase is Undecidable. In *ICALP*, pages 293–304, 2014.
12. Gösta Grahne and Adrian Onet. Anatomy of the Chase. *Fundam. Inform.*, 157(3):221–270, 2018.
13. Bernardo Cuenca Grau, Ian Horrocks, Markus Krötzsch, Clemens Kupke, Despoina Magka, Boris Motik, and Zhe Wang. Acyclicity Notions for Existential Rules and Their Application to Query Answering in Ontologies. *J. Artif. Intell. Res.*, 47:741–808, 2013.
14. Sergio Greco, Cristian Molinaro, and Francesca Spezzano. *Incomplete Data and Data Dependencies in Relational Databases*. Morgan & Claypool Publishers, 2012.
15. Sergio Greco, Francesca Spezzano, and Irina Trubitsyna. Stratification Criteria and Rewriting Techniques for Checking Chase Termination. *PVLDB*, 4(11):1158–1168, 2011.
16. André Hernich and Nicole Schweikardt. CWA-solutions for data exchange settings with target dependencies. In *PODS*, pages 113–122, 2007.
17. David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. Testing Implications of Data Dependencies. *ACM Trans. Database Syst.*, 4(4):455–469, 1979.
18. Bruno Marnette. Generalized schema-mappings: from termination to tractability. In *PODS*, pages 13–22, 2009.
19. Michael Meier, Michael Schmidt, and Georg Lausen. On Chase Termination Beyond Stratification. *PVLDB*, 2(1):970–981, 2009.

A Single Approach to Decide Chase Termination on Linear Existential Rules

Michel Leclère
University of Montpellier, CNRS, Inria, LIRMM, France
leclere@lirmm.fr

Marie-Laure Mugnier
University of Montpellier, CNRS, Inria, LIRMM, France
mugnier@lirmm.fr

Michaël Thomazo
Inria, DI ENS, ENS, CNRS, PSL University, France
michael.thomazo@inria.fr

Federico Ulliana
University of Montpellier, CNRS, Inria, LIRMM, France
ulliana@lirmm.fr

Abstract

Existential rules, long known as tuple-generating dependencies in database theory, have been intensively studied in the last decade as a powerful formalism to represent ontological knowledge in the context of ontology-based query answering. A knowledge base is then composed of an instance that contains incomplete data and a set of existential rules, and answers to queries are logically entailed from the knowledge base. This brought again to light the fundamental chase tool, and its different variants that have been proposed in the literature. It is well-known that the problem of determining, given a chase variant and a set of existential rules, whether the chase will halt on any instance, is undecidable. Hence, a crucial issue is whether it becomes decidable for known subclasses of existential rules. In this work, we consider linear existential rules with atomic head, a simple yet important subclass of existential rules that generalizes inclusion dependencies. We show the decidability of the *all-instance* chase termination problem on these rules for three main chase variants, namely *semi-oblivious*, *restricted* and *core* chase. To obtain these results, we introduce a novel approach based on so-called derivation trees and a single notion of forbidden pattern. Besides the theoretical interest of a unified approach and new proofs for the semi-oblivious and core chase variants, we provide the first positive decidability results concerning the termination of the restricted chase, proving that chase termination on linear existential rules with atomic head is decidable for both versions of the problem: Does *every* chase sequence terminate? Does *some* chase sequence terminate?

2012 ACM Subject Classification Theory of computation → Logic; Computing methodologies → Knowledge representation and reasoning

Keywords and phrases Chase, Tuple Generating Dependencies, Existential rules, Decidability

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.18

Related Version `https://arxiv.org/abs/1810.02132`

Acknowledgements We thank the reviewers for their insightful comments, as well as Antoine Amarilli for Example 21, which is simpler than a previous example and shows that the fairness issue already occurs with binary predicates.

1 Introduction

The chase procedure is a fundamental tool for solving many issues involving tuple-generating dependencies, such as data integration [21], data-exchange [11], query answering using views [16] or query answering on probabilistic databases [24]. In the last decade, tuple-generating dependencies raised a renewed interest under the name of *existential rules* for the problem known as ontology-based query answering. In this context, the aim is to query a knowledge base (I, Σ), where I is an instance and Σ is an ontology given as a set of existential rules (see e.g. the survey chapters [5, 23]). In more classical database terms, this problem can be recast as querying an instance I under open-world assumption, provided with a set of constraints Σ, which are tuple-generating dependencies. The chase is a fundamental tool to solve dependency-related problems as it allows one to compute a (possibly infinite) *universal model* of (I, Σ), *i.e.*, a model that can be homomorphically mapped to any other model of (I, Σ). Hence, the answers to a conjunctive query (and more generally to any kind of query closed by homomorphism) over (I, Σ) can be defined by considering solely this universal model.

Several variants of the chase have been introduced, and we focus in this paper on the main ones: semi-oblivious [22] (aka Skolem [22]), restricted [3, 11] (aka standard [25]) and core [9]. Any chase variant starts from an instance and exhaustively performs a sequence of non-redundant rule applications according to a redundancy criterion which characterizes the variant itself. The built sequence is required to be *fair*, i.e., no rule application deemed non-redundant can be indefinitely left out. It is well known that all the above variants produce homomorphically equivalent results but obey increasingly stronger redundancy criteria, hence terminate for increasingly larger subclasses of existential rules.

The question of whether a chase variant terminates on *all instances* for a given set of existential rules is known to be undecidable when there is no restriction on the kind of rules [1, 12]. A number of *sufficient* syntactic conditions for termination have been proposed in the literature for the semi-oblivious chase (see e.g. [25, 15, 27] for syntheses), as well as for the restricted chase [8] (note that the latter paper also defines a sufficient condition for non-termination). However, only few positive results exist regarding the termination of the chase on specific classes of rules. Decidability was shown for the semi-oblivious chase on guarded-based rules (linear rules, and their extension to (weakly-)guarded rules) [4]. Decidability of the core chase termination on guarded rules for a fixed instance was shown in [17].

In this work, we provide new insights on the chase termination problem for *linear* existential rules with atomic head, a simple yet important subclass of guarded existential rules, which generalizes inclusion dependencies [10] and some practical ontological languages [6]. When we are interested in the ontology-based query answering problem, we note that linear rules are first-order rewritable, hence answering conjunctive queries on linear rule knowledge bases can be solved by query rewriting [1, 13, 18]. However, it is well known that the size and the unusual form of the rewritten query may lead to practical efficiency issues. Hence, the materialization of ontological inferences in the data is often an effective alternative to query rewriting, provided that some chase algorithm terminates. Finally, having the choice of how to process a set of linear rules extends the applicability of query answering techniques that combine query rewriting and materialization [1].

Precisely, the question of whether a chase variant terminates on all instances for a set of linear existential rules with atomic head is studied in two fashions:

- does *every* chase sequence terminate?
- does *some* chase sequence terminate?

It is well-known that these two questions have the same answer for the semi-oblivious and the core chase variants, but not for the restricted chase. Indeed, this last one may admit both terminating and non-terminating sequences over the same knowledge base. We show that the termination problem is decidable for linear existential rules with atomic head, no matter which version of the problem and which chase variant we consider.

We study chase termination by exploiting in a novel way a graph structure, namely the *derivation tree*, which was originally introduced to solve the ontology-based (conjunctive) query answering problem for the family of greedy-bounded treewidth sets of existential rules [2, 28], a class that generalizes guarded-based rules and in particular linear rules. We first use derivation trees to show the decidability of the termination problem for the semi-oblivious and restricted chase variants, and then generalize them to *entailment trees* to show the decidability of termination for the core chase. For every chase variant we consider, we adopt the same high-level procedure: starting from a finite set of canonical instances (representative of all possible instances), we build a (set of) tree structures for each canonical instance, while forbidding the occurrence of a specific pattern, that we call *unbounded-path witness*. The structures that we build are finite thanks to this forbidden pattern, and this allows us to decide if the chase terminates on the associated canonical instance. By doing so, we obtain a uniform approach to study the termination of several chase variants, which we believe to be of theoretical interest per se. In particular, some important differences in the chase behaviors become more visible. The derivation tree is moreover a simple structure, which makes the algorithms built on it effectively implementable. Let us also point out that our approach is constructive: if the chase terminates on a given instance, the algorithm that decides termination actually computes the result of the chase (or a superset of it in the case of the core chase), otherwise it pinpoints a forbidden pattern responsible for non-termination.

Besides providing new theoretical tools to study chase termination for linear existential rules with atomic head, we obtain the following results:

- a new proof of the decidability of the semi-oblivious chase termination, building on different objects than the previous proof provided in [4]; we show that our algorithm provides the same complexity upper-bound;
- the decidability of the restricted chase termination, for both versions of the problem, i.e., termination of all chase sequences and termination of some chase sequence; to the best of our knowledge, these are the first positive results on the decidability of the restricted chase termination;
- a new proof of the decidability of the core chase termination, with different objects than previous work reported in [17]; although this latter paper solves the question of the core chase termination given a fixed instance, the results actually allow to infer the decidability of the *all*-instance version of the problem (still on linear rules), by noticing that only a finite number of instances need to be considered (see the next section).

The paper is organized as follows. After introducing some preliminary notions (Section 2), we define the main components of our framework, namely derivation trees and unbounded-path witnesses (Section 3). We build on these objects to prove the decidability of the semi-oblivious and restricted chase termination (Section 4). Finally, we generalize derivation-trees to entailment trees and use them to prove the decidability of the core chase termination (Section 5). Detailed proofs are provided in [19]. A previous version of this work was presented as an extended abstract in [20].

2 Preliminaries

We consider a logical *vocabulary* composed of a finite set of predicates and an infinite set of constants. An *atom* α has the form $r(t_1, \ldots, t_n)$ where r is a predicate of arity n and the t_i are terms (i.e., variables or constants). We denote by $\mathit{terms}(\alpha)$ (resp. $\mathit{vars}(\alpha)$) the set of terms (resp. variables) in α and extend the notations to a set of atoms. A *ground* atom does not contain any variable. It is convenient to identify (the existential closure of) a conjunction of atoms with the set of these atoms. A set of atoms is *atomic* if it contains a single atom. An *instance* is a set of (non-necessarily ground) atoms, which is finite unless otherwise specified. Abusing terminology, we will often see an instance as its isomorphic model.

Given two sets of atoms S and S', a *homomorphism* from S' to S is a substitution π of $\mathit{vars}(S')$ by $\mathit{terms}(S)$ such that $\pi(S') \subseteq S$. It holds that $S \models S'$ (where $\models$ denotes classical logical entailment) iff there is a homomorphism from S' to S. An *endomorphism* of S is a homomorphism from S to itself. A set of atoms is a *core* if it admits only injective endomorphisms. Any finite set of atoms is logically (or: homomorphically) equivalent to one of its subsets that is a core, and this core is unique up to isomorphism (i.e., bijective variable renaming). Given sets of atoms S and S', we say that S' *folds* onto S if there is a homomorphism π from $S' \setminus S$ to S such that π is the identity on $\mathit{vars}(S)$. The homomorphism π is called a *folding*. In particular, it is well-known that any set of atoms *folds* onto its core.

An *existential rule* (or simply *rule*) is of the form $\sigma = \forall \mathbf{x} \forall \mathbf{y}\, [\mathrm{body}(\mathbf{x}, \mathbf{y}) \rightarrow \exists \mathbf{z}\, \mathrm{head}(\mathbf{x}, \mathbf{z})]$ where $\mathbf{x}, \mathbf{y}$ and $\mathbf{z}$ are pairwise disjoint tuples of variables, $\mathrm{body}(\mathbf{x}, \mathbf{y})$ and $\mathrm{head}(\mathbf{x}, \mathbf{z})$ are non-empty conjunctions of atoms respectively on $\mathbf{x} \cup \mathbf{y}$ and $\mathbf{x} \cup \mathbf{z}$, called the *body* and the *head* of the rule, and also denoted by $\mathrm{body}(\sigma)$ and $\mathrm{head}(\sigma)$. The variables of $\mathbf{z}$ are called *existential variables*. The variables of $\mathbf{x}$ form the *frontier* of σ, which is also denoted by $\mathrm{fr}(\sigma)$. For brevity, we will omit universal quantifiers in the examples. A *knowledge base* (KB) is of the form $\mathcal{K} = (I, \Sigma)$, where I is an instance and Σ is a finite set of existential rules.

A rule $\sigma = \mathrm{body}(\sigma) \rightarrow \mathrm{head}(\sigma)$ is *applicable* to an instance I if there is a homomorphism π from $\mathrm{body}(\sigma)$ to I. The pair (σ, π) is called a *trigger* for I. The result of the application of σ according to π on I is the instance $I' = I \cup \pi^s(\mathrm{head}(\sigma))$, where π^s extends π by assigning a distinct fresh variable (also called a *null*) to each existential variable.[1] We also say that I' is obtained by *firing* the trigger (σ, π) on I. By $\pi_{|\mathrm{fr}(\sigma)}$ we denote the restriction of π to the domain $\mathrm{fr}(\sigma)$.

▶ **Definition 1** (Derivation). *A* Σ-derivation *(or simply* derivation *when* Σ *is clear from the context) from an instance* $I = I_0$ *to an instance* I_n *is a sequence* $I_0, (\sigma_1, \pi_1), I_1 \ldots, (\sigma_n, \pi_n), I_n$*, such that for all* $1 \leq i \leq n$*:* $\sigma_i \in \Sigma$*,* (σ_i, π_i) *is a trigger for* I_{i-1}*,* I_i *is obtained by firing* (σ_i, π_i) *on* I_{i-1}*, and* $I_i \neq I_{i-1}$*. The notion of derivation can be naturally extended to an* infinite *sequence.*

Note that the condition $I_i \neq I_{i-1}$ in the above definition implies that all triggers in a derivation produce distinct atoms. We briefly introduce below the main chase variants and refer to [25] for a detailed presentation.

The *semi-oblivious* chase prevents several applications of the same rule through the same mapping of its frontier. Given a derivation from I_0 to I_i, a trigger (σ, π) for I_i is said to be *active according to the semi-oblivious criterion*, if (1) there is no trigger (σ_j, π_j) in the

[1] The "s" superscript stands for *safe*, as newly introduced variables are fresh, hence cannot be confused with already existing variables.

derivation with $\sigma = \sigma_j$ and $\pi_{|\mathrm{fr}(\sigma)} = \pi_{j|\mathrm{fr}(\sigma_j)}$ and (2) the extension of I_i with (σ, π) remains a derivation, i.e., $\pi(\mathrm{head}(\sigma)) \not\subseteq I_i$. The *restricted* chase performs a rule application only if the added set of atoms is not redundant with respect to the current instance. Given a derivation from I_0 to I_i, a trigger (σ, π) for I_i is said to be *active according to the restricted criterion* if π cannot be extended to a homomorphism from $(\mathrm{body}(\sigma) \cup \mathrm{head}(\sigma))$ to I_i (equivalently, $\pi^s(\mathrm{head}(\sigma))$ does not fold onto I_i). We say that a (possibly infinite) derivation is a *semi-oblivious (resp. restricted) derivation* if it is built by firing only active triggers according to the semi-oblivious (resp. restricted) criterion. A (possibly infinite) derivation is *fair* if no firing of an active trigger is indefinitely delayed; formally: if some I_i in the derivation admits an active trigger (σ, π), then there is $j > i$ such that, either I_j is obtained by firing (σ, π) on I_{j-1}, or (σ, π) is not an active trigger anymore on I_j. A *terminating* derivation is a finite fair derivation (in other words, a finite derivation that does not admit any active trigger on its final instance). A semi-oblivious (resp. restricted) *chase sequence* is a fair semi-oblivious (resp. restricted) derivation. Hence, a chase sequence is terminating if and only if it is finite.

In its original definition [9], the *core* chase proceeds in a breadth-first manner, and, at each step, first fires in parallel all active triggers according to the restricted chase criterion, then computes the core of the result. Alternatively, to bring the definition of the core chase closer to the above definitions of the semi-oblivious and restricted chases, one can define a *core derivation* as a possibly infinite sequence $I_0, (\sigma_1, \pi_1), I_1, \ldots$, alternating instances and triggers, such that each (σ_i, π_i) is an active trigger on I_{i-1} according to the restricted criterion, and I_i is obtained from I_{i-1} by first firing (σ_i, π_i), then computing the core of the result. Then, a core chase sequence is a fair core derivation. An instance admits a terminating core chase sequence in that sense if and only if the core chase as originally defined terminates on that instance.

For the three chase variants, a chase sequence defines a (possibly infinite) *universal model* of the KB, but only the core chase stops if and only if the KB has a *finite* universal model.

It is well-known that, for the semi-oblivious and the core chase, if there is a terminating chase sequence from an instance I then all chase sequences from I are terminating. This is not the case for the restricted chase, since the order in which rules are applied has an impact on termination, as illustrated by Example 2.

▶ **Example 2.** Let $\Sigma = \{\sigma_1, \sigma_2\}$, with $\sigma_1 = p(x, y) \to \exists z\ p(y, z)$ and $\sigma_2 = p(x, y) \to p(y, y)$. Let $I = p(a, b)$. The KB (I, Σ) has a finite universal model, for example, $I^* = \{p(a, b), p(b, b)\}$. The semi-oblivious chase does not terminate on I as σ_1 is applied indefinitely, while the core chase terminates after one breadth-first step and returns I^*. The restricted chase has a terminating sequence, for example $I_0 = I, (\sigma_2, \{x \mapsto a, y \mapsto b\}), I_1 = I^*$, but it also has infinite sequences, for instance sequences that apply always σ_1 before σ_2 on a given atom.

We study the following problems for the semi-oblivious, restricted and core chase variants:

- *(All-instance) all-sequence termination:* Given a set of rules Σ, is it true that, for any instance, all chase sequences are terminating?
- *(All-instance) one-sequence termination:* Given a set of rules Σ, is it true that, for any instance, there is a terminating chase sequence?

Note that, according to the terminology of [14], these problems can be recast as deciding whether, for a chase variant, a given set of rules belongs to the class $\mathrm{CT}_{\forall\forall}$ or $\mathrm{CT}_{\forall\exists}$, respectively.

An existential rule is called *linear* if its body is atomic, see e.g., [6]. As often done, we moreover restrict linear rules to atomic heads (and still use the name linear rules). Linear rules generalize *inclusion dependencies* [10] by allowing several occurrences of the same

variable in an atom. They also generalize positive inclusions in the description logic DL-Lite$_\mathcal{R}$ (the formal basis of the web ontological language OWL 2 QL) [7], which can be seen as inclusion dependencies restricted to unary and binary predicates.

Note that the restriction of existential rules to rules with an atomic head is often made in the literature, considering that any existential rule with a complex head can be decomposed into several rules with an atomic head, by introducing a fresh predicate for each rule. However, while this translation preserves the termination of the semi-oblivious chase, it is not the case for the restricted and the core chases. Hence, considering linear rules with a complex head would require to extend the techniques developed in this paper.

To simplify the presentation, we assume in the following that each rule frontier is of size at least one. This assumption is made without loss of generality. [2]

We first point out that the termination problem on linear rules can be recast by considering solely atomic instances (as already remarked in several contexts).

▶ **Proposition 3.** *Let Σ be a set of linear rules. The semi-oblivious (resp. restricted, core) chase terminates on all instances if and only if it terminates on all atomic instances.*

We will furthermore rely on the following notion of the type of an atom.

▶ **Definition 4** (Type of an atom)**.** *The* type of an atom $\alpha = r(t_1, \ldots, t_n)$*, denoted by type*$(\alpha)$*, is the pair $(r, \mathcal{P})$ where $\mathcal{P}$ is the partition of $\{1, \ldots, n\}$ induced by term equality (i.e., i and j are in the same class of $\mathcal{P}$ iff $t_i = t_j$).*

There are finitely (more specifically, exponentially) many types for a given vocabulary. When two atoms α and α' have the same type, there is a *natural mapping* from α to α', denoted by $\varphi_{\alpha \to \alpha'}$, and defined as follows: it is a bijective mapping from *terms*(α) to *terms*(α'), that maps the i-th term of α to the i-th term of α'. Note that $\varphi_{\alpha \to \alpha'}$ may not be an isomorphism, as constants from α may not be mapped to themselves. However, if (σ, π) is a trigger for $\{\alpha\}$, then $(\sigma, \varphi_{\alpha \to \alpha'} \circ \pi)$ is a trigger for $\{\alpha'\}$, as there are no constants in the considered rules.

Together with Proposition 3, this implies that one can check all-instance all-sequence termination by checking all-sequence termination on a finite set of instances, called *canonical instances*: for each type, we take exactly one canonical instance that has this type.

We will consider different kinds of tree structures, which have in common to be *trees of bags*.

▶ **Definition 5** (Tree of bags)**.** *A* tree of bags *is a rooted tree whose nodes, called bags,*[3] *are labeled by an atom. For any bag B, we define the following notations:*

- *atom*(B) *is the label of B;*
- *terms*$(B) =$ *terms*$($*atom*$(B))$ *is the set of terms of B;*
- *terms*(B) *is divided into two sets of terms, those* generated *in B, denoted by* generated(B)*, and those shared with its parent, denoted by* shared(B)*; precisely,* terms$(B) =$ shared$(B) \cup$ generated(B)*,* shared$(B) \cap$ generated$(B) = \emptyset$*, and if B is the root of $\mathcal{T}$, then*

[2] For instance, it can always be ensured by adding a position to all predicates, which is filled by the same fresh constant in the initial instance and by a new frontier variable in each rule. E.g. let c_0 be the new constant, then an instance atom of the form $p(a, b)$ would become $p(c_0, a, b)$ and a rule of the form $p(x, y) \to \exists z_1 \exists z_2\ q(z_1, z_2)$ would become $p(u, x, y) \to \exists z_1 \exists z_2\ q(u, z_1, z_2)$, where u is a variable. This translation does not change the behavior of any chase variant.

[3] The trees of bags that we consider are tree decompositions [26], hence the term "bag", which is classical in this setting.

generated(B) = *terms*(B) *(hence* *shared*$(B) = \emptyset$*), otherwise* B *has a parent* B_p *and* *generated*(B) = *terms*$(B) \setminus$ *terms*(B_p) *(hence,* *shared*(B) = *terms*$(B_p) \cap$ *terms*(B)*).*
Last, we denote by atoms$(\mathcal{T})$ *the set of atoms that label the bags in* $\mathcal{T}$*.*

Finally, we recall some classical mathematical notions. A *subsequence* S' of a sequence S is a sequence that can be obtained from S by deleting some (or no) elements without changing the order of the remaining elements. The *arity* of a tree is the maximal number of children for a node. A *prefix* T' of a tree T is a tree that can be obtained from T by repeatedly deleting some (or no) leaves of T.

3 Derivation Trees

A classical tool to reason about the chase is the so-called *chase graph* (see e.g., [6]), which is the directed graph consisting of all atoms that appear in the considered derivation, and with an arrow from a node n_1 to a node n_2 iff n_2 is created by a rule application on n_1 and possibly other atoms. [4] In the specific case of KBs of the form (I, Σ), where I is an atomic instance and Σ a set of linear rules, the chase graph is a tree. We recall below its definition in this specific case, in order to emphasize its differences with another tree, called *derivation tree*, on which we will actually rely.

▶ **Definition 6** (Chase Graph for Linear Rules)**.** *Let* $I_0 = \{\alpha\}$ *be an atomic instance,* Σ *be a set of linear rules, and* $S = I_0, (\sigma_1, \pi_1), I_1, \ldots, (\sigma_n, \pi_n), I_n$ *be a semi-oblivious* Σ*-derivation. The* chase graph *(also called* chase tree*) assigned to* S *is a tree of bags built as follows:*

- *the set of bags is in bijection with* I_n *via the labeling function* atom()*;*
- *the set of edges is in bijection with the set of triggers in* S *and is built as follows: for each trigger* (σ_i, π_i) *in* S*, there is an edge* (B, B') *with* $\text{atom}(B) = \pi_i(\text{body}(\sigma_i))$ *and* $\text{atom}(B') = \pi_i^s(\text{head}(\sigma_i))$*.*

▶ **Example 7.** Let $I_0 = q(a)$ and $\Sigma = \{\sigma_1, \sigma_2\}$ where $\sigma_1 = q(x) \to \exists y \exists z \exists t\ p(x, y, z, t)$ and $\sigma_2 = p(x, y, z, t) \to p(x, z, t, y)$. Let $S = I_0, (\sigma_1, \pi_1), I_1, (\sigma_2, \pi_2), I_3, (\sigma_2, \pi_3), I_3$ with $\pi_1 = \{x \mapsto a\}$, $\pi_1^s(\text{head}(\sigma_1)) = p(a, y_0, z_0, t_0)$, $\pi_2 = \{x \mapsto a, y \mapsto y_0, z \mapsto z_0, t \mapsto t_0\}$ and $\pi_3 = \{x \mapsto a, y \mapsto z_0, z \mapsto t_0, t \mapsto y_0\}$. The chase graph associated with S is a path of four nodes as pictured in Figure 1.

To check termination of a chase variant on a given KB (with an atomic instance), the general idea is to build a tree of bags associated with the chase on this KB in such a way that the occurrence of some forbidden pattern indicates that a path of unbounded length can be developed, hence the chase does not terminate. The forbidden pattern is composed of two distinct nodes such that one is an ancestor of the other and, intuitively speaking, these nodes "can be extended in similar ways", which leads to an arbitrarily long path that repeats the pattern.

Two atoms with the same type admit the same rule triggers, however, within a derivation, the same rule applications cannot necessarily be performed on both of them because of the presence of other atoms (this is already true for datalog rules, since the same atom is never produced twice). Hence, on the one hand we will specialize the notion of type, into that of a *sharing type*, and, on the other hand, adopt another tree structure, called a *derivation tree*, in which two nodes with the same sharing type have the required similar behavior.

[4] Note that the chase graph in [9] is a different notion.

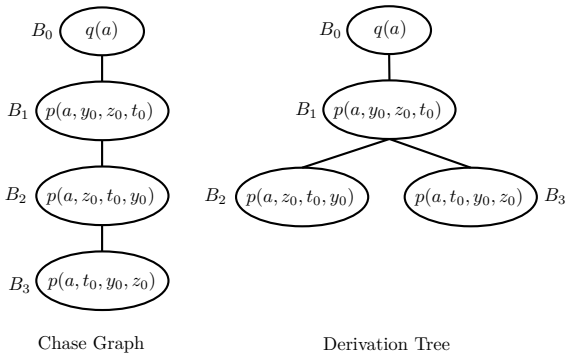


Figure 1 Chase Graph and Derivation Tree of Example 7.

▶ **Definition 8** (Sharing Type). *Given a tree of bags, the* sharing type *of a bag B is a pair $(type(atom(B)), P)$ where P is the set of positions in $atom(B)$ in which a term of $shared(B)$ occurs. We denote the fact that two bags B and B' have the same sharing type by $B \equiv_{st} B'$.*

We can now specify the forbidden pattern that we will consider: it is a pair of two distinct nodes with the same sharing type, such that one is an ancestor of the other.

▶ **Definition 9** (Unbounded-Path Witness). *An* unbounded-path witness *(UPW) in a derivation tree is a pair of distinct bags (B, B') such that B and B' have the same sharing type and B is an ancestor of B'.*

As explained below on Example 7, the chase graph is not the appropriate tool to use this forbidden pattern as a witness of chase non-termination.

▶ **Example 7** (continued). B_1, B_2 and B_3 in the chase graph of Figure 1 have the same classical type, $t = (p, \{\{1\}, \{2\}, \{3\}, \{4\}\})$. The sharing type of B_1 is $(t, \{1\})$, while B_2 and B_3 have the same sharing type $(t, \{1, 2, 3, 4\})$. B_2 and B_3 fulfill the condition of the forbidden pattern, however it is easily checked that any derivation that extends this derivation is finite.

Derivation trees were introduced as a tool to define the *greedy bounded treewidth set (gbts)* family of existential rules [2, 28]. A derivation tree is associated with a derivation, however it does not have the same structure as the chase graph. The fundamental reason is that, when a rule σ is applied to an atom α via a homomorphism π, the newly created bag is not necessarily attached in the tree as a child of the bag labeled by α. Instead, it is attached as a child of the *highest* bag in the tree labeled by an atom that contains $\pi(\text{fr}(\sigma))$, the image by π of the frontier of σ (note that $\pi(\text{fr}(\sigma))$ remains the set of terms shared between the new bag and its parent). This highest bag is unique. It is also the first created bag that contains $\pi(\text{fr}(\sigma))$.

In the following definition, a derivation tree is not associated with *any* derivation, but with a semi-oblivious derivation, which has the advantage of yielding trees with *bounded arity*. This is appropriate to study the termination of the semi-oblivious chase, and later the restricted chase, as a restricted derivation is a specific semi-oblivious derivation.

▶ **Definition 10** (Derivation Tree). *Let $I_0 = \{\alpha\}$ be an atomic instance, Σ be a set of linear rules, and $S = I_0, (\sigma_1, \pi_1), I_1, \ldots, (\sigma_n, \pi_n), I_n$ be a semi-oblivious Σ-derivation. The* derivation tree *assigned to S is a tree $\mathcal{T}$ of bags built as follows:*

- *the root of the tree, B_0, is such that $atom(B_0) = \alpha$;*

- *for each trigger* (σ_i, π_i), $0 < i \leq n$, *let* B_i *be the bag such that* $\text{atom}(B_i) = \pi_i^s(\text{head}(\sigma_i))$. *Let* j *be the smallest integer such that* $\pi_i(\text{fr}(\sigma_i)) \subseteq \text{terms}(B_j)$*: then* B_i *is added as a child to* B_j.

By extension, we say that a derivation tree $\mathcal{T}$ *is* associated with α and Σ *if there exists a semi-oblivious* Σ*-derivation* S *from* α *such that* $\mathcal{T}$ *is assigned to* S.

▶ **Example 7** (continued). The derivation tree associated with S is represented in Figure 1. Bags have the same sharing types in the chase tree and in the derivation tree. However, we can see here that they are not linked in the same way: B_3 was a child of B_2 in the chase tree, it becomes a child of B_1 in the derivation tree. Hence, the forbidden pattern cannot be found anymore in the tree.

Note that every non-root bag B shares at least one term with its parent (since the rule frontiers are not empty), furthermore this term is *generated* in its parent (otherwise B would have been added at a higher level in the tree).

4 Semi-Oblivious and Restricted Chase Termination

We now use derivation trees and sharing types to characterize the termination of the semi-oblivious chase. The fundamental property of derivation trees that we exploit is that, when two nodes have the same sharing type, the considered (semi-oblivious) derivation can always be extended so that these nodes have the same number of children, and in turn these children have the same sharing type. We first specify the notion of *bag copy*.

▶ **Definition 11** (Bag Copy). *Let* $\mathcal{T}$ *and* $\mathcal{T}'$ *be (possibly equal) trees of bags. Let* B *be a bag of* $\mathcal{T}$ *and* B' *be a bag of* $\mathcal{T}'$ *such that* $B \equiv_{st} B'$. *Let* B_c *be a child of* B. *A* copy *of* B_c under B' *is a bag* B'_c *such that* $\text{atom}(B'_c) = \varphi^s(\text{atom}(B_c))$, *where* φ^s *is a substitution of* $\text{terms}(B_c)$ *defined as follows:*

- *if* $t \in \text{shared}(B_c)$, *then* $\varphi^s(t) = \varphi_{\text{atom}(B)\to\text{atom}(B')}(t)$, *where* $\varphi_{\text{atom}(B)\to\text{atom}(B')}$ *is the natural mapping from* $\text{atom}(B)$ *to* $\text{atom}(B')$;
- *if* $t \in \text{generated}(B_c)$, *then* $\varphi^s(t)$ *is a fresh new variable.*

Let $\mathcal{T}_e$ be obtained from a derivation tree $\mathcal{T}$ by adding a copy of a bag: strictly speaking, $\mathcal{T}_e$ may not be a derivation tree in the sense that there may be no derivation to which it can be assigned, as illustrated by Example 12. Intuitively, some rule applications that would allow to produce the copy may be missing. However, there is some derivation tree of which $\mathcal{T}_e$ is a *prefix* (intuitively, one can add bags to $\mathcal{T}_e$ to obtain a derivation tree). That is why the following proposition considers more generally prefixes of derivation trees.

▶ **Example 12.** Let $\Sigma = \{\sigma_1, \sigma_2, \sigma_3\}$ with:

$$\sigma_1 : r(x,y) \to \exists z\ s(y,z) \qquad \sigma_2 : s(x,y) \to \exists z\ t(y,z) \qquad \sigma_3 : t(x,y) \to \exists z\ s(x,z)$$

Figure 2 pictures the result of copying a bag in a derivation tree $\mathcal{T}$ associated with a Σ-derivation: $\mathcal{T}$ has bags B_0, B_1, B_2 and B_3; then the bag B_3 is copied under itself yielding B'_3 (this copy is possible as B_1 and B_3 have the same sharing type and B_3 is a child of B_1). The obtained tree structure is not a derivation tree, as $s(z_3, z_4)$ cannot be generated by applying a rule on the instance associated with $\mathcal{T}$, however it could be completed to yield a derivation tree.

▶ **Proposition 13.** *Let* $\mathcal{T}$ *be a prefix of a derivation tree,* B *and* B' *be two bags of* $\mathcal{T}$ *such that* $B \equiv_{st} B'$, *and* B_c *be a child of* B. *Let* B'_c *be a copy of* B_c *under* B'. *Then: (a) it holds that* $B_c \equiv_{st} B'_c$, *and* (b) *the tree obtained from* $\mathcal{T}$ *by adding* B'_c *under* B', *if no copy of* B_c *already exists under* B', *is a prefix of a derivation tree.*

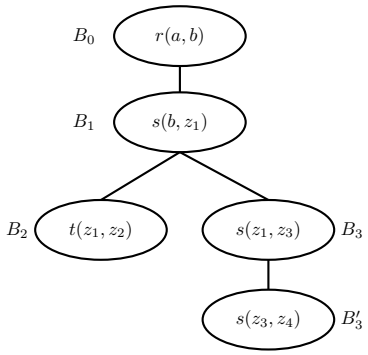


Figure 2 Copy Operation and Derivation Trees.

The size of a derivation tree without UPW is bounded, since its arity is bounded and its depth is bounded by the number of sharing types. It remains to show that a derivation tree that contains a UPW can be extended to an arbitrarily large derivation tree. We recall that a similar property would not hold for the chase tree, as witnessed by Example 7.

▶ **Proposition 14.** *There exists an arbitrary large derivation tree associated with α and Σ if and only if there exists a derivation tree associated with α and Σ that contains an unbounded-path witness.*

The previous proposition yields a characterization of the existence of an infinite semi-oblivious derivation. At this point, one may notice that an infinite semi-oblivious derivation is not necessarily fair. However, from this infinite derivation one can always build a fair derivation by inserting missing triggers. Obviously, this operation has no effect on the termination of the semi-oblivious chase. More precaution will be required for the restricted chase.

One obtains an algorithm to decide termination of the semi-oblivious chase for a given set of rules: for each canonical instance, build a semi-oblivious derivation and the associated derivation tree by applying rules until a UPW is created (in which case the answer is no) or all possible rule applications have been performed; if no instance has returned a negative answer, the answer is yes.

▶ **Corollary 15.** *The all-sequence termination problem for the semi-oblivious chase on linear rules is decidable.*

This algorithm can be modified to run in polynomial space (which is optimal [4]), by guessing a canonical instance and a UPW of its derivation tree.

▶ **Proposition 16.** *The all-sequence termination problem for the semi-oblivious chase on linear rules is in* PSpace.

We now consider the restricted chase. To this aim, we call *restricted derivation tree* associated with α and Σ a derivation tree associated with a restricted Σ-derivation from α.

We first point out that Proposition 13 is not true anymore for a restricted derivation tree, as the order in which rules are applied matters. This is illustrated by the next example.

▶ **Example 17.** Consider a restricted tree that contains bags B and B' with the same sharing type, labeled by atoms $q(t, u)$ and $q(v, w)$ respectively, where the second term is generated. Consider the following rules (the same as in Example 2):

$\sigma_1 : q(x, y) \to \exists z\ q(y, z) \qquad \sigma_2 : q(x, y) \to q(y, y)$

Assume B has a child B_c labeled by $q(u, z_0)$ obtained by an application of σ_1, and B' has a child B'_1 labeled by $q(w, w)$ obtained by an application of σ_2. It is not possible to extend this tree by copying B_c under B'. Indeed, the corresponding application of σ_1 does not comply with the restricted chase criterion: it would produce an atom of the form $q(w, z_1)$ that folds onto $q(w, w)$.

We thus prove a weaker proposition by considering that B' is a leaf in the restricted derivation tree.

▶ **Proposition 18.** *Let $\mathcal{T}$ be a prefix of a restricted derivation tree, B and B' be two bags of $\mathcal{T}$ such that $B \equiv_{st} B'$ and B'* is a leaf. *Let B_c be a child of B and B'_c be a copy of B_c under B'. Then:* (a) *$B_c \equiv_{st} B'_c$ and* (b) *the tree obtained from $\mathcal{T}$ by adding the copy B'_c of B_c under B' is a prefix of a restricted derivation tree.*

The previous proposition allows us to obtain a variant of Proposition 14 adapted to the restricted chase: [5]

▶ **Proposition 19.** *There exists an arbitrary large restricted derivation tree associated with α and Σ if and only if there exists a restricted derivation tree associated with α and Σ that contains an unbounded-path witness.*

It is less obvious than in the case of the semi-oblivious chase that the existence of an infinite derivation entails the existence of an infinite *fair* derivation. However, this property still holds:

▶ **Proposition 20.** *For linear rules, every (infinite) non-terminating restricted derivation is a subsequence of a fair restricted derivation (i.e., a restricted chase sequence).*

We point out that the previous property is not true anymore if linear rules are extended to non-atomic heads, as illustrated by the next example (in which only binary atoms are used).

▶ **Example 21.** Let $\Sigma = \{\sigma_1 : r(x, y) \to r(x, x);\ \sigma_2 : r(x, y) \to s(x, x);\ \sigma_3 : r(x, y) \to \exists z\ r(x, z) \wedge s(z, x) \wedge s(y, z)\}$. Starting from the instance $r(a, b)$, one builds an infinite restricted derivation that repeatedly applies σ_3, ignoring the two other rules. In order to turn this derivation into a fair derivation, one should at some point perform the applications of σ_1 and σ_2 to the initial instance, which produce the atoms $r(a, a)$ and $s(a, a)$. However, as soon as these atoms are produced, all triggers involving σ_3 are deactivated. Actually, there is no infinite fair restricted Σ-derivation from $r(a, b)$.

Similarly to Proposition 14 for the semi-oblivious chase, Proposition 19 provides an algorithm to decide termination of the restricted chase. The difference is that it is not sufficient to build a single derivation for a given canonical instance; instead, all possible restricted derivations from this instance have to be built (note that the associated restricted derivation trees are finite for the same reasons as before, and there is obviously a finite number of them). Hence, we obtain:

▶ **Corollary 22.** *The all-sequence termination problem for the restricted chase on linear rules is decidable.*

[5] Actually, a weak version of Proposition 13 where B' is a leaf would be sufficient to prove Proposition 14. However, the interest of Proposition 13 is to pinpoint an important difference between the semi-oblivious and restricted chase behaviors.

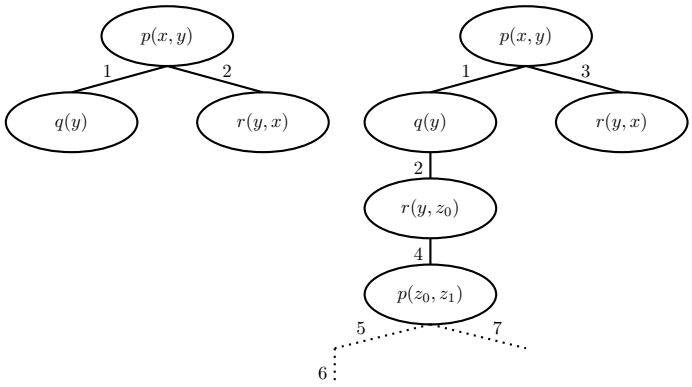


Figure 3 Finite versus Infinite Derivation Tree for Example 25.

A rough analysis of the proposed algorithm provides a co-N2ExpTime upper-bound for the complexity of the problem, by guessing a derivation that is of length at most double exponential, and checking whether there is a UPW in the corresponding derivation tree.

We now prove the decidability of the one-sequence termination problem. Note that a restricted derivation tree $\mathcal{T}$ that contains a UPW is a witness of the existence of an infinite restricted chase sequence, but does not prove that *every* restricted chase sequence that extends $\mathcal{T}$ is infinite. However, we prove that, if *all* restricted derivation trees contain a UPW, then there is no terminating restricted chase sequence.

▶ **Proposition 23.** *There exists a finite fair restricted derivation (i.e., a terminating restricted chase sequence) associated with α and Σ if and only if there exists one whose associated derivation tree does not contain any UPW.*

Proof sketch. We show that the derivation tree of the shortest finite fair restricted derivation cannot contain a UPW. This is done by contradiction: we assume there is a UPW in the associated derivation tree of the shortest finite fair restricted derivation, and we choose a UPW (B, B'), such that B' is a deepest bag among all UPWs. We create a new derivation that is equal to the original one up to the creation of B, then (1) copy the subtree rooted in B' under B and (2) perform "similarly" the missing rule applications so as to restore a fair restricted derivation. We show that during step (2) at least one trigger of the original derivation becomes inactive, which yields a strictly shorter fair restricted derivation. ◀

An algorithm to decide the existence of a finite restricted chase sequence for any instance is as follows: for any canonical instance, build all restricted derivations until either *(i)* there is a UPW in their associated derivation tree or *(ii)* there is no active trigger. Output Yes if and only if for all instances, there is a least one restricted derivation that halts because of the second condition. Such a derivation is of size at most double exponential, and we obtain a N2ExpTime upper bound for the complexity of the one-sequence termination problem.

▶ **Corollary 24.** *The one-sequence termination problem for the restricted chase on linear rules is decidable.*

Importantly, the previous algorithms are naturally able to consider solely some type of restricted derivations, i.e., build only derivation trees associated with such derivations, which is of theoretical but also of practical interest. Indeed, implementations of the restricted chase often proceed by building *breadth-first* derivations (which are intrinsically fair when they

are infinite), or variants of these. As witnessed by the next example, the termination of all breadth-first fair derivations is a strictly weaker requirement than the termination of all fair sequences, in the sense that the restricted chase terminates on more sets of rules.

▶ **Example 25.** Consider the following set of rules:

$\sigma_1 = p(x,y) \to q(y)$ $\quad$ $\sigma_2 = p(x,y) \to r(y,x)$
$\sigma_3 = q(y) \to \exists z\ r(y,z)$ $\quad$ $\sigma_4 = r(x,y) \to \exists z\ p(y,z)$

All breadth-first (fair) restricted derivations terminate, whatever the initial instance is. Remark that every application of σ_1 is followed by an application of σ_2 in the same breadth-first step, which prevents the application of σ_3. However, there is a fair restricted derivation that does not terminate (and this is even true for any instance). Indeed, an application of σ_2 can always be delayed, so that it comes too late to prevent the application of σ_3. See Figure 3: on the left, a finite derivation tree associated with a breadth-first derivation from instance $p(x,y)$; on the right, an infinite derivation tree associated with a (non-breadth-first) fair infinite derivation from the same instance. The numbers on edges give the order in which bags are created.

We conclude this section by noting that the previous Example 25 may give the (wrong) intuition that, given a set of rules, it is sufficient to consider breadth-first fair derivations to decide if there exists a terminating sequence. The following example shows that it is not the case: here, no breadth-first chase sequence is terminating, while there exists a terminating chase sequence for the given instance.

▶ **Example 26.** Let $\Sigma = \{\sigma_1, \sigma_2, \sigma_3\}$ with $\sigma_1 = p(x,y) \to \exists z\ p(y,z)$, $\sigma_2 = p(x,y) \to h(y)$, and $\sigma_3 = h(x) \to\ p(x,x)$. In this case, for every instance, there is a terminating restricted chase sequence, where the application of σ_2 and σ_3 prevents the indefinite application of σ_1. However, starting from $I = \{p(a,b)\}$, by applying rules in a breadth-first fashion one obtains a non-terminating restricted chase sequence, since σ_1 and σ_2 are always applied in parallel from the same atom, before applying σ_3.

As for the all-sequence termination problem, the algorithm may restrict the derivations of interest to specific kinds.

5 Core Chase Termination

We now consider the termination of the core chase on linear rules. Keeping the same approach, we prove that the finiteness of the core chase is equivalent to the existence of a finite tree of bags whose set of atoms is a minimal universal model. We call this a *(finite) complete core*. To bound the size of a complete core, we show that it cannot contain an unbounded-path witness. However, we cannot rely on derivation trees: indeed, there are linear sets of rules for which no derivation tree forms a complete core, as shown in Example 27. We will thus introduce a more general tree structure, namely *entailment trees*.

▶ **Example 27.** Let us consider the following rules:

$s(x) \to \exists y \exists z\ p(y,z,x)$ $\quad$ $p(y,z,x) \to \exists v\ q(y,v,x)$ $\quad$ $q(y,v,x) \to p(y,v,x)$

Let $I = \{s(a)\}$. The first rule applications yield a derivation tree $\mathcal{T}$ which is a path of bags B_0, B_1, B_2, B_3 respectively labeled by the following atoms:

$s(a)$, $p(y_0,z_0,a), q(y_0,v_0,a)$ and $p(y_0,v_0,a)$

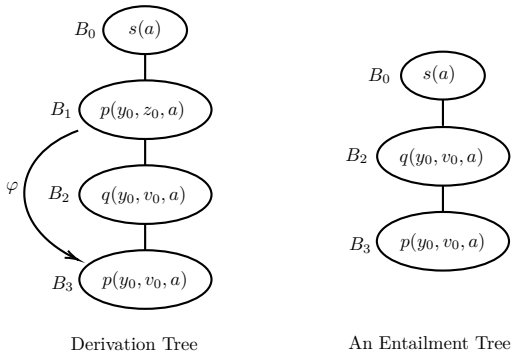


Figure 4 Derivation tree and entailment tree for Example 27.

$\mathcal{T}$ is represented on the left of Figure 4. Let A be this set of atoms. First, note that A is not a core: indeed it is equivalent to its strict subset A' defined by $\{B_0, B_2, B_3\}$ with a homomorphism π that maps $atom(B_1)$ to $atom(B_3)$. Trivially, A' is a core since it does not contain two atoms with the same predicate. Second, note that any further rule application on $\mathcal{T}$ is redundant, i.e., generates a set of atoms equivalent to A (and A'). Hence, A' is a complete core. However, there is no derivation tree that corresponds to A'. There is even no *prefix* of a derivation tree that corresponds to it (which ruins the alternative idea of building a prefix of a derivation tree that would be associated with a complete core). In particular, note that $\{B_0, B_1, B_2\}$ is indeed a core, but it is not complete.

The following notion of *twins* will be used in the definition of entailment trees to ensure that they have a bounded arity.

▶ **Definition 28.** *Two bags B and B' of a tree of bags are* twins *if they have the same sharing type, the same parent B_p and if the natural mapping $\varphi_{atom(B)\to atom(B')}$ is the identity on the terms of $atom(B_p)$.*

In the following definition of entailment tree, we use the notation $\alpha_1 \to \alpha_2$, where α_i is an atom, to denote the rule $\forall X(\alpha_1 \to \exists Y\ \alpha_2)$ with $X = vars(\alpha_1)$ and $Y = vars(\alpha_2) \setminus X$.

▶ **Definition 29** (Entailment Tree)**.** *An* entailment tree *associated with α and Σ is a tree of bags $\mathcal{T}$ such that:*

1. *B_r, the root of $\mathcal{T}$, is such that $\Sigma \models \alpha \to atom(B_r)$ and $\Sigma \models atom(B_r) \to \alpha$;*
2. *For any bag B_c child of a node B, the following holds:* (i) *$terms(B_c) \cap generated(B) \neq \emptyset$* (ii) *The terms in $generated(B_c)$ are variables that do not occur outside the subtree of $\mathcal{T}$ rooted in B_c* (iii) *$\Sigma \models atom(B) \to atom(B_c)$.*
3. *There is no pair of twins.*

If α is a ground atom, then B_r is simply labeled by α. Otherwise, it may happen that α does not belong to the result of the core chase (e.g., $\alpha = p(x, y)$, with x and y variables, and $\Sigma = \{p(x, y) \to p(x, x)\}$), hence Point 1.

First note that an entailment tree is independent from any derivation (in particular, fairness is not an issue). The main difference with a derivation tree is that it employs a more general parent-child relationship, that relies on entailment rather than on rule application, hence the name entailment tree. Intuitively, with respect to a derivation tree, one is allowed to move a bag B higher in the tree, provided that it contains at least one term generated in its new parent B_p; then, the terms of B that are not shared with B_p are freshly renamed. Finally,

since the problem of whether an atom is entailed by a linear existential rule knowledge base is decidable (precisely PSPACE-complete [5], even in the case of atomic instances), one can actually generate all non-twin children of a bag. As a bag can only have a bounded number of non-twin children, we are guaranteed to keep a tree of bounded arity.

Derivation trees are entailment trees, but not necessarily conversely. A crucial distinction between these two structures is the following statement, which does not hold for derivation trees, as illustrated by Example 27.

▶ **Proposition 30.** *If the core chase associated with α and Σ is finite, then there exists an entailment tree $\mathcal{T}$ such that the set of atoms associated with $\mathcal{T}$ is a complete core.*

▶ **Example 27** (continued)**.** The tree defined by the path of bags B_0, B_2, B_3 is an entailment tree, represented on the right of Figure 4, which defines a complete core.

Differently from the semi-oblivious case, we cannot conclude that the chase does not terminate as soon as a UPW is built, because the associated atoms may later be mapped to other atoms, which would remove the UPW. Instead, starting from the initial bag, we recursively add bags that do not generate a UPW (for instance, we can recursively add all such non-twin children to a leaf). Once the process terminates (the non-twin condition and the absence of UPW ensure that it does), we check that the obtained set of atoms C is complete (i.e., is a model of the KB): for that, it suffices to perform each possible rule application on C and check if the resulting set of atoms is equivalent to C (i.e., maps by homomorphism to C). See Algorithm 1. The set C may not be a core, but it is complete iff it contains a complete core.

We now focus on the key properties of entailment trees associated with complete cores. We first introduce the notion of *redundant bags*, which captures some cases of bags that cannot appear in a finite core. As witnessed by Example 27, this is not a characterization: B_1 is not redundant (according to Definition 31 below), but cannot belong to a complete core.

▶ **Definition 31** (Redundancy)**.** *Given an entailment tree, a bag B_c child of B is* redundant *if there exists an atom β (that may not belong to the tree) with* (i) $\Sigma \models atom(B) \to \beta$*;* (ii) *there is a homomorphism from $atom(B_c)$ to β that is the identity on $shared(B_c)$;* (iii) $|terms(\beta) \setminus terms(B)| < |terms(B_c) \setminus terms(B)|$.

Note that B_c may be redundant even if the "cause" for redundancy, i.e., β, is not in the tree yet. The role of this notion in the proofs is as follows: we show that if a complete entailment tree contains a UPW then it contains a redundant bag, and that a complete core cannot contain a redundant bag, hence a UPW. To prove this, we rely on Proposition 32 below, which is the counterpart for entailment trees of Proposition 13: performing a bag copy from an entailment tree results in an entailment tree (the notion of prefix is not needed, since a prefix of an entailment tree is an entailment tree) and keeps the properties of the copied bag.

▶ **Proposition 32.** *Let B be a bag of an entailment tree $\mathcal{T}$ and B' be a bag of an entailment tree $\mathcal{T}'$ such that $B \equiv_{st} B'$. Let B_c be a child of B. Let B'_c be a copy of B_c under B'. Then: (a) it holds that $B_c \equiv_{st} B'_c$,* (b) *the tree obtained from $\mathcal{T}$ by adding B'_c under B', if no copy of B_c already exists under B', is an entailment tree, and* (c) *B'_c is redundant if and only if B_c is redundant.*

In light of this, the copy of a bag can be naturally extended to the copy of the whole subtree rooted in a bag, which is a crucial element in the proof of Proposition 33 below.

▶ **Proposition 33.** *A complete core contains neither* (i) *a redundant bag, nor* (ii) *an unbounded-path witness.*

From Propositions 30 and 33, we conclude that Algorithm 1 decides the all-sequence termination problem for the core chase.

▶ **Corollary 34.** *The all-sequence termination problem for the core chase on linear rules is decidable.*

Algorithm 1: Deciding core chase termination.

Input : A set of linear rules
Output : `true` if and only if the core chase terminates on all instances

```
for each canonical atom α do
    Let T be the entailment tree restricted to a single root bag labeled by α;
    while a bag B can be added to T without creating twins nor a UPW do
        add B to T
    for (σ, π) such that σ is applicable to atoms(T) by π do
        if atoms(T) ⊭ atoms(T) ∪ π^s(head(σ));          // homomorphism check
        then
            return false
return true
```

A rough complexity analysis of this algorithm yields a 2ExpTime upper bound for the termination problem. Indeed, the exponential number of (sharing) types yields a bound on the number of canonical instances to be checked, the arity of the tree, as well as the length of a path without UPW in the tree, and each edge can be generated with a call to a PSpace oracle.

Finally, note that for predicates of arity at most two, it would still be possible to rely on derivation trees (instead of entailment trees), provided that the initial instance is ground. Indeed, in this specific case, the core of a (finite) derivation tree is a prefix of this tree (note that Example 27 uses predicates of arity three). Then, we can incrementally build a prefix of derivation tree until no bag can be added without creating a UPW, and, as in the higher arity case, check if the built tree is complete.

6 Concluding Remarks

We have shown the decidability of all-instance chase termination over atomic-head linear rules for three main chase variants (semi-oblivious, restricted, core) following a novel approach based on derivation trees, and their generalization to entailment trees, and a single notion of forbidden pattern. As far as we know, these are the first decidability results for the restricted chase, on both versions of the termination problem (i.e., *all-sequence* and *one-sequence* termination). The simplicity of the structures and algorithms makes it realistic to implement them.

We leave for future work the study of the precise complexity of the termination problems. A straightforward analysis of the complexity of the algorithms that decide the termination of the restricted and core chases yields upper bounds, however we believe that a finer analysis of the properties of sharing types would provide tighter upper bounds. It is an open question whether our results can be extended to more complex classes of existential rules, i.e., linear rules with a complex head, which is relevant for the termination of the restricted and core

chases, and more expressive classes from the guarded family. Concerning the extension to complex-head rules, the difficulty is that an infinite restricted derivation may not be transformable into a fair restricted derivation. Concerning the extension to more expressive classes, derivation trees were precisely defined to represent derivations with guarded rules and their extensions (i.e., greedy bounded treewidth sets), hence they seem to be a promising tool to study chase termination on that family, at least for the one-instance version of the problem, i.e., given a knowledge base. To consider the all-instance termination problem, a preliminary issue would be whether a finite set of canonical instances can be defined.

References

1 Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. On rules with existential variables: Walking the decidability line. *Artif. Intell.*, 175(9-10):1620–1654, 2011. doi:10.1016/j.artint.2011.03.002.

2 Jean-François Baget, Marie-Laure Mugnier, Sebastian Rudolph, and Michaël Thomazo. Walking the Complexity Lines for Generalized Guarded Existential Rules. In Toby Walsh, editor, *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 712–717. IJCAI/AAAI, 2011. doi:10.5591/978-1-57735-516-8/IJCAI11-126.

3 Catriel Beeri and Moshe Y. Vardi. The Implication Problem for Data Dependencies. In Shimon Even and Oded Kariv, editors, *Automata, Languages and Programming, 8th Colloquium, Acre (Akko), Israel, July 13-17, 1981, Proceedings*, volume 115 of *Lecture Notes in Computer Science*, pages 73–85. Springer, 1981. doi:10.1007/3-540-10843-2_7.

4 Marco Calautti, Georg Gottlob, and Andreas Pieris. Chase Termination for Guarded Existential Rules. In Tova Milo and Diego Calvanese, editors, *Proceedings of the 34th ACM Symposium on Principles of Database Systems, PODS 2015, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 91–103. ACM, 2015. doi:10.1145/2745754.2745773.

5 Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. Datalog Extensions for Tractable Query Answering over Ontologies. In Roberto De Virgilio, Fausto Giunchiglia, and Letizia Tanca, editors, *Semantic Web Information Management - A Model-Based Perspective*, pages 249–279. Springer, 2009. doi:10.1007/978-3-642-04329-1_12.

6 Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general Datalog-based framework for tractable query answering over ontologies. *J. Web Sem.*, 14:57–83, 2012. doi:10.1016/j.websem.2012.03.001.

7 Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable Reasoning and Efficient Query Answering in Description Logics: The *DL-Lite* Family. *J. Autom. Reasoning*, 39(3):385–429, 2007. doi:10.1007/s10817-007-9078-x.

8 David Carral, Irina Dragoste, and Markus Krötzsch. Detecting Chase (Non)Termination for Existential Rules with Disjunctions. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 922–928. ijcai.org, 2017. doi:10.24963/ijcai.2017/128.

9 Alin Deutsch, Alan Nash, and Jeffrey B. Remmel. The chase revisited. In Maurizio Lenzerini and Domenico Lembo, editors, *Proceedings of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2008, June 9-11, 2008, Vancouver, BC, Canada*, pages 149–158. ACM, 2008. doi:10.1145/1376916.1376938.

10 Ronald Fagin. A Normal Form for Relational Databases That Is Based on Domians and Keys. *ACM Trans. Database Syst.*, 6(3):387–415, 1981. doi:10.1145/319587.319592.

11 Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005. doi:10.1016/j.tcs.2004.10.033.

12 Tomasz Gogacz and Jerzy Marcinkowski. All-Instances Termination of Chase is Undecidable. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 293–304. Springer, 2014. doi:10.1007/978-3-662-43951-7_25.

13 Georg Gottlob, Giorgio Orsi, and Andreas Pieris. Query Rewriting and Optimization for Ontological Databases. *ACM Trans. Database Syst.*, 39(3):25:1–25:46, 2014. doi:10.1145/2638546.

14 Gösta Grahne and Adrian Onet. Anatomy of the Chase. *Fundam. Inform.*, 157(3):221–270, 2018. doi:10.3233/FI-2018-1627.

15 Bernardo Cuenca Grau, Ian Horrocks, Markus Krötzsch, Clemens Kupke, Despoina Magka, Boris Motik, and Zhe Wang. Acyclicity Notions for Existential Rules and Their Application to Query Answering in Ontologies. *J. Artif. Intell. Res.*, 47:741–808, 2013. doi:10.1613/jair.3949.

16 Alon Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, 2001. doi:10.1007/s007780100054.

17 André Hernich. Computing universal models under guarded TGDs. In Alin Deutsch, editor, *15th International Conference on Database Theory, ICDT '12, Berlin, Germany, March 26-29, 2012*, pages 222–235. ACM, 2012. doi:10.1145/2274576.2274600.

18 Mélanie König, Michel Leclère, Marie-Laure Mugnier, and Michaël Thomazo. Sound, complete and minimal UCQ-rewriting for existential rules. *Semantic Web*, 6(5):451–475, 2015. doi:10.3233/SW-140153.

19 Michel Leclère, Marie-Laure Mugnier, Michaël Thomazo, and Federico Ulliana. A Single Approach to Decide Chase Termination on Linear Existential Rules. *CoRR*, abs/1602.05828, 2018. arXiv:1810.02132.

20 Michel Leclère, Marie-Laure Mugnier, Michaël Thomazo, and Federico Ulliana. A Single Approach to Decide Chase Termination on Linear Existential Rules. In Magdalena Ortiz and Thomas Schneider, editors, *Proceedings of the 31st International Workshop on Description Logics co-located with 16th International Conference on Principles of Knowledge Representation and Reasoning (KR 2018), Tempe, Arizona, US, October 27th - to - 29th, 2018.*, volume 2211 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2018. URL: http://ceur-ws.org/Vol-2211/paper-45.pdf.

21 Maurizio Lenzerini. Data Integration: A Theoretical Perspective. In Lucian Popa, Serge Abiteboul, and Phokion G. Kolaitis, editors, *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, pages 233–246. ACM, 2002. doi:10.1145/543613.543644.

22 Bruno Marnette. Generalized schema-mappings: from termination to tractability. In Jan Paredaens and Jianwen Su, editors, *Proceedings of the Twenty-Eigth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2009, June 19 - July 1, 2009, Providence, Rhode Island, USA*, pages 13–22. ACM, 2009. doi:10.1145/1559795.1559799.

23 Marie-Laure Mugnier and Michaël Thomazo. An Introduction to Ontology-Based Query Answering with Existential Rules. In *Reasoning Web. Reasoning on the Web in the Big Data Era - 10th International Summer School 2014, Athens, Greece, September 8-13, 2014. Proceedings*, pages 245–278, 2014. doi:10.1007/978-3-319-10587-1_6.

24 Dan Olteanu, Jiewen Huang, and Christoph Koch. SPROUT: lazy vs. eager query plans for tuple-independent probabilistic databases. In Yannis E. Ioannidis, Dik Lun Lee, and Raymond T. Ng, editors, *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*, pages 640–651. IEEE Computer Society, 2009. doi:10.1109/ICDE.2009.123.

25 Adrian Onet. *The chase procedure and its applications.* PhD thesis, Concordia University, Canada, 2012. URL: https://pdfs.semanticscholar.org/6b1b/327a989d3d8e2488f645488063f391391b89.pdf.

26 Neil Robertson and Paul D. Seymour. Graph Minors. II. Algorithmic Aspects of Tree-Width. *J. Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.

27 Swan Rocher. *Querying Existential Rule Knowledge Bases: Decidability and Complexity. (Interrogation de Bases de Connaissances avec Règles Existentielles : Décidabilité et Complexité)*. PhD thesis, University of Montpellier, France, 2016. URL: https://tel.archives-ouvertes.fr/tel-01483770.

28 Michaël Thomazo. *Conjunctive Query Answering Under Existential Rules - Decidability, Complexity, and Algorithms*. PhD thesis, Montpellier 2 University, France, 2013. URL: https://tel.archives-ouvertes.fr/tel-00925722.

Additive First-Order Queries

Gerald Berger
TU Wien, Austria

Martin Otto
TU Darmstadt, Germany

Andreas Pieris
University of Edinburgh, Scotland

Dimitri Surinx
Hasselt University, Belgium

Jan Van den Bussche
Hasselt University, Belgium

Abstract

A database query q is called additive if $q(A \cup B) = q(A) \cup q(B)$ for domain-disjoint input databases A and B. Additivity allows the computation of the query result to be parallelised over the connected components of the input database. We define the "connected formulas" as a syntactic fragment of first-order logic, and show that a first-order query is additive if and only if it expressible by a connected formula. This characterisation specializes to the guarded fragment of first-order logic. We also show that additivity is decidable for formulas of the guarded fragment, establish the computational complexity, and do the same for positive-existential formulas. Our results hold when restricting attention to finite structures, as is common in database theory, but also hold in the unrestricted setting.

2012 ACM Subject Classification Information systems → Query languages

Keywords and phrases Expressive power

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.19

1 Introduction

Motivated by cloud computing and big data, in the past few years, there has been interest in logical characterisations of queries that can be answered using parallelism [11]. An attractive class of queries that was identified is formed by those that "distribute over components" [3]. These queries can be faithfully answered by the following three-step strategy: first, partition the input database in any way that does not split connected components; second, evaluate the query separately on each part, thus allowing some degree of parallelism; third, collect the final result by taking the union of all partial results. Equivalently, over finite databases, a query q for which this works correctly is *additive*, meaning that $q(A \cup B) = q(A) \cup q(B)$ for any two domain-disjoint databases A and B. An added bonus is that the two partial results $Q(A)$ and $Q(B)$ are disjoint. Apart from the relevance to distributed computing, additivity is also a useful notion in the analysis of expressive power of query languages [20, 21].

Additivity is a semantic property that is undecidable for queries given, say, as Datalog programs, or as first-order logic formulas. It is therefore desirable to match additivity to a syntactic restriction in the query language. This was done successfully in the setting of Datalog programs, where it is a natural idea to restrict rule bodies so that they must be connected. It is then relatively straightforward to show that a Datalog query is additive if and only if it can be computed by a Datalog program with connected rule bodies. This program was successfully carried out for a range of Datalog variants, such as Datalog extended

with stratified or well-founded negation, or value invention [3, 4], as well as for unions of conjunctive queries, and ontology-mediated querying with conjunctive queries over linear, guarded, or sticky tuple-generating dependencies [6].

The case of first-order queries (equivalently, relational algebra queries), however, has remained open until now. In this paper we define a syntactical notion of connectedness for first-order logic formulas, as well as for relational algebra expressions. In a connected relational algebra expression, equijoins are allowed, but cartesian products are not. Connected first-order logic formulas are defined similarly. Queries expressible by connected formulas are always additive. We show that the converse holds as well: if a first-order formula φ expresses an additive query, then φ is equivalent to a connected formula. A similar result then follows for relational algebra expressions.

Results of this kind are colloquially known as expressive completeness results, characterisation theorems, or preservation theorems [2, 18]. For example, modal characterisation theorems link bisimulation-invariant first-order formulas (in one free variable) to formulas in modal logics [23, 17, 14, 15, 16]. Indeed, the proof of our result starts from ideas that were developed to prove modal characterisation theorems over finite structures as well as over all structures. As a consequence, also our result holds over finite structure as well as over all structures.

In a second part of the paper, we consider the guarded fragment (GF) of first-order logic [5]. GF was originally introduced as a first-order generalisation of modal logic that preserves good properties such as the tree model property, the finite model property, and a decidable satisfiability problem. Satisfiability for GF is 2ExpTime-complete in general and ExpTime-complete for bounded arity [9].

When restricting attention to queries returning guarded tuples, the guarded fragment corresponds to the semijoin algebra [12]. Such queries are always additive. We complement this picture and show that a guarded formula φ expresses an additive query if and only if φ is equivalent to a connected guarded formula. We will see that additivity is decidable for GF as well, by a reduction to satisfiability. While our reduction preserves bounded arity, it is unfortunately exponential, so we only obtain a 2ExpTime upper bound, even for bounded arity. By a very simple reduction from satisfiability, additivity for GF is 2ExpTime-complete in general and ExpTime-hard for bounded arity.

Finally, we will consider positive-existential (PE) first-order formulas. Such formulas have the same expressive power as unions of conjunctive queries (UCQ) [1]. In earlier work [6], additivity for UCQs was shown to be NP-complete. In this paper we complement this result and show that additivity for PE formulas is Π_2^P-complete. The pattern that seems to emerge here is that additivity has the same complexity as containment.

2 Preliminaries

From the outset, we assume a supply of *relation names*, where each relation name has an associated arity (a natural number). We use R/k to denote that the relation name R has arity k. In this paper we do not consider nullary relation names, as their presence corrupts the intuitive notion of domain-disjointness which will play an important role in this work (see Section 3).

We further assume an infinite domain **dom** of atomic data elements called *constants*. A *fact* is of the form $R(a_1, \ldots, a_k)$ where $a_1, \ldots, a_k$ are constants and R/k is a relation name. We call R the *predicate* of the fact.

A *database schema* (or schema for short) is a finite set of relation names. An *instance* of a schema $\mathbf{S}$ is a nonempty set of facts with predicates from $\mathbf{S}$. The set of all constants appearing in an instance $\mathcal{A}$ is called the *active domain* of $\mathcal{A}$ and denoted by $adom(\mathcal{A})$.

▶ Remark (Finite vs unrestricted instances). In database theory it is common and natural to restrict attention to finite instances. Our results will hold in restriction to finite instances, as well as in the unrestricted setting where we allow all instances.

Let $\mathbf{S}$ be a schema, and let k be a natural number. A *k-ary query over* $\mathbf{S}$ is a function that maps each instance $\mathcal{A}$ of $\mathbf{S}$ to a k-ary relation on $adom(\mathcal{A})$. Note that a nullary query ($k = 0$) has only two possible results, $\{()\}$ and $\emptyset$, which can be interpreted as the boolean values true and false respectively. Thus nullary queries capture the notion of boolean or yes/no query.

A standard language for expressing queries is the relational algebra [22]. To fix notation, we define it formally [13]. Note that we only consider equality selections and equijoins; the extension of our work to selection and join predicates involving constants, order, or arithmetic, is an interesting direction for further research.

▶ **Definition 1.** *The expressions of the relational algebra (RA) over a schema* $\mathbf{S}$ *are generated as follows:*

- *Each relation name* $R \in \mathbf{S}$ *is a relational algebra expression. Its arity comes from* $\mathbf{S}$.
- *If* $E_1, E_2 \in RA$ *have arity* n*, then also* $E_1 \cup E_2$ *(union) and* $E_1 - E_2$ *(difference) belong to RA and are of arity* n.
- *If* $E \in RA$ *has arity* n *and* $i_1, \ldots, i_k \in \{1, \ldots, n\}$*, then* $\pi_{i_1,\ldots,i_k}(E)$ *(projection) belongs to RA and is of arity* k.
- *If* $E \in RA$ *has arity* n *and* $i, j \in \{1, \ldots, n\}$*, then* $\sigma_{i=j}(E)$ *(selection) belongs to RA and is of arity* n.
- *If* $E_1, E_2 \in RA$ *have arities* n *and* m*, respectively, then* $E_1 \times E_2$ *(cartesian product) belongs to RA and is of arity* $n + m$.

The semantics is well known; we recall it for the sake of completeness. Let $\mathcal{A}$ *be an instance of* $\mathbf{S}$.

- *If* E *is a relation name* R *then* $E(\mathcal{A}) := \{\bar{a} \mid R(\bar{a}) \in \mathcal{A}\}$.
- *If* E *is* $E_1 \cup E_2$*, or* $E_1 - E_2$*, or* $E_1 \times E_2$*, then* $E(\mathcal{A}) := E_1(\mathcal{A}) \cup E_2(\mathcal{A})$*, or* $E_1(\mathcal{A}) - E_2(\mathcal{A})$*, or* $E_1(\mathcal{A}) \times E_2(\mathcal{A})$*, respectively.*
- *If* E *is* $\pi_{i_1,\ldots,i_k}(E_1)$ *then* $E(\mathcal{A}) := \{(a_{i_1}, \ldots, a_{i_k}) \mid \bar{a} \in E_1(\mathcal{A})\}$.
- *If* E *is* $\sigma_{i=j}(E_1)$ *then* $E(\mathcal{A}) := \{\bar{a} \in E_1(\mathcal{A}) \mid a_i = a_j\}$.

The relational algebra is equivalent in expressive power to the language of first-order logic (FO) [22, 1]. To match RA as formalised above, we use FO with equality but without constants. An FO formula φ with free variables $x_1, \ldots, x_k$ expresses the k-ary query that maps an instance $\mathcal{A}$ to the set of all k-tuples $(\alpha(x_1), \ldots, \alpha(x_k))$, where α is a valuation from $\{x_1, \ldots, x_k\}$ to $adom(\mathcal{A})$ such that $\mathcal{A}, \alpha \models \varphi$. Here, $\mathcal{A}$ is viewed as a structure with domain $adom(\mathcal{A})$, so we employ the active-domain semantics for first-order logic. We denote the resulting relation by $\varphi(\mathcal{A})$. In particular, sentences (formulas without free variables) express nullary queries.

Notation.

- To indicate an ordering of the free variables of φ, we will use the notation $\varphi(x_1, \ldots, x_k)$.
- We will use FO to denote the class of queries expressible in FO.

3 Additive queries

Let q be a query over schema $\mathbf{S}$. We call q *additive* if $q(\mathcal{A} \cup \mathcal{B}) = q(\mathcal{A}) \cup q(\mathcal{B})$ for any two $\mathbf{S}$-instances $\mathcal{A}$ and $\mathcal{B}$ that are *domain-disjoint*, meaning that $adom(\mathcal{A})$ is disjoint from $adom(\mathcal{B})$. Note that, formally, this definition applies equally well to boolean (nullary)

queries. Interpreting $\{()\}$ as true and $\emptyset$ as false, as we will always do, the condition $q(\mathcal{A} \cup \mathcal{B}) = q(\mathcal{A}) \cup q(\mathcal{B})$ can be equivalently read as "$q(\mathcal{A} \cup \mathcal{B})$ is true if and only if $q(\mathcal{A})$ is true or $q(\mathcal{B})$ is true".

Notation. When we write $\mathcal{A} = \mathcal{B} + \mathcal{C}$ we will mean that $\mathcal{A} = \mathcal{B} \cup \mathcal{C}$ and $\mathcal{B}$ and $\mathcal{C}$ are domain-disjoint.

▶ **Example 2.** Consider queries about two binary relations R and T, expressed in RA.

1. The selection $\sigma_{1=2}(R)$ is clearly additive.
2. The join $\pi_{1,2,4}\sigma_{2=3}(R \times T)$ is additive.
3. The union $R \cup T$ and the difference $R - T$ are additive.

In order to give examples of queries that are not additive, it is useful to introduce the following definitions and lemma.

▶ **Definition 3** ([3]). *A* strict component *of an instance $\mathcal{A}$ is an instance $\mathcal{C}$ such that $\mathcal{A} = \mathcal{C} + \mathcal{B}$ for some instance $\mathcal{B}$, and $\mathcal{C}$ is minimal with this property. (Recall that instances are nonempty.) We now say that an instance $\mathcal{C}$ is a* component *of an instance $\mathcal{A}$, if $\mathcal{C}$ is a strict component of $\mathcal{A}$, or if $\mathcal{C}$ equals $\mathcal{A}$ and $\mathcal{A}$ has no strict components.*

▶ **Definition 4.** *Let $\mathcal{A}$ be an instance and let t be a tuple of elements of $adom(\mathcal{A})$. We call t* mixed *with respect to $\mathcal{A}$, if t contains elements from (at least) two different components of $\mathcal{A}$.*

▶ **Lemma 5.** *Let q be an additive query and let $\mathcal{A}$ be an instance. Then $q(\mathcal{A})$ does not contain any mixed tuples (with respect to $\mathcal{A}$).*

Proof. Suppose $t \in q(\mathcal{A})$ contains elements from two distinct components $\mathcal{B}$ and $\mathcal{C}$ (and possibly more). Write $\mathcal{A} = \mathcal{B} + \mathcal{C} + \mathcal{D}$. Then $t \in q(\mathcal{B}) \cup q(\mathcal{C}) \cup q(\mathcal{D})$. However, t cannot be in $q(\mathcal{B})$ since $q(\mathcal{B})$ is a relation on $adom(\mathcal{B})$ and t contains elements from $adom(\mathcal{C})$. For similar reasons, t can neither be in $q(\mathcal{C})$ nor in $q(\mathcal{D})$. We have arrived at a contradiction. ◀

We can now follow up Example 2 with some negative examples.

▶ **Example 6.**

1. Let E be the cartesian product $R \times T$. Then E is not additive. Indeed, consider $\mathcal{A} = \{R(a, a), T(b, b)\}$. Then $(a, a, b, b) \in E(\mathcal{A})$ is a mixed tuple, contradicting Lemma 5.
2. Let $\varphi(x, y, z)$ be the FO query $R(x, y) \vee T(y, z)$. Then φ is not additive. Indeed, consider $\mathcal{A} = \{R(a, b), T(c, d)\}$. Then $(a, b, c) \in \varphi(\mathcal{A})$ is a mixed tuple, again contradicting Lemma 5.
3. Let φ be the boolean FO query $\neg\exists z\, R(z, z)$. Then φ is not additive. Indeed, consider $\mathcal{A} = \{R(a, b)\}$ and $\mathcal{B} = \{R(c, c)\}$. Then $\varphi(\mathcal{A})$ is true, but $\varphi(\mathcal{A} \cup \mathcal{B})$ is false. Note that φ does not return mixed tuples, yet is not additive.

3.1 Queries that distribute over components

A notion closely related to additivity is that of *distributing over components* [3]. A query q is said to distribute over components if $q(\mathcal{A})$ equals the union of $q(\mathcal{C})$ over all components $\mathcal{C}$ of $\mathcal{A}$.

Let us denote the class of additive queries by ADD and the class of queries distributing over components by DIST. Clearly, DIST implies ADD. Over instances that have only finitely many components, the converse implication is quite clear as well. Over infinite instances, the converse implication can still be shown quite simply, but only for non-boolean queries.

We summarise the situation as follows. Let ADD* be the class of additive queries that are not nullary, and similarly for DIST*.

▶ **Proposition 7.** *In restriction to finite instances,* ADD = DIST. *In the unrestricted setting,* $\text{ADD}^* = \text{DIST}^*$.

Proof. The only part that is not immediately clear is that ADD implies DIST for non-boolean queries in the unrestricted setting. To show this, let q be an additive non-boolean query, and let $\mathcal{A}$ be an instance. We need to verify that $q(\mathcal{A})$ equals the union of $q(\mathcal{C})$ over all components $\mathcal{C}$ of $\mathcal{A}$. For the inclusion from right to left, take a component $\mathcal{C}$ and write $\mathcal{A} = \mathcal{C} + \mathcal{B}$. Then $q(\mathcal{C}) \subseteq q(\mathcal{C}) \cup q(\mathcal{B}) = q(\mathcal{A})$.

For the inclusion from left to right, let $t \in q(\mathcal{A})$. By Lemma 5, t consists of elements from a single component $\mathcal{C}$ of $\mathcal{A}$. Moreover, since q is non-boolean, t is nonempty. Write $\mathcal{A} = \mathcal{C} + \mathcal{B}$. Thus $t \in q(\mathcal{C}) \cup q(\mathcal{B})$. However, t cannot be in $q(\mathcal{B})$ since $\mathcal{B}$ is domain-disjoint from $\mathcal{C}$. Hence $t \in q(\mathcal{C})$ as desired. ◀

Actually, over infinite instances, ADD does not imply DIST for boolean queries. Indeed, the boolean query "does the instance have infinitely many components?" is in ADD but not in DIST. Within FO, however, the equivalence between ADD and DIST does hold. This will follow from our result on additive boolean FO queries (Proposition 14).

4 The first-order case

In this section we introduce the connected formulas as a syntactical restriction on FO formulas. Our main result will be that a query expressible in FO is additive if and only if it is expressible by a connected FO formula. A similar result will then follow for the relational algebra. We will conclude the section with a very simple reduction from satisfiability to additivity.

4.1 Connected FO

Connected FO formulas are generated according to the following syntax rules:

- Every atomic formula is connected.
- If φ is connected then $\exists y\, \varphi$, for any variable y, is also connected.
- If φ and ψ are connected and they share at least one free variable, then $\varphi \wedge \psi$ is also connected.
- If φ and ψ are connected and have exactly the same free variables (possibly none), then $\varphi \vee \psi$ is also connected.
- If φ and ψ are connected, ψ has at least one free variable, and φ has all the free variables of ψ, then $\varphi \wedge \neg\psi$ is also connected.

▶ **Example 8.** Revisiting Examples 2 and 6, we see that the positive examples are expressible by connected formulas: the selection example by $R(x, y) \wedge x = y$; the equijoin by $R(x, y) \wedge T(y, z)$; the union and difference by $R(x, y) \vee T(x, y)$ and $R(x, y) \wedge \neg T(x, y)$, respectively.

We also see that the negative examples are not connected formulas: the formula $R(x, y) \wedge T(u, v)$ for cartesian product violates the conjunction rule; the formula $R(x, y) \vee T(y, z)$ violates the disjunction rule; and the formula $\neg\exists z\, R(z, z)$ violates the negation rule.

It is readily verified that connected FO queries are always additive. Our main result in this section is that the converse holds as well. Let CFO denote the queries expressible by a connected FO formula. We establish:

▶ **Theorem 9.** $\text{FO} \cap \text{ADD} = \text{CFO}$.

4.2 Non-boolean queries

Our theorem is proven separately for formulas with free variables, and for sentences. In this subsection we handle the case with free variables.

We need to recall the basic notions concerning the locality of first-order logic [8]. The *Gaifman graph* of an instance $\mathcal{A}$, denoted by $G(\mathcal{A})$, is the undirected graph with $adom(\mathcal{A})$ as the set of nodes; there is an edge between distinct nodes a and b if a and b occur together in some fact in $\mathcal{A}$. The distance between a and b in $G(\mathcal{A})$ is denoted by $d^{\mathcal{A}}(a,b)$, or simply $d(a,b)$ if $\mathcal{A}$ is understood. For any natural number ℓ, the set $\{b \in adom(\mathcal{A}) \mid d(a,b) \leq \ell\}$ is denoted by $B^{\mathcal{A}}(a,\ell)$ or $B(a,\ell)$ if $\mathcal{A}$ is understood (B for "ball"). The notation $B(\bar{a},\ell)$, for a tuple $\bar{a} = a_1,\ldots,a_k$, denotes the union of $B(a_i,\ell)$ for $i = 1,\ldots,k$. The restriction of $\mathcal{A}$ to $B(\bar{a},\ell)$ is denoted by $N^{\mathcal{A}}(\bar{a},\ell)$ (N for "neighbourhood"). A k-ary formula $\varphi(\bar{x})$ is called *ℓ-local* if for every instance $\mathcal{A}$ and every k-tuple $\bar{a}$ over $adom(\mathcal{A})$, we have

$$\bar{a} \in \varphi(\mathcal{A}) \quad \Leftrightarrow \quad \bar{a} \in \varphi(N^{\mathcal{A}}(\bar{a},\ell)).$$

We say that a formula is local if it is ℓ-local for some ℓ.

We are now ready to state an important lemma. In modal characterisation theorems, one considers unary FO formulas over unary and binary relations that are invariant under bisimulation [23]. Invariance under bisimulation implies invariance under disjoint sums [15]. Additivity is the natural generalisation of invariance under disjoint sums to higher-arity queries. In earlier work, one of us showed, by a detailed Ehrenfeucht-Fraïssé game argument, that unary FO formulas, invariant under disjoint sums, are always local [16, Lemma 3.5]. We have carefully verified that the exact same argument also works for formulas about higher-arity relations and with multiple free variables. We thus obtain:

▶ **Lemma 10.** *Let φ be an additive FO formula and let q be its quantifier rank. Then φ is 2^q-local.*

By the above lemma, all subformulas of $\varphi(\bar{x})$ may be assumed to talk about elements y that belong to $B(\bar{x},2^q)$. Moreover, by Lemma 5 we already know that the values of the free variables $\bar{x}$ must come from the same component. This is not quite enough, however, to express $y \in B(\bar{x},2^q)$ by a connected FO formula; for that we need a finite upper bound on the diameter of the tuple of (valuations of) free variables that holds uniformly over all instances. This is provided by the following:

▶ **Proposition 11.** *Let φ be an additive FO formula of quantifier rank q. Assume $(a_1,\ldots,a_n)$ is in $\varphi(\mathcal{A})$. Then $d^{\mathcal{A}}(a_i,a_j) \leq (n-1)2^q$ for all $i,j \in \{1,\ldots,n\}$.*

Proof. Let $\bar{a} = a_1,\ldots,a_n$ and let $\ell = 2^q$. Suppose to the contrary that $d(a_i,a_j) > (n-1)\ell$. Then there exists a partition $\{I,J\}$ of $\{1,\ldots,n\}$ such that $d(\bar{a}|_I,\bar{a}|_J) > \ell$. Here, $d(\bar{a}|_I,\bar{a}|_J)$ naturally stands for the minimum of $d(a_i,a_j)$ for $i \in I$ and $j \in J$.

Let us make a domain-disjoint copy $f(\mathcal{A})$ of $\mathcal{A}$ by using a bijection $f : adom(\mathcal{A}) \to B$ for some set B disjoint from $adom(\mathcal{A})$. Take q additional domain-disjoint copies of $\mathcal{A}$, which we denote by $q \cdot \mathcal{A}$. Define the tuple $\bar{b} = b_1,\ldots,b_n$ by

$$b_i = \begin{cases} a_i & \text{if } i \in I, \\ f(a_i) & \text{if } i \in J. \end{cases}$$

Consider the instance $\mathcal{C} = \mathcal{A} + f(\mathcal{A}) + q \cdot \mathcal{A}$. Since $\bar{a} \in \varphi(\mathcal{A})$ and φ is additive, also $\bar{a} \in \varphi(\mathcal{C})$. Using an Ehrenfeucht-Fraïssé game argument, inspired on the original proof of Lemma 10, we obtain that $\bar{b} \in \varphi(\mathcal{C})$. This contradicts the additivity of φ, since $\bar{b}$ is a mixed tuple with respect to $\mathcal{C}$ (Lemma 5). ◀

Our final lemma is the following:

▶ **Lemma 12.** *Let $\bar{x}$ be a nonempty list of variables. Let φ be a formula, with free variables among $\bar{x}$, so that each quantifier in φ is of the form $\exists y \in B(\bar{x}', \ell)$ with $y \notin \bar{x}$, and $\bar{x}'$ a nonempty subtuple of $\bar{x}$. Let δ be a connected formula with exactly $\bar{x}$ as free variables. Then $\varphi \wedge \delta$ can be rewritten as a connected formula.*

Proof. It is readily verified that predicates of the form $d(x, y) \leq m$ can be expressed by connected formulas. The lemma can be proven by a straightforward structural induction. ◀

▶ **Example 13.** Let us illustrate Lemma 12 with φ being $\exists y \in B(x_1, x_2, 3)\, \neg S(y)$, and δ being $R(x_1, x_2)$. The formula $\varphi \wedge \delta$ can be rewritten into the connected formula

$$\exists y\big((\neg S(y) \wedge d(x_1, y) \leq 3 \wedge R(x_1, x_2)) \vee (\neg S(y) \wedge d(x_2, y) \leq 3 \wedge R(x_1, x_2))\big).$$

We now have all ingredients to prove that every additive FO formula φ with at least one free variable can be rewritten as a connected formula. Let $\bar{x} = x_1, \ldots, x_n$ be the list of free variables of φ. By Lemma 10, we may assume that each quantifier in φ is of the form $\exists y \in B(\bar{x}, \ell)$. We may always assume that $y \notin \bar{x}$ simply by choosing another bound variable. Furthermore, by Proposition 11, φ is equivalent to $\varphi \wedge \delta$, where δ is the conjunction of $d(x_1, x_i) \leq (n-1)\ell$ for $i = 2, \ldots, n$. (If $n = 1$, set δ to $x_1 = x_1$.) Hence, by Lemma 12, φ is equivalent to a connected formula as desired.

4.3 Boolean queries

The argument in the previous subsection relies on the presence of at least one free variable to connect everything inside the formula. So, to prove Theorem 9 for sentences, we need a separate argument. Interestingly, this argument will then only work for sentences and not with free variables. However, in the next section, we will be able to adapt the argument at least to guarded formulas, with or without free variables.

Recall that a *simple local sentence* is a sentence of the form $\exists x\, \varphi$ where φ is local [15]. By Lemma 12, such sentences can be rewritten in connected form (use $x = x$ for δ). Since a disjunction of connected sentences is again connected, the following proposition proves all we need.

▶ **Proposition 14.** *Every additive FO sentence can be rewritten as a finite disjunction of simple local sentences.*

Proof. Let φ be an additive sentence. The formula φ^* that is $(x = x) \wedge \varphi$ is *invariant under disjoint copies*, by which we mean the following. For an instance $\mathcal{A}$, let $q \cdot \mathcal{A}$ denote an instance consisting of q domain-disjoint copies of $\mathcal{A}$. Then, for any $a \in adom(\mathcal{A})$ and any q, we have $a \in \varphi^*(\mathcal{A})$ iff $a \in \varphi^*(\mathcal{A} + q \cdot \mathcal{A})$. Over unary and binary relations, it was already proven that any formula in one free variable, invariant under disjoint copies, is equivalent to a boolean combination of simple local sentences and local formulas [15, Proposition 19]. That proof goes through verbatim for higher-arity relations as well. It follows that $\varphi \equiv \exists x\, \varphi^*$ is equivalent to a boolean combination of simple local sentences.

So we now assume that φ is a boolean combination, in disjunctive normal form, over a finite set Σ of simple local sentences. We may assume that each clause is *complete*, meaning that it either contains σ or $\neg\sigma$ for every $\sigma \in \Sigma$. We may also assume that each clause is satisfiable. We will identify φ with the set of its clauses.

The plan of the proof is to first get rid of negations, then of conjunctions, leaving a disjunction as desired. For the first step, let Γ denote the set of all satisfiable complete clauses over Σ. For two clauses γ_1 and γ_2 in Γ, we write $\gamma_1 \leq \gamma_2$ if every $\sigma \in \Sigma$ that occurs positively in γ_1 also occurs positively in γ_2. We claim the following *monotonicity property:*

Let $\gamma_1 \in \varphi$ and let $\gamma_2 \in \Gamma$ such that $\gamma_1 \leq \gamma_2$. Then also $\gamma_2 \in \varphi$.

In proof, let $\mathcal{A}$ and $\mathcal{B}$ be domain-disjoint instances such that $\mathcal{A} \models \gamma_1$ and $\mathcal{B} \models \gamma_2$. Since $\mathcal{A} \models \varphi$, by additivity also $\mathcal{A} + \mathcal{B} \models \varphi$, so $\mathcal{A} + \mathcal{B} \models \gamma$ for some $\gamma \in \varphi$. We verify that $\gamma = \gamma_2$. Consider any $\sigma \in \Sigma$.

- If σ occurs positively in γ_2, then $\mathcal{B} \models \sigma$, so also $\mathcal{A} + \mathcal{B} \models \sigma$. Hence, σ cannot occur negated in γ, i.e., σ occurs positively also in γ.
- If σ occurs negated in γ_2, then also in γ_1, so neither $\mathcal{A} \models \sigma$ nor $\mathcal{B} \models \sigma$ holds, and thus $\mathcal{A} + \mathcal{B} \models \neg\sigma$. Hence, σ cannot occur positively in γ, i.e., σ occurs negated also in γ.

By the monotonicity property, we can simplify φ so that it is now a disjunction of conjunctions, each conjunction over some finite subset of Σ. We will naturally view φ as a set of subsets of Σ. We may also assume that φ is simplified further so that every $\gamma \in \varphi$ is *minimal*, meaning that replacing γ by a strict subset $\gamma' \subsetneq \gamma$ would yield a sentence not equivalent to φ.

Now assume, for the sake of contradiction, that some $\gamma \in \varphi$ has at least two elements. Then we can split $\gamma = \gamma_1 \cup \gamma_2$ in two disjoint nonempty subsets. By the minimality assumption, there exist instances $\mathcal{A}$ and $\mathcal{B}$ such that $\mathcal{A} \models \gamma_1 \wedge \neg\varphi$, and $\mathcal{B} \models \gamma_2 \wedge \neg\varphi$. (Here, we view each γ_i as the conjunction of its elements.) By additivity, $\mathcal{A} + \mathcal{B} \models \neg\varphi$. However, clearly $\mathcal{A} + \mathcal{B} \models \gamma$, whence $\mathcal{A} + \mathcal{B} \models \varphi$, a contradiction. ◀

4.4 Some consequences

Connected RA. We can give an analogue to Theorem 9 for RA instead of FO. Call an RA expression *connected* if every cartesian product subexpression of the form $e_1 \times e_2$ occurs in the context of a selection subexpression of the form $\sigma_{i=j}(e_1 \times e_2)$, with $i \in \{1, \ldots, n\}$ and $j \in \{n+1, \ldots, i+m\}$, where n and m are the arities of e_1 and e_2 respectively. In other words, in connected RA, pure cartesian products are disallowed, but equijoins are still allowed. Denote the queries expressible by connected RA expressions by CRA. It is clear that CRA queries are always additive. Again we have the converse:

▶ **Corollary 15.** $\mathrm{RA} \cap \mathrm{ADD} = \mathrm{CRA}$.

Indeed, it is well known that RA can be translated into FO, so, by Theorem 9, all we need to show is that CFO can be translated back into CRA. The translation from FO to RA given by Ullman [22] can be easily adapted to this effect.

Finite vs infinite. Theorem 9 holds in restriction to finite instances, as well as over all instances, but the two settings still need to be kept separate. CFO formulas are always additive, over all instances. This, however, should not lull the reader into believing that an FO formula that is additive over finite instances is also additive over all instances. Indeed, our results only show that such a formula is equivalent to a CFO formula over finite instances.

An example of an FO sentence over a binary relation R that is additive over finite instances but not in general, expresses that R consists of two total orders, over two disjoint sets, each order without a maximum.

4.5 Reduction from satisfiability

It is expected that additivity of FO formulas is an undecidable property. There is actually an extremely simple reduction from unsatisfiability. The proof of the correctness of this reduction, while short, is not entirely trivial and given in the Appendix.

▶ **Proposition 16.** *Let φ be an FO sentence over schema $\mathbf{S}$ and let S and T be two unary relation names not in $\mathbf{S}$. Then φ is unsatisfiable if and only if $\varphi \wedge \exists x\, S(x) \wedge \exists y\, T(y)$ is additive.*

Over all instances, the additive FO formulas are recursively enumerable; this actually follows from our theorem, but can also be seen by a direct reduction from additivity to unsatisfiability. We will see this reduction later when showing decidability of validity for guarded formulas. Over finite instances, the additive FO formulas are clearly co-r.e.

5 The guarded case

In this section we specialise Theorem 9 to the guarded fragment. We also show that additivity for guarded formulas is decidable and 2ExpTime-complete.

5.1 Guarded and connected formulas

We recall the syntax of the guarded fragment [5], which we denote by GF.

- Every atomic formula is guarded.
- If φ and ψ are guarded, then so are $\varphi \wedge \psi$, $\varphi \vee \psi$, and $\neg\varphi$.
- If α is an atomic formula, ψ is a guarded formula such that all free variables of ψ occur in α, and $\bar{y}$ is a subtuple of the free variables of α, then $\exists \bar{y}(\alpha \wedge \psi)$ is guarded.

Let us denote the class of queries expressible by guarded formulas by GF, and the class of queries expressible by formulas that are both guarded and connected by CGF. We will establish:

▶ **Theorem 17.** $\mathrm{GF} \cap \mathrm{ADD} = \mathrm{CGF}$.

Towards the proof, we introduce:

▶ **Definition 18.** *The* relaxed version *of CGF, denoted by* CGF^+*, is defined as follows:*

1. *Every atomic formula is in* CGF^+.
2. *If φ and ψ are in* CGF^+*, then so are $\varphi \wedge \psi$, $\varphi \vee \psi$, and $\varphi \wedge \neg\psi$, on condition that the rules for building connected formulas (Section 4.1) are satisfied.*
3. *If α is an atomic formula, and ψ is a boolean combination of* CGF^+ *formulas, all with at least one free variable, and all free variables of ψ occur in α, then both $\alpha \wedge \psi$ and $\exists \bar{y}(\alpha \wedge \psi)$ belong to* CGF^+*, with $\bar{y}$ a subtuple of the free variables of α.*

▶ **Example 19.** The sentence φ: $\exists x, y\,(R(x,y) \wedge (S(x) \vee T(y)))$ belongs to CGF^+, but it is not connected due to the subformula $S(x) \vee T(y)$. However, φ can be equivalently rewritten as $\exists x, y(R(x,y) \wedge S(x)) \vee \exists x, y(R(x,y) \wedge T(y))$ which is in CGF.

In general we note:

▶ **Lemma 20.** *Every* CGF^+ *formula is equivalent to a* CGF *formula.*

We also introduce a subclass of CGF^+ formulas, which we call the *simple guarded formulas*. These are the formulas that can be generated using only syntax rules 1 and 3 of CGF^+ (Definition 18). Simple guarded formulas are useful building blocks. First of all, they are additive. They are q-local if q is their quantifier rank. Furthermore we have the following lemma:

▶ **Lemma 21.** *Every* GF *formula is equivalent to a boolean combination of simple guarded formulas.*

So, we can start the proof of Theorem 17 with an additive boolean combination φ of simple guarded formulas, in disjunctive normal form. Within each clause ψ, we identify the *pseudopositive part* as the part consisting of all positive literals, plus all negative literals with a single free variable. The remainder of the clause is called the *negative part*. We will denote the pseudopositive part of a clause ψ by ψ^{pp} and the negative part by ψ^{neg}.

We may assume that φ has been simplified so that it has no redundancies in the following sense:

(A) If we would remove a subpart of the pseudopositive part of some clause, the resulting formula would not be logically equivalent to φ.

(B) If we would remove an entire clause, the resulting formula would again not be logically equivalent to φ.

(C) No clause contains a negated literal $\neg\eta$ such that η is a sentence that logically implies φ.

If, after these simplifications, we end up with an empty disjunction, then the original φ expresses the n-ary empty query, with n the number of free variables of φ. It is a simple exercise to express this query in CGF.

The plan of the proof is as follows. Each step relies on the additivity of φ; the steps are proven in the given order.

Claim 1: The pseudopositive part of each clause is connected.

Claim 2: In each clause, the free variables of the negative part are included in those of the pseudopositive part.

Claim 3: No clause can have negated literals that are sentences.

Claim 4: All clauses have the same free variables.

The result is a CGF^+ formula and we are done by Lemma 20.

5.2 Complexity of additivity for GF

We show:

▶ **Theorem 22.** *Additivity of guarded formulas is* 2ExpTime*-complete.*

Since satisfiability for guarded formulas is 2ExpTime-hard [9], the hardness follows directly from the simple reduction from satisfiability to additivity given by Proposition 16. Conversely, the membership in 2ExpTime is shown by a reduction from validity to unsatisfiability, which we next describe.

Consider an arbitrary guarded formula φ over a schema $\mathbf{S}$. We will construct a guarded formula φ' over a larger schema $\mathbf{S}^+$ such that φ is additive iff φ' is unsatisfiable. Specifically, $\mathbf{S}^+ = \mathbf{S} \cup \{U_1, U_2\}$, for two fresh unary relation names U_1 and U_2.

Satisfiability for guarded formulas of size n over relations of maximum arity a is decidable in time $2^{O(n)\cdot 2^{a \log a}}$ [10]. If n is the size of φ, our formula φ' will have size $2^{O(n)}$ but the arity does not change. Hence, we will obtain that additivity is in 2ExpTime as desired.

The high-level idea underlying the reduction can be sketched as follows. From φ, we construct guarded formulas φ^+ and $\varphi^\vee$ such that φ is additive iff φ^+ and $\varphi^\vee$ are equivalent over consistent instances (the definitions follow). A crucial property of consistency is that it can be checked via a guarded sentence φ_{cons}. Therefore, φ is additive iff $\varphi_{\mathsf{cons}} \models \varphi^+ \leftrightarrow \varphi^\vee$ iff $\varphi_{\mathsf{cons}} \wedge \neg(\varphi^+ \leftrightarrow \varphi^\vee)$ is unsatisfiable, while $\varphi_{\mathsf{cons}} \wedge \neg(\varphi^+ \leftrightarrow \varphi^\vee)$ is guarded.

The formula φ^+. We inductively define a translation $\cdot^+$ that converts φ into a formula φ^+ over $\mathbf{S}^+$ as follows:

1. If $\varphi = R(x_1, \ldots, x_n)$ for some $R \in \mathbf{S}$, then
$$\varphi^+ \coloneqq (R(x_1, \ldots, x_n) \wedge \bigwedge_{1 \le i \le n} U_1(x_i)) \vee (R(x_1, \ldots, x_n) \wedge \bigwedge_{1 \le i \le n} U_2(x_i)).$$
2. If $\varphi = (x = y)$, then $\varphi^+ \coloneqq (x = y)$.
3. The translation $\cdot^+$ commutes with the boolean connectives, i.e., $(\psi \wedge \chi)^+ \coloneqq \psi^+ \wedge \chi^+$, $(\psi \vee \chi)^+ \coloneqq \psi^+ \vee \chi^+$, and $(\neg\psi)^+ \coloneqq \neg\psi^+$.
4. If $\varphi = \exists y\, \psi$, then $\varphi^+ \coloneqq \exists y\, \psi^+$.

Strictly speaking, the formula φ^+ may not be guarded. However, it is readily transformed into a guarded formula of size $2^{O(|\varphi|)}$, where $|\varphi|$ denotes the size of φ. In what follows, for brevity, we write φ^+ for the exponentially sized rewritten guarded formula.

The formula $\varphi^\vee$. For $k = 1, 2$, we define φ^k as the formula obtained from φ by relativising all quantifiers and free variables to U_k. The latter means that quantifiers are of the form $\exists x \in U_k$, and for every free variable x we have a conjunct $U_k(x)$. A guarded existential formula $\exists y_1, \ldots, y_l(\alpha \wedge \psi)$ can be relativised as $\exists y_1, \ldots, y_l(\alpha \wedge U_k(y_1) \wedge \cdots \wedge U_k(y_l) \wedge \psi)$, which is still guarded. The formula $\varphi^\vee$ is then defined as the (guarded) formula $\varphi^1 \vee \varphi^2$.

Consistency. An $\mathbf{S}^+$-instance $\mathcal{A}^+$ is *consistent* if it admits a partition $(\mathcal{A}_1, \mathcal{A}_2)$ such that:

1. $\mathcal{A}_1$ is an $(\mathbf{S} \cup \{U_1\})$-instance and $\mathcal{A}_2$ is an $(\mathbf{S} \cup \{U_2\})$-instance.
2. $adom(\mathcal{A}_1) \cap adom(\mathcal{A}_2) = \emptyset$.
3. $\{a \mid U_1(a) \in \mathcal{A}_1\} = adom(\mathcal{A}_1)$ and $\{b \mid U_2(b) \in \mathcal{A}_2\} = adom(\mathcal{A}_2)$.

So, consistent instances describe pairs of domain-disjoint instances. Applied to a consistent instance $(\mathcal{A}_1, \mathcal{A}_2)$, we see that φ^+ returns $\varphi(\mathcal{A}_1 \cup \mathcal{A}_2)$, while $\varphi^\vee$ returns $\varphi(\mathcal{A}_1) \cup \varphi(\mathcal{A}_2)$. Hence we obtain:

▶ **Lemma 23.** *The following are equivalent:*

1. *φ is additive.*
2. *For every consistent $\mathbf{S}^+$-instance $\mathcal{A}^+$, $\varphi^+(\mathcal{A}^+) = \varphi^\vee(\mathcal{A}^+)$.*

It remains to show that consistency can be checked via a guarded formula, which will give us the desired reduction. In fact, consistency can be checked via the formula

$$\varphi_{\mathsf{cons}} \coloneqq \neg\exists x(U_1(x) \wedge U_2(x)) \wedge \exists x\, U_1(x) \wedge \exists x\, U_2(x) \wedge \psi_1 \wedge \psi_2 \wedge \chi,$$

where, for $k = 1, 2$, and assuming that $\mathbf{S} = \{R_1, \ldots, R_n\}$ (and R_i is n_i-ary),

$$\psi_k \coloneqq \forall x\,(U_k(x) \to \bigvee_{i=1}^{n} \exists y_1, \ldots, y_{n_i - 1}(R_i(x, y_1, \ldots, y_{n_i-1}) \vee R_i(y_1, x, y_2, \ldots, y_{n_i-1}) \vee \cdots \vee R_i(y_1, \ldots, y_{n_i-1}, x))),$$

and

$$\chi \coloneqq \bigwedge_{i=1}^{n} \forall x_1, \ldots, x_{n_i} \, (R_i(x_1, \ldots, x_{n_i}) \rightarrow ((U_1(x_1) \wedge \cdots \wedge U_1(x_{n_i})) \vee (U_2(x_1) \wedge \cdots \wedge U_2(x_{n_i}))))$$

The above formula can be easily transformed in a guarded formula. We now obtain, as announced, that φ is additive if and only if $\varphi_{\mathsf{cons}} \models \varphi^{+} \leftrightarrow \varphi^{\vee}$.

▶ Remark. At the end of Section 4.4 we gave an example of an FO formula additive over finite instances but not over all instances. Since GF has the finite model property, the above reduction to unsatisfiability implies that a GF formula that is additive over finite instances is automatically additive over all instances.

6 The positive-existential case

In this final section, we concentrate on queries expressed via *positive-existential* (PE) first-order formulas, i.e., formulas that use only $\wedge$, $\vee$, and existential quantification. As always we also use PE to denote the class of all queries expressible in this manner.

It is well-known [1] that PE has the same expressive power as the class of *unions of conjunctive queries (UCQ)*, i.e., disjunctive FO-formulas of the form $\bigvee_{1 \leq i \leq n} \varphi_i(\bar{x})$, with $\varphi_i(\bar{x}) = \exists \bar{y} \, (\alpha_{i,1} \wedge \cdots \wedge \alpha_{i,m_i})$, where each $\alpha_{i,j}$ is an atomic formula. Formulas of the form $\varphi_i(\bar{x})$ are called *conjunctive queries*. In earlier work [6], two of us already showed that UCQ $\cap$ ADD = CUCQ, where CUCQ refers to the connected UCQs. As a consequence, also PE $\cap$ ADD = CPE, where CPE refers to the connected PE formulas.

In this section, we focus on the complexity of deciding additivity for PE formulas. For UCQs, additivity was already shown to be NP-complete [6]. Unfortunately, in general, it takes exponential time to convert a PE formula into a union of conjunctive queries. Thus, the naive algorithm provides only an exponential time upper bound, which is not optimal. We can, however, show that our problem lies at the second level of the polynomial hierarchy:

▶ **Theorem 24.** *Additivity of* PE *formulas is* Π_2^P*-complete. The lower bound holds even over unary and binary relations.*

Towards the proof, let us first introduce some useful notions. A *component* of a conjunctive query $\varphi_i(\bar{x}) = \exists \bar{y} \, (\alpha_1 \wedge \cdots \wedge \alpha_n)$ is a connected formula $\psi = \exists \bar{y} \, (\alpha_{i_1} \wedge \cdots \wedge \alpha_{i_m})$, where $\{i_1, \ldots, i_m\} \subseteq \{1, \ldots, n\}$, such that, for every $j \in \{1, \ldots, n\} \setminus \{i_1, \ldots, i_m\}$, the formula $\alpha_{i_1} \wedge \cdots \wedge \alpha_{i_m} \wedge \alpha_j$ is not connected anymore. Given two formulas $\varphi(\bar{x})$ and $\psi(\bar{x})$ over a schema $\mathbf{S}$, we say that φ is *contained* in ψ, written $\varphi \subseteq \psi$, if $\varphi(\mathcal{A}) \subseteq \psi(\mathcal{A})$ for every $\mathbf{S}$-instance $\mathcal{A}$. We need the following:

▶ **Lemma 25** ([6])**.** *Consider a union of conjunctive queries $\varphi(\bar{x})$ of the form $\bigvee_{1 \leq i \leq n} \varphi_i(\bar{x})$. The following are equivalent:*

- *φ is additive;*
- *for each $i \in \{1, \ldots, n\}$, there exists a component $\psi(\bar{x})$ of $\varphi_i(\bar{x})$ such that $\psi \subseteq \varphi$.*

Consider an arbitrary PE-formula φ. Even though we cannot efficiently convert φ into an equivalent union of conjunctive queries $\bigvee_{1 \leq i \leq n} \varphi_i$, we can non-deterministically construct a disjunct φ_i in polynomial time. This fact, together with Lemma 25, leads to the following non-deterministic algorithm for checking whether φ is *not* additive:

1. Non-deterministically construct a disjunct φ_i of the union of conjunctive queries obtained after converting φ into an equivalent union of conjunctive queries.
2. Compute the set C of all the components of φ_i.
3. If, for each $\psi \in C$, it holds that $\psi \not\subseteq \varphi$, then ACCEPT; otherwise, REJECT.

It is easy to verify, due to Lemma 25, that φ is *not* additive iff the above procedure accepts. Observe now that steps 1 and 2 are feasible in polynomial time. Finally, during step 3, we need to check polynomially many times for non-containment of a conjunction of atomic formulas, i.e., a conjunctive query, into a PE-formula. The latter is feasible in coNP. Thus, the obtained upper bound for non-additivity is $\mathrm{NP}^{\mathrm{NP}} = \Sigma_2^P$, as needed.

It remains to establish that additivity for PE-formulas is Π_2^P-hard. This is shown by a reduction from the problem of containment for PE-sentences, i.e., given two PE-sentences φ and ψ, to decide whether $\varphi \subseteq \psi$, which is Π_2^P-hard [19]. Actually, this problem remains Π_2^P-hard even if the left-hand side formula is connected, and the relation symbols have arity at most two; this is implicit from the work by Bienvenu et al. [7].

Given a connected PE-formula φ, and an arbitrary PE-formula ψ, we can show that $\varphi \subseteq \psi$ iff the PE-formula $\varphi \wedge P(x_\varphi) \wedge \psi$ is additive, where x_φ is an existentially quantified variable in φ, and P a fresh relation name. This shows that additivity for PE-formulas is Π_2^P-hard even for unary and binary relations.

7 Conclusion

We conclude with a few directions for further research.

1. Investigate the complexity, in terms of formula length, of the shortest connected formula equivalent to an additive formula.
2. Our reduction from additivity to unsatisfiability for GF formulas is exponential. However, the GF formula that is produced has exponential length only to get rid of disjunctive guards. If disjunctive guards would be allowed, the reduction would be polynomial and would preserve bounded arity. We conjecture that additivity for GF formulas of bounded arity is actually in ExpTime, and plan to investigate this in the near future.
3. Also, our reduction from additivity to unsatisfiability works in general, for FO formulas, and for other reasonable logics. It is interesting to apply the reduction to other decidable logics, and see if the reduction produces a formula in the same logic. For example, our reduction also works for FO^2, thus additivity for FO^2 formulas is decidable.
4. Give a short classical model-theoretic proof, e.g., involving the compactness theorem, that every FO formula, additive over all structures, is equivalent to a CFO formula.
5. Our version for additivity is rather strict in the context of distributed computation. Considering classes of queries that distribute over other kinds of partitions for instances is an interesting line for future research.

References

1 S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

2 N. Alechina and Y. Gurevich. Syntax vs semantics on finite structures. In J. Mychielski, G. Rozenberg, and A. Salomaa, editors, *Structures in Logic and Computer Science*, volume 1261 of *Lecture Notes in Computer Science*, pages 14–33. Springer, 1997.

3 T.J. Ameloot, B. Ketsman, F. Neven, and D. Zinn. Weaker forms of monotonicity for declarative networking: A more fine-grained answer to the CALM-conjecture. *ACM Transactions on Database Systems*, 40(4):article 21, 2016.

4 T.J. Ameloot, B. Ketsman, F. Neven, and D. Zinn. Datalog queries distributing over components. *ACM Transactions on Computational Logic*, 18:article 5, 2017.

5 H. Andréka, I. Németi, and J. van Benthem. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27(3):217–274, 1998.

6 G. Berger and A. Pieris. Ontology-mediated queries distributing over components. In S. Kambhampati, editor, *Proceedings 25th International Joint Conference on Artificial Intelligence*, pages 943–949. IJCAI/AAAI Press, 2016.

7 M. Bienvenu, C. Lutz, and F. Wolter. Query containment in description logics reconsidered. In G. Brewka, T. Eiter, and S.A. McIlraith, editors, *Principles of Knowledge Representation and Reasoning: Proceedings 13th KR*. AAAI Press, 2012.

8 H. Gaifman. On local and nonlocal properties. In *Proceedings of the Herbrand symposium (Marseilles, 1981)*, volume 107 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1982.

9 E. Grädel. On the restraining power of guards. *Journal of Symbolic Logic*, 64(4):1719–1742, 1999.

10 E. Grädel and I. Walukiewicz. Guarded fixed point logic. In *Proceedings 14th Annual IEEE Symposium on Logic in Computer Science*, pages 45–54, 1999.

11 J.M. Hellerstein. The declarative imperative: Experiences and conjectures in distributed logic. *SIGMOD Record*, 39(1):5–19, 2010.

12 D. Leinders, M. Marx, J. Tyszkiewicz, and J. Van den Bussche. The semijoin algebra and the guarded fragment. *Journal of Logic, Language and Information*, 14:331–343, 2005.

13 D. Leinders and J. Van den Bussche. On the complexity of division and set joins in the relational algebra. *J. Comput. Syst. Sci.*, 73(4):538–549, 2007.

14 M. Otto. Elementary proof of the van Benthem-Rosen characterization theorem. Fachbereich Informatik online preprint 2342, TU Darmstadt, 2004. URL: `http://www3.mathematik.tu-darmstadt.de/fb/mathe/preprints.html`.

15 M. Otto. Modal and guarded characterisation theorems over finite transition systems. *Annals of Pure and Applied Logic*, 130:173–205, 2004.

16 M. Otto. Bisimulation invariance and finite structures. In Z. Chatzidakis, P. Koepke, and W. Pohlers, editors, *Logic Colloquium '02*, volume 27 of *Lecture Notes in Logic*, pages 276–298. Cambridge University Press, 2006.

17 E. Rosen. Modal logic over finite structures. *Journal of Logic, Language and Information*, 5:427–439, 1997.

18 E. Rosen. Some aspects of model theory and finite structures. *Bulletin of Symbolic Logic*, 8:380–403, 2002.

19 Y. Sagiv and M. Yannakakis. Equivalence among Relational Expressions with the Union and Difference Operators. *J. ACM*, 27(4):633–655, 1980.

20 D. Surinx, J. Van den Bussche, and D. Van Gucht. The primitivity of operators in the algebra of binary relations under conjunctions of containments. In *Proceedings 32nd Annual ACM/IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 2017.

21 D. Surinx, J. Van den Bussche, and D. Van Gucht. A framework for comparing query languages in their ability to express boolean queries. In *Foundations of Information and Knowledge Systems*, pages 360–378, 2018.

22 J.D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume I. Computer Science Press, 1988.

23 J. van Benthem. *Modal Logic and Classical Logic*. Bibliopolis, Naples, 1983.

Characterizing Tractability of Simple Well-Designed Pattern Trees with Projection

Stefan Mengel
CNRS, CRIL UMR 8188, Lens, France
mengel@cril.fr

Sebastian Skritek
Faculty of Informatics, TU Wien, Vienna, Austria
skritek@dbai.tuwien.ac.at

Abstract

We study the complexity of evaluating well-designed pattern trees, a query language extending conjunctive queries with the possibility to define parts of the query to be optional. This possibility of optional parts is important for obtaining meaningful results over incomplete data sources as it is common in semantic web settings.

Recently, a structural characterization of the classes of well-designed pattern trees that can be evaluated in polynomial time was shown. However, projection – a central feature of many query languages – was not considered in this study. We work towards closing this gap by giving a characterization of all tractable classes of simple well-designed pattern trees with projection (under some common complexity theoretic assumptions). Since well-designed pattern trees correspond to the fragment of well-designed {AND, OPTIONAL}-SPARQL queries this gives a complete description of the tractable classes of queries with projections in this fragment that can be characterized by the underlying graph structures of the queries.

2012 ACM Subject Classification Theory of computation → Database query languages (principles); Theory of computation → Database query processing and optimization (theory)

Keywords and phrases SPARQL, well-designed pattern trees, query evaluation, FPT, characterizing tractable classes

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.20

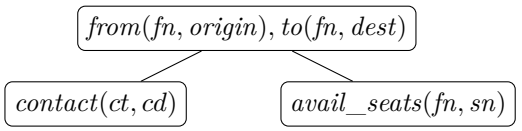
Related Version An extended version of the paper is available at `https://arxiv.org/abs/1712.08939`.

Funding *Sebastian Skritek*: Supported by the Austrian Science Fund (FWF): P30930-N35.

1 Introduction

Well-designed pattern trees (wdPTs) are a query formalism well-suited to deal with the ever increasing amount of incomplete data. Well-designed pattern trees over SPARQL triple patterns are equivalent to the class of well-designed {AND, OPTIONAL}-SPARQL queries [19] and were in fact originally introduced as a formalism to more easily study SPARQL queries. By replacing triple patterns with relational atoms, wdPTs can also be seen as an extension of Conjunctive Queries (CQs): a wdPT is a rooted tree where each node represents a conjunction of atoms, and the tree structure represents a nesting of optional matching. The idea is to start evaluating the CQ at the root and to iteratively extend the retrieved results as much as possible by the results of the CQs in the other nodes. This allows wdPTs to return partial answers in case that not the complete query can be mapped into the database – unlike CQs which in such a situation return no answer.

Well-designed pattern trees and the corresponding SPARQL fragment represent an important class of SPARQL queries and have been studied intensively within the last decade [19, 16, 2, 21, 20, 14, 3, 1, 22]. Thus many properties of and problems related to these



$\mathbf{D} = \{from(1, \text{“LHR”}), to(1, \text{“LIS”}),$
$from(2, \text{“LHR”}), to(2, \text{“LIS”}),$
$contact(email, \text{“e@ma.il”}),$
$avail\_seats(2, \text{“1A”})\}$

Figure 1 The wdPT p and database $\mathbf{D}$ from Example 1.

queries are now well understood. For example, the evaluation problem for wdPTs (i.e., given a wdPT, a database and a mapping, is this mapping an answer to the wdPT over the database) is coNP-complete for projection free wdPTs [19] and Σ_2P-complete in the presence of projection [16]. However, certain tractable classes of wdPTs have been identified [3]. The main idea there is to extend known tractability conditions for CQs to wdPTs. However, the question of characterizing exactly the classes of wdPTs for which tractable query evaluation is possible – and thus the question how good the approach of extending tractability conditions of CQs to wdPTs is – has been largely ignored. Only very recently, this question was addressed for wdPTs without projection, and a characterization of the classes for which query evaluation is in PTIME was given [22]. Notably, as also observed for Boolean Conjunctive Queries [12, 11], for wdPTs without projection these classes coincide with the ones for which evaluation is in FPT.

However, [22] does not touch projection, an essential and central feature of query languages. Thus the question “What are all tractable classes of wdPTs with projection?” remains open. We work towards closing this gap.

One observation consistently made in all the aforementioned work on wdPTs is that problems become much more complex once projection is included. This is true for both, the computational complexity of the problems (e.g., as mentioned, for the evaluation problem it increases from coNP- to Σ_2P-completeness; for classical query containment, the NP-complete problem becomes even undecidable [21]) as well as for establishing these results.

This is because of the particular semantics of well-designed SPARQL with projection. For wdPTs *without* projection, given some database, the set of answers consist of all variable mappings such that there exists a subtree of the wdPT satisfying the following conditions: first, it must contain the root node of the tree. Second, the set of variables occurring in the subtree must be the same as the domain of the mapping. Third, the mapping must map each atom in the subtree into the database, and fourth, no extension of the mapping is allowed to map all atoms of any child node of any node in the subtree into the database. This is illustrated by the following example (a precise definition is given in Section 2).

▶ **Example 1.** Figure 1 shows a wdPT p with three nodes where the top node represents the root of the tree, having two child nodes as depicted. At its root node, the query is looking for information on flights: flight number, origin and destination. This information should be extended by some contact information (left child), and information on available seats on the flight (right child) in case any of this information is available. Observe that the two extensions are independent of each other. The equivalent SPARQL query (replacing relational atoms by triple patterns) would be

```
{{?fn from ?origin . ?fn to ?dest} OPTIONAL {?ct contact ?cd}}
                                        OPTIONAL {?fn avail_seats ?sn}
```

For the database instance $\mathbf{D}$ also shown in the figure, the mapping μ with $\mu(fn) = 1$, $\mu(origin) =$ “LHR”, $\mu(dest) =$ “LIS”, $\mu(ct) = email$, and $\mu(cd) =$ “e@ma.il” is an answer to p over $\mathbf{D}$. This is because of the subtree of p consisting of the root node and the left

child. It can be checked that it satisfies all four conditions. For the fourth condition, just observe that there exists no extension of μ that maps $avail\_seats(fn, sn)$ into $\mathbf{D}$. Because of the fourth condition, the mapping ν with $\nu(fn) = 2$, $\nu(origin) =$ "LHR", $\nu(dest) =$ "LIS", $\nu(ct) = email$, and $\nu(cd) =$ "e@ma.il" is no solution, because this mapping can be extended by $\nu(sn) =$ "1A" in a way that maps $avail\_seats(fn, sn)$ also into $\mathbf{D}$.

Thus, without projection, the only hard part in deciding whether some mapping is a solution is to check for the existence of an extension, since this basically includes a homomorphism test. However, for wdPTs *with* projection, a mapping is a solution if there exists an extension of this mapping to some subset of the existential variables in the tree, such that the extended mapping is a solution to the wdPT considered without projection.

▶ **Example 2.** Consider the wdPT from Example 1, but now assume that the flight number fn is an existential variable and thus not part of the output. Then μ with $\mu(origin) =$ "LHR", $\mu(dest) =$ "LIS", $\mu(ct) = email$, and $\mu(cd) =$ "e@ma.il" is a solution because of the extension $\mu(fn) = 1$. Observe that the extension $\mu(fn) = 2$ does not witness μ to be a solution, since, as already discussed before, this mapping is not maximal.

As a consequence, beside testing some mapping for maximality, as a second source of hardness, different mappings on the existential variables have to be taken into account.

Besides the already mentioned increased complexity of many problems, it was also observed that for wdPTs with projection it is no longer the case that the classes for which query evaluation is in PTIME and in FPT coincide [15]. Thus, in this setting, the choice of the tractability notion makes a difference when describing all tractable classes.

We choose to study the complexity of query evaluation in the model of parameterized complexity where, as usual, we take the size of the query as the parameter. As already argued in [18], this model allows for a more fine-grained analysis than the classical perspectives of data- and query complexity. In parameterized complexity, query answering is considered tractable, formally in FPT, if, after a preprocessing that only depends on the query, the actual evaluation can be done in polynomial time [9, 10]. Parameterized complexity has found many applications in the complexity of query evaluation problems, see e.g. [12, 11, 17, 4, 22].

In our efforts to better understand the tractability frontier for wdPTs, we provide a complete characterization of the tractable classes of *simple* wdPTs, i.e., wdPTs where no two atoms share the same relation symbol. Because of the relationship between wdPTs and well-designed {AND, OPTIONAL}-SPARQL queries, this immediately gives a complete description of the tractable classes of well-designed {AND, OPTIONAL}-SPARQL queries with projection that can be characterized by only considering the graph structures of the queries, similar e.g. to the work of [12, 4]. We note that the results showing the existence of classes of wdPTs for which the evaluation problem is NP-hard but in FPT can be easily extended to simple wdPTs. Moreover, our tractability criteria are not restricted to simple wdPTs. In fact, the same tractability criteria can also directly be applied to give tractable classes of non-simple *wdPTs*. However, in this case, there are classes of queries that do not satisfy our tractability criteria and are still tractable. Thus, the restriction to simple wdPTs is crucial for the lower bounds.

Summary of results and organization of the paper. We study the following decision problem: Given a wdPT, a database, and a mapping, is the mapping a solution of the wdPT over the database? This is the standard formulation of the evaluation problem usually studied (cf. [16, 13, 22, 3]). It reveals the influence of the optional query parts on the evaluation problem, which is lost e.g. when considering Boolean queries. Instead of just SPARQL

triple patterns, we consider the more general case of wdPTs with arbitrary relational atoms where we always assume that the classes of queries we consider have bounded arity. Our main result is a characterization of the classes of simple wdPTs with projection that allow fixed-parameter tractable query evaluation.

After some preliminaries in Section 2, we define two tractability conditions in Section 3. By comparing these conditions with the tractability criterion from [22], we discuss how they describe the additional complexity introduced by projection. Note that some of the conditions provided here have precursors in [3] and [15] that had to be carefully refined to provide a fine-grained complexity analysis.

In Section 4 we prove that the two tractability conditions imply FPT membership of the evaluation problem by presenting an algorithm that exploits these conditions.

In Section 5 we then show that both tractability conditions are indeed necessary for a class of simple wdPTs to be tractable. That is, we show that if either of them is not satisfied by a class of wdPTs, the evaluation problem for this class is either W[1]- or coW[1]-hard.

In Section 6, we discuss our results and also potential extensions of the tractability conditions to conclude the paper.

2 Preliminaries

Basics. Let $\mathcal{C}onst$ and $\mathcal{V}ar$ be two disjoint countable infinite sets of constants and variables, respectively. A *relational schema* σ is a set $\{R_1, \ldots, R_n\}$ of relation symbols R_i, each having an assigned arity $r_i \geq 0$. A *relational atom* $R_i(\vec{v})$ over σ consists of a relation symbol $R_i \in \sigma$ and a tuple $\vec{v} \in (\mathcal{C}onst \cup \mathcal{V}ar)^{r_i}$. For an atom $\tau = R_i(\vec{v})$, let $\mathsf{dom}(\tau)$ denote the set of variables and constants occurring in τ. This extends to sets $\mathbf{R} = \{\tau_1, \ldots, \tau_m\}$ of atoms as $\mathsf{dom}(\mathbf{R}) = \bigcup_{i=1}^{m} \mathsf{dom}(\tau_i)$. Furthermore, $\mathsf{var}(\tau) = \mathsf{dom}(\tau) \cap \mathcal{V}ar$ and $\mathsf{var}(\mathbf{R}) = \mathsf{dom}(\mathbf{R}) \cap \mathcal{V}ar$. Observe that, by slight abuse of notation, we use operators $\cup, \cap, \setminus$ also between sets $\mathcal{V}$ and tuples $\vec{v}$ of variables and constants. For example, $\mathsf{var}(\tau) = \vec{v} \cap \mathcal{V}ar$. We call a set of atoms *simple* if no relation symbol appears more than once in it.

Similarly, for a mapping μ we denote with $\mathsf{dom}(\mu)$ the set of elements on which μ is defined. For a mapping μ and a set $\mathcal{V} \subseteq \mathcal{V}ar$, we use $\mu|_{\mathcal{V}}$ to describe the restriction of μ to the variables in $\mathsf{dom}(\mu) \cap \mathcal{V}$. We say that a mapping μ is an *extension* of a mapping ν if $\mu|_{\mathsf{dom}(\nu)} = \nu$, and that two mappings are *compatible* if they agree on the shared variables.

For a set $\mathbf{A}$ of atoms and a set $\mathcal{A} \subseteq \mathsf{dom}(\mathbf{A})$, we write $\mathbf{A} \setminus \mathcal{A}$ to denote the restriction of $\mathbf{A}$ to $\mathsf{dom}(\mathbf{A}) \setminus \mathcal{A}$. That is, we substitute every atom $R(\vec{v}) \in \mathbf{A}$ by an atom $R^s(\vec{v}')$, where $\vec{v}'$ is obtained from $\vec{v}$ by removing elements of $\mathcal{A}$, and s is the list of the removed positions.

A *database* $\mathbf{D}$ over σ is a finite set of atoms over σ with $\mathsf{var}(\mathbf{D}) = \emptyset$. For a database $\mathbf{D}$ and relation symbol R we denote by $R^{\mathbf{D}}$ the set of all atoms in $\mathbf{D}$ with relation symbol R.

Homomorphisms and Conjunctive Queries. A homomorphism h between two sets $\mathbf{A}$ and $\mathbf{B}$ of atoms over σ is a mapping $h\colon \mathsf{dom}(\mathbf{A}) \to \mathsf{dom}(\mathbf{B})$ such that for all atoms $R(\vec{v}) \in \mathbf{A}$ we have $R(h(\vec{v})) \in \mathbf{B}$, and such that $h(x) \neq x$ is only allowed if $x \in \mathsf{var}(\mathbf{A})$ (we thus restrict the definition of homomorphisms to $\mathsf{var}(\mathbf{A})$, the extension to constants via the identity mapping is implicit). We write $h\colon \mathbf{A} \to \mathbf{B}$ to denote a homomorphism h from $\mathbf{A}$ to $\mathbf{B}$.

We write CQs q as $Ans(\vec{x}) \leftarrow \mathbf{B}$, where the body $\mathbf{B} = \{R_1(\vec{v}_1), \ldots, R_m(\vec{v}_m)\}$ is a set of atoms and $\vec{x}$ are the *free variables*. A Boolean CQ (BCQ) is a CQ with no free variables. We define $\mathsf{var}(q) = \mathsf{var}(\mathbf{B})$. The existential variables are implicitly given by $\mathsf{var}(\mathbf{B}) \setminus \vec{x}$. The result $q(\mathbf{D})$ of q over a database $\mathbf{D}$ is the set of tuples $\{h(\vec{x}) \mid h\colon \mathbf{B} \to \mathbf{D}\}$.

Graphs. We consider only undirected, simple graphs $G = (V, E)$ with standard notations but sometimes write $t \in G$ to refer to a node $t \in V(G)$. A graph G_2 is a subgraph of a graph G_1 if $V(G_2) \subseteq V(G_1)$ and $E(G_2) \subseteq E(G_1)$. A tree is a connected, acyclic graph. A subtree is a connected, acyclic subgraph. A *rooted tree* T is a tree with one node $r \in T$ marked as its root. Given two nodes $t, \hat{t} \in T$, we say that $\hat{t}$ is an ancestor of t if $\hat{t}$ lies on the path from r to t. Likewise, $\hat{t}$ is the parent node of t (t is a child of $\hat{t}$) if $\hat{t}$ is an ancestor of t and $\{t, \hat{t}\} \in E(T)$. For a subtree T' of T, a node $t \in T$ is a child of T' if $t \notin T'$ and $\hat{t} \in T'$ for the parent node $\hat{t}$ of t. We write $ch(T')$ for the set of all children of T'. For a node $t \in T$ the set of nodes on the path from r to the parent node of t is denoted by $\mathsf{branch}(t)$. Moreover, $\mathsf{cbranch}(t) = \mathsf{branch}(t) \cup \{t\}$.

A *tree decomposition* of a graph $G = (V, E)$ is a pair (T, ν), where T is a tree and $\nu \colon V(T) \to 2^V$, that satisfies the following: (1) For each $u \in V$ the set $\{s \in V(T) \mid u \in \nu(s)\}$ is a connected subset of $V(T)$, and (2) each edge of E is contained in at least one of the sets $\nu(s)$, for $s \in V(T)$. The *width* of (T, ν) is $(\max\{|\nu(s)| \mid s \in V(T)\}) - 1$. The *treewidth* of G is the minimum width of its tree decompositions.

For a set $\mathbf{A}$ of atoms, the *Gaifman graph* of $\mathbf{A}$ is the graph $G = (V, E)$ with $V = \{v_i \mid v_i \in \mathsf{var}(\mathbf{A})\}$ and E contains an edge $\{v_i, v_j\}$ if v_i and v_j occur together in some atom in $\mathbf{A}$. The treewidth of a set of atoms is the treewidth of its Gaifman graph.

Well-designed pattern trees (wdPTs). A *pattern tree* (short: PT) p over a relational schema σ is a tuple $(T, \lambda, \mathcal{X})$ where T is a rooted tree and λ maps each node $t \in T$ to a set of relational atoms over σ. We may write $((T, r), \lambda, \mathcal{X})$ to emphasize that r is the root node of T. The set $\mathcal{X}$ of variables denotes the *free variables* of the PT. For a PT $(T, \lambda, \mathcal{X})$ and a subtree T' of T, let $\lambda(T') = \bigcup_{t \in V(T')} \lambda(t)$. We may write $\mathsf{var}(t)$ instead of $\mathsf{var}(\lambda(t))$, and $\mathsf{var}(T')$ instead of $\mathsf{var}(\lambda(T'))$. We further define $\mathsf{fvar}(t) = \mathsf{var}(t) \cap \mathcal{X}$ as the free variables in t. Again this definition extends naturally to subtrees T' of T. We call a PT $(T, \lambda, \mathcal{X})$ *projection free* if $\mathcal{X} = \mathsf{var}(T)$ and may write (T, λ) to emphasize a PT to be projection free. The size $|p|$ of a pattern tree is $\sum_{t \in V(T)} |\lambda(t)|$.

Well-designed PTs restrict the distribution of variables among their nodes.

▶ **Definition 3** (Well-Designed Pattern Tree (wdPT)). *A PT $(T, \lambda, \mathcal{X})$ is* well-designed *if for every variable $y \in \mathsf{var}(T)$, the set of nodes of T where y appears is connected.*

As an immediate consequence of this restriction, in a wdPT $p = (T, \lambda, \mathcal{X})$, for every variable $y \in \mathsf{var}(T)$ there exists a unique node $t \in T$ such that $y \in \mathsf{var}(t)$ and all nodes $t' \in T$ with $y \in \mathsf{var}(t')$ are descendants of t.

Evaluating a wdPT p with free variables $\mathcal{X}$ over a database $\mathbf{D}$ returns a set $p(\mathbf{D})$ of mappings $\mu \colon \mathcal{V} \to \mathsf{dom}(\mathbf{D})$ with $\mathcal{V} \subseteq \mathcal{X}$. We follow the characterization of $p(\mathbf{D})$ in terms of maximal subtrees [16], but borrow the term pp-solution from [13].

▶ **Definition 4** (pp-solution). *For a wdPT $p = ((T, r), \lambda)$ and a database $\mathbf{D}$, a mapping $\mu \colon \mathcal{V} \to \mathsf{dom}(\mathbf{D})$ (with $\mathcal{V} \subseteq \mathsf{var}(T)$) is a* potential partial solution (pp-solution) *to p over $\mathbf{D}$ if there is a subtree T' of T containing r such that $\mu \colon \lambda(T') \to \mathbf{D}$.*

The semantics of wdPTs can now be defined in terms of maximal pp-solutions.

▶ **Definition 5** (Semantics of wdPTs). *Let $p = (T, \lambda, \mathcal{X})$ be a wdPT, and let $p' = (T, \lambda, \mathsf{var}(T))$, i.e., the projection-free wdPT retrieved from p by considering all of its variables as free, and let $\mathbf{D}$ be a database. The set $p'(\mathbf{D})$ contains all pp-solutions μ to p' over $\mathbf{D}$ such that there exists no pp-solution μ' to p' over $\mathbf{D}$ that is a proper extension of μ.*

The set $p(\mathbf{D})$ is then defined as $p(\mathbf{D}) = \{\mu|_{\mathcal{X}} \mid \mu \in p'(\mathbf{D})\}$.

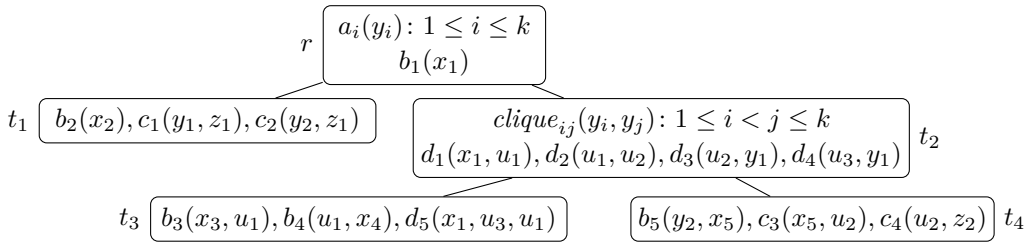


Figure 2 The well-designed pattern tree of Example 6.

▶ **Example 6.** Consider the PT $p = (T, \lambda, \mathcal{X})$ depicted in Figure 2, where k may be any integer with $k \geq 2$, and $\mathcal{X} = \{x_1, x_2, x_3, x_4, x_5\}$. All variable occurrences in p are connected, thus it is well-designed. If for example the atom $a_2(y_2)$ was missing in the root node, the tree would not be well-designed because of the occurrences of y_2 in both, t_1 and t_2.

Consider a database $\mathbf{D}$ that, for each atom $R(\vec{v})$ in $\lambda(T)$ contains one atom $R(1, \ldots, 1)$ (i.e., with the value 1 at each position) and in addition the atoms $d_1(1, 2)$, $d_2(2, 2)$, and $d_3(2, 1)$. Then $p(\mathbf{D}) = \{\mu_1, \mu_2\}$ where $\mathsf{dom}(\mu_1) = \{x_1, \ldots, x_5\}$, $\mathsf{dom}(\mu_2) = \{x_1, x_2\}$, and $\mu_i(x) = 1$ for $i \in \{1, 2\}$ and all $x \in \mathsf{dom}(\mu_i)$. This is because of the following extensions μ_1' and μ_2' of μ_1 and μ_2, respectively. For μ_1', we have $\mathsf{dom}(\mu_1') = \mathsf{var}(T)$ and $\mu_1'(x) = 1$ for all $x \in \mathsf{dom}(\mu_1')$, and for μ_2' we have $\mathsf{dom}(\mu_2') = \mathsf{var}(\lambda(\{r, t_1, t_2\}))$ with $\mu_2'(x) = 1$ for all $x \in \mathsf{dom}(\mu_2')$ except for u_1 and u_2, for which $\mu_2'(u_i) = 2$. Observe that μ_2' maps r, t_1, and t_2 into $\mathbf{D}$, but cannot be extended to neither t_3 nor t_4.

Parameterized complexity. We only give a bare-bones introduction to parameterized complexity and refer the reader to [8] for more details. Let Σ be a finite alphabet. A *parameterization* of Σ^* is a polynomial time computable mapping $\kappa\colon \Sigma^* \to \mathbb{N}$. A *parameterized problem* over Σ is a pair (L, κ) where $L \subseteq \Sigma^*$ and κ is a parameterization of Σ^*. We refer to $x \in \Sigma^*$ as the instances of a problem, and to the numbers $\kappa(x)$ as the parameters.

A parameterized problem $E = (L, \kappa)$ belongs to the class FPT of *fixed-parameter tractable* problems if there is an algorithm A deciding L, a polynomial *pol*, and a computable function $f\colon \mathbb{N} \to \mathbb{N}$ such that the running time of A on every input $x \in \Sigma^*$ is at most $f(\kappa(x)) \cdot pol(|x|)$.

In this paper, for classes $\mathcal{P}$ of wdPTs, we study the problem p-Eval($\mathcal{P}$) defined below.

p-Eval($\mathcal{P}$)	
INSTANCE:	A wdPT $p \in \mathcal{P}$, a database $\mathbf{D}$, and a mapping μ.
PARAMETER:	$\lvert p \rvert$
QUESTION:	Does $\mu \in p(\mathbf{D})$ hold?

We always assume that the arity of all atoms of the queries in $\mathcal{P}$ is bounded by a constant, i.e., that there is a constant c (possibly depending on $\mathcal{P}$) such that no atom in the queries in $\mathcal{P}$ has an arity of more than c.

Let $E = (L, \kappa)$ and $E' = (L', \kappa')$ be parameterized problems. An FPT-*reduction* from E to E' is a mapping $R\colon \Sigma^* \to (\Sigma')^*$ such that (1) for all $x \in \Sigma^*$ we have $x \in L$ if and only if $R(x) \in L'$, (2) there is a computable function f and a polynomial *pol* such that $R(x)$ can be computed in time $f(\kappa(x)) \cdot pol(|x|)$, and (3) there is a computable function $g\colon \mathbb{N} \to \mathbb{N}$ such that $\kappa'(R(x)) \leq g(\kappa(x))$ for all $x \in \Sigma^*$.

Of the rich parameterized hardness theory, we will only use the classes $\mathsf{W}[1]$ and $\mathsf{coW}[1]$ of parameterized problems. To keep this introduction short, we define a parameterized problem

(L, κ) to be W[1]-hard if there is a W[1]-hard problem (L', κ') that FPT-reduces to (L, κ). We define (L, κ) to be coW[1]-hard if its complement is W[1]-hard. It is generally conjectured that FPT $\neq$ W[1] and thus in particular W[1]-hard problems and coW[1]-hard problems are not in FPT. We will take the hardness results for problems (L', κ') from the literature. One important such problem is the homomorphism problem p-HOM($\mathcal{C}$) for a class $\mathcal{C}$ of sets of atoms. Its input is one set $\mathbf{A} \in \mathcal{C}$ of atoms and another set $\mathbf{B}$ of atoms, and the question is whether there exists a homomorphism $h\colon \mathbf{A} \to \mathbf{B}$. The parameter is the size of $\mathbf{A}$.

▶ **Theorem 7** ([12]). *Let $\mathcal{C}$ be a decidable class of simple sets of atoms. Then* p-HOM($\mathcal{C}$) *is in* FPT *if there exists some constant c such that the treewidth of each set in $\mathcal{C}$ is bounded by c, and* W[1]*-hard otherwise.*

3 Tractability Conditions

In this section we will introduce our tractability criteria which we tailor towards simple wdPTs to simplify the presentation. While, as mentioned, our criteria also apply to arbitrary wdPTs, they are not optimal in this case. In fact, in the extended version we show generalizations of these conditions that, for general wdPTs, describe more tractable classes.

We start with the definition of what we consider as *simple* pattern trees. Basically, in a simple pattern tree, no relation symbol is allowed to occur more than once.

▶ **Definition 8** (Simple PTs). *A PT $p = (T, \lambda, \mathcal{X})$ over σ is a* simple pattern tree *if $\lambda(T)$ is simple and $\lambda(t) \cap \lambda(t') = \emptyset$ for all $t, t' \in T$ with $t \neq t'$.*

Our overall idea of solving p-Eval($\mathcal{P}$) is as follows: given a wdPT p, a database $\mathbf{D}$, and a mapping μ, construct a set of CQs q with free variables $\vec{x}$ and associated databases $\mathbf{D}'$ such that $\mu \in p(\mathbf{D})$ if and only if for at least one of these CQs q the tuple $\mu(\vec{x})$ is in $q(\mathbf{D}')$. We give two tractability criteria ensuring that the algorithm is in FPT. Intuitively, one condition guarantees that deciding $\mu(\vec{x}) \in q(\mathbf{D}')$ is in PTIME, while both conditions in combination guarantee that $\mathbf{D}'$ can be computed efficiently.

We will state the tractability conditions with respect to a class $\mathcal{P}$ of wdPTs. So in the remainder of this section let $\mathcal{P}$ be an arbitrary but fixed class of wdPTs.

We start with some additional notation and results. One effect introduced by projections is that some nodes of wdPTs may not be relevant for the results in the following sense as already observed in [16].

▶ **Definition 9** (Relevant Nodes). *Let $p = (T, \lambda, \mathcal{X})$ be a wdPT. A node $t \in T$ is* relevant *if there exists a database $\mathbf{D}$ such that $p(\mathbf{D}) \neq p'(\mathbf{D})$ where p' is constructed from p by removing from T the subtree rooted in t. We use* $\mathsf{relv}(T)$ *to denote the set of relevant nodes in T.*

In [16], the authors introduced a normal form excluding non-relevant nodes. Here, in order to make our results more explicit, we do not follow this approach but allow wdPTs to contain non-relevant nodes. Luckily, it follows from [16] that these nodes can be easily detected.

▶ **Proposition 10.** *Let $p = (T, \lambda, \mathcal{X})$ be a wdPT. Then a node $t \in T$ is* relevant *if and only if $\mathsf{fvar}(T') \setminus \mathsf{fvar}(\hat{t}) \neq \emptyset$, where T' is the subtree of T rooted in t and $\hat{t}$ is the parent node of t.*

In what is to follow, the variables that are shared between a node and its parent node will be of crucial importance. To this end, we make the following definition.

▶ **Definition 11** (Interface $\mathcal{I}(t)$ of a Node). *Let $(T, \lambda, \mathcal{X})$ be a wdPT, $t \in T$ (but not the root node), and $\hat{t}$ the parent node of t. The* interface $\mathcal{I}(t)$ *of t is the set $\mathcal{I}(t) = \mathsf{var}(t) \cap \mathsf{var}(\hat{t})$. The interface of the root node r is $\mathcal{I}(r) = \emptyset$.*

It was already remarked e.g. in [3] and [15] that restrictions on the number of variables shared between different sets of nodes can be used to define tractable classes. The above definition however differs slightly from the notion of interfaces in these works. E.g., in [15], the interface of a node describes the set of variables shared between the node and any of its neighbors, while here it is restricted to the variables shared with its parent node.

Restrictions on the size of node interfaces turn out to be quite coarse, and we provide more fine grained tractability criteria here. To this end, we recall the notion of an S-component from [7]: let $G = (V, E)$ be a graph, and $S \subseteq V$. Then let $\mathcal{K}$ be the set of connected components of $G[V \setminus S]$, and for each $K \in \mathcal{K}$, let $S_K \subseteq S$ be the set of nodes in S that have (in G) an edge to some node in K, i.e., $S_K = \{v \in S \mid \{v, v'\} \in E \text{ for some } v' \in K\}$. The set of S-components of G now is the set $\{G[K \cup S_K] \mid K \in \mathcal{K}\}$.

For a set S of variables, the notion of S-components extends to sets of atoms in the obvious way via the Gaifman graph. We will thus talk about S-components of sets of atoms.

▶ **Definition 12** ([15] Node Components). *Let $p = (T, \lambda, \mathcal{X})$ be a wdPT and $t \in T$. The set of* node components $\mathcal{NC}_t$ of t *is a set of set of atoms, defined as the union of:*

1. *The set $\{\{\tau\} \mid \tau \in \lambda(t)$ and $\mathsf{var}(\tau) \subseteq \mathcal{I}(t)\}$ consisting of singleton sets for every atom $\tau \in \lambda(t)$ which contains only "interface variables", i.e., variables from $\mathcal{I}(t)$.*
2. *The set of all $\mathcal{I}(t)$-components of $\lambda(t)$.*

In the following, node components of *type (1)* are those containing single atoms, while node components of *type (2)* are sets of possibly several atoms.

▶ **Example 13.** Recall the wdPT p from Example 6 and consider the node t_2. It contains the following node components: each atom $clique_{ij}(y_i, y_j)$ forms a node component of type (1) since all variables y_i occur also in r. In addition, there are two node components of type (2): the sets $\{d_1(x_1, u_1), d_2(u_1, u_2), d_3(u_2, y_1)\}$ and $\{d_4(u_3, y_1)\}$. Observe that while all atoms d_i $(1 \leq i \leq 4)$ are within the same connected component of the Gaifman graph, they are not in the same node component, since y_1 separates the connected component into two parts.

To understand why node components are essential for our results, recall that solutions to wdPTs must be maximal, i.e., they map some subtree into the database, but cannot be extended to map some child node of this subtree into the database as well. But such an extension to a node does not exist if and only if the mapping cannot be extended to one of the node components. Thus instead of testing extensions to the complete node at once (which might be intractable), we test the maximality of a mapping independently for each node component (which might be tractable). This is possible because for all variables shared between any two node components, the values are already determined by the mapping to be extended. Extensions to different node components are thus independent of each other.

For node components, we are in particular interested in the contained interface variables.

▶ **Definition 14** (Interface of a Node Component). *For a wdPT $(T, \lambda, \mathcal{X})$ and a node $t \in T$, the* interface of a node component $\mathbf{S} \in \mathcal{NC}_t$ *is $\mathcal{I}_t(\mathbf{S}) = (\mathcal{I}(t) \cap \mathsf{var}(\mathbf{S}))$, and the* existential *interface is $\mathcal{I}_t^{\exists}(\mathbf{S}) = \mathcal{I}_t(\mathbf{S}) \setminus \mathcal{X}$.*

We are now ready to formulate our first tractability condition.

Tractability condition (a): There is a constant c, such that for each $p = (T, \lambda, \mathcal{X}) \in \mathcal{P}$, the treewidth of $\mathbf{S} \setminus \mathcal{I}_t(\mathbf{S})$ is bounded by c for all $t \in \mathsf{relv}(T)$ (with $t \neq r$) and all $\mathbf{S} \in \mathcal{NC}_t$.

Intuitively, condition (a) guarantees that for each node component $\mathbf{S}$, given some mapping on the variables in its interface, one can decide in polynomial time whether this mapping can be extended to map all atoms in the node component into a given database. This is because once a mapping on $\mathcal{I}_t(\mathbf{S})$ is given, these variables can be treated as constants.

▶ **Example 15.** To demonstrate the situation after introducing condition (a), let p_k be the wdPT p from Example 6 parameterized by k, let $\mathcal{P} = \{p_k \mid k \geq 2\}$, and let $T' = T[\{r, t_1, t_2\}]$. Assume, for some $k \geq 2$, in order to test if some mapping is a solution to p_k, we would like to verify whether some mapping μ' with $\mathsf{dom}(\mu') = \mathsf{var}(T')$ is a maximal pp-solution. Deciding whether it is a pp-solution is easy, and because $\mathcal{P}$ satisfies tractability condition (a), testing if there exists in both, t_3 and t_4 a node component to which μ' cannot be extended is feasible in polynomial time as well. In fact, $\mathcal{NC}_{t_3} = \{\{b_3(x_3, u_1)\}, \{b_4(u_1, x_4)\}, \{d_5(x_1, u_3, u_1)\}\}$ and $\mathcal{NC}_{t_4} = \{\{b_5(y_2, x_5), c_3(x_5, u_2)\}, \{c_4(u_2, z_2)\}\}$ (this holds for every $p_k \in \mathcal{P}$). However, there may exist an exponential number of pp-solutions on T' (T' contains $k + 4$ existential variables). Thus testing one mapping on $\mathsf{var}(T')$ after the other is not feasible.

One way to overcome the problem sketched in the example is to not have two separate tests for being a pp-solution and being maximal. To combine these tests, we first compute for each node component the set of mappings on its interface variables that do not extend to the component, and then require pp-solutions to be consistent with these mappings.

It turns out that this idea can be encoded as an evaluation problem for CQs. One important step in this encoding is to introduce new relations, one for each node component, that store those mappings on the interface variables that cannot be extended to the component. In order for the resulting CQ evaluation problem to be in polynomial time, we require two properties. First, the resulting CQs must be from some tractable fragment of CQ evaluation, and, second, the size of the newly added relations must be at most polynomial. One way to achieve this second goal is to restrict the arity of these relations. Tractability for the CQ evaluation problem holds exactly if the class of resulting CQs is of bounded treewidth. As it turns out, this restriction also implies a bound on the arity of the new relations, and thus represents our second tractability condition.

To formalize the construction, we introduce the notion of a *component interface atom*. For a wdPT $(T, \lambda, \mathcal{X})$, a subtree T' of T, a node $t \in ch(T')$, and node component $\mathbf{S} \in \mathcal{NC}_t$, let the interface atom be an atom $R(\vec{v})$ where $\vec{v}$ contains the variables in $\mathcal{I}_t^{\exists}(\mathbf{S})$ and R is a fresh relation symbol. For a node component $\mathbf{S}$, we use $\mathsf{cia}(\mathbf{S})$ to refer to the corresponding interface atom $R(\vec{v})$. Observe that these definitions imply $\mathsf{cia}(\mathbf{S}) = R()$ in case $\mathcal{I}_t^{\exists}(\mathbf{S}) = \emptyset$.

The intuition for $\mathsf{cia}(\mathbf{S})$ is that for each node component, we get one atom that covers exactly the variables in $\mathcal{I}_t^{\exists}(\mathbf{S})$. The free variables in the interface can be excluded from the considerations since a fixed value is provided for them as part of the input.

▶ **Example 16.** Recall again the wdPT from Example 6 and consider the node t_1. It contains two node components: $\mathbf{S}_1 = \{b_2(x_2)\}$ and $\mathbf{S}_2 = \{c_1(y_1, z_1), c_2(y_2, z_1)\}$. Observe that whether $\mathbf{S}_1$ can be mapped into some database $\mathbf{D}$ is completely independent of the interface variables. Thus $\mathsf{cia}(\mathbf{S}_1) = R_{\mathbf{S}_1}()$. However, for $\mathbf{S}_2$ the values of y_1 and y_2 influence whether $\mathbf{S}_2$ may be mapped. Thus we get $\mathsf{cia}(\mathbf{S}_2) = R_{\mathbf{S}_2}(y_1, y_2)$. In case of the node component $\{d_5(x_1, u_3, u_1)\}$ of t_3, observe that we can assume some fixed value $\mu(x_1)$, and thus can reduce the atom $d_5(\mu(x_1), u_3, u_1)$ according to this value, and get as interface atom $d_5^{x_1=\mu(x_1)}(u_3, u_1)$.

Since we are looking for *one* pp-solution that cannot be extended to *any* child node, combining the two tests as sketched means that we must test all children simultaneously instead of individually. However, since each CQ tests only one node component for each child, we need one CQ for each possible combination, leading to our second tractability condition.

Tractability condition (b): There is a constant c such that for every well-designed pattern tree $p = ((T, r), \lambda, \mathcal{X}) \in \mathcal{P}$ and every subtree T' of T containing r, the treewidth of $(\lambda(T') \cup \bigcup_{i=1}^{n} \{\mathsf{cia}(\mathbf{S}_i)\}) \setminus \mathsf{fvar}(T')$ is bounded by c for every $(\mathbf{S}_1, \ldots, \mathbf{S}_n) \in \mathcal{NC}_{t_1} \times \cdots \times \mathcal{NC}_{t_n}$ where $\{t_1, \ldots, t_n\} = ch(T') \cap \mathsf{relv}(T)$.

The following example breaks down condition (b) to demonstrate its intuition.

▶ **Example 17.** Recall the setting in Example 15, as well as the database instance $\mathbf{D}$ and the mappings μ_2 and μ_2' from Example 6. Assume that, in order to show that $\mu_2 \in p_k(\mathbf{D})$ for any $k \geq 2$, we are looking for a pp-solution for T' that does not extend neither to the node component $\mathbf{S}_1 = \{b_3(x_3, u_1)\}$ of t_3 (and thus not to $\lambda(t_3)$), nor to the node component $\mathbf{S}_2 = \{b_5(y_2, x_5), c_3(x_5, u_2)\}$ of t_4 (and thus not to $\lambda(t_4)$). The idea is to construct a CQ $Ans(x_1, x_2) \leftarrow \lambda(T') \cup \{R_1(u_1), R_2(y_2, u_2)\}$, where $R_1(u_1)$ and $R_2(y_2, u_2)$ are the component interface atoms for $\mathbf{S}_1$ and $\mathbf{S}_2$, respectively. Observe that this query has exactly the structure described in condition (b), illustrating the motivation for the definition of this condition. Also, because of the $clique_{ij}$-atoms, $\mathcal{P}$ does not satisfy tractability condition (b).

The query is evaluated over the instance $\mathbf{D}$ extended by relations for R_1 and R_2. As mentioned, these relations contain all values that cannot be extended to map $\mathbf{S}_1$ and $\mathbf{S}_2$ into $\mathbf{D}$, respectively. For $\mathbf{S}_1$, we get $\{R_1(2)\}$ (since $b_3(1,1) \in \mathbf{D}$, thus a mapping assigning 1 to u_1 could be extended to map $\mathbf{S}_1$ into $\mathbf{D}$), and for $\mathbf{S}_2$ we get $\{R_2(1,2), R_2(2,1), R_2(2,2)\}$.

Now the mapping μ_2' witnesses the fact that $(1,1) \in q(\mathbf{D})$, and thus μ_2' is a maximal pp-solution also witnessing $\mu_2 \in p_k(\mathbf{D})$.

The main result of this paper is that the conditions (a) and (b) characterize exactly the classes of simple wdPTs which can be evaluated efficiently.

▶ **Theorem 18.** *Assume that* $\mathsf{FPT} \neq \mathsf{W}[1]$*, and let* $\mathcal{P}$ *be a decidable class of simple wdPTs of bounded arity. Then the following statements are equivalent.*

1. *The tractability conditions (a) and (b) hold for* $\mathcal{P}$*.*
2. p-Eval($\mathcal{P}$) *is in* FPT*.*

We will show the upper bound of Theorem 18 in Section 4, where we describe how the different ideas described so far can be combined to an FPT algorithm, while the lower bounds will be shown in Section 5.

But before we turn to the proof of Theorem 18, let us interpret the result in the setting without projections to better understand the influence of projection. First note that in that case, by Definition 14, we have $\mathcal{I}_t^{\exists}(\mathbf{S}) = \emptyset$ for every $t \in T$ and every $\mathbf{S} \in \mathcal{NC}_t$. Thus all atoms $\mathsf{cia}(\mathbf{S})$ (for any node component $\mathbf{S}$) are of arity 0 as are all atoms in $(\lambda(T') \cup \bigcup_{i=1}^{n} \{\mathsf{cia}(\mathbf{S}_i)\}) \setminus \mathsf{fvar}(T')$. Tractability condition (b) is therefore void in this setting, leaving only (a) as a useful condition in the projection free case. This immediately implies the following corollary.

▶ **Corollary 19.** *Assume that* $\mathsf{FPT} \neq \mathsf{W}[1]$*, and let* $\mathcal{P}$ *be a decidable class of simple wdPTs of bounded arity without projections. Then* p-Eval($\mathcal{P}$) *is in* FPT *if and only if tractability condition (a) holds for* $\mathcal{P}$*.*

We remark that Corollary 19 could also be inferred as a special case of the main result of [22]. Stating the corollary explicitly here lets us better understand the role of projection for our problem: in fact, the role of condition (a) is essentially to deal with the complexity that we already have without projection, while condition (b) is necessary to deal with the additional source of hardness that is introduced by projections and does not appear without them.

Since it will simplify the discussion in the upcoming sections, we conclude the section by explicitly working out the third tractability condition already mentioned above in the discussion towards tractability condition (b). As described there, at some point we extend a given database by relations for the atoms $\mathsf{cia}(\mathbf{S})$ that contain for the corresponding node component all mappings on its existential interface that cannot be extended to a mapping

Algorithm 1 EvalFPT($p = ((T, r), \lambda, \mathcal{X})$, $\mathbf{D}$, μ).

1: $T = T[\mathsf{relv}(T)]$ ▷ Remove all nodes from T that are not relevant
2: **for all** subtrees T' of T with $\mathsf{fvar}(T') = \mathsf{dom}(\mu)$ and $r \in T'$ **do**
3: Let $\{t_1, \ldots, t_n\} = ch(T')$
4: **for all** $(\mathbf{S}_1, \ldots, \mathbf{S}_n) \in \mathcal{NC}_{t_1} \times \cdots \times \mathcal{NC}_{t_n}$ **do**
5: $q = \text{“}Ans(\vec{x}) \leftarrow \lambda(T') \cup \{\mathsf{cia}(\mathbf{S}_1), \ldots, \mathsf{cia}(\mathbf{S}_n)\}\text{”}$ ▷ Let $\vec{x}$ contain all $x \in \mathsf{fvar}(T')$
6: $\mathbf{D}' = \mathbf{D} \cup \bigcup_{i=1}^{n} \{R_i(\nu(\vec{v}_i)) \mid \nu \in stop(\mathbf{S}_i, \mathbf{D})\}$ ▷ Assume $\mathsf{cia}(\mathbf{S}_i) = R_i(\vec{v}_i)$
7: **if** $\mu(\vec{x}) \in q(\mathbf{D}')$ **then** EXIT(YES)
8: EXIT(NO)

on the whole node component. To guarantee that all these relations are of polynomial size, we restrict the number of variables in the existential interfaces of the node components by some constant c. We formalize this notion in terms of a suitable width measure.

▶ **Definition 20** (Component Width). *Let $p = (T, \lambda, \mathcal{X})$ be a wdPT, $t \in T$, and $\mathbf{S} \in \mathcal{NC}_t$. The* width *of the node component $\mathbf{S}$ is $|\mathcal{I}_t^{\exists}(\mathbf{S})|$. For a node $t \in T$, the* component width *of t is the maximum width over all node components $\mathbf{S}$ of t. The component width of p is the maximum component width over all $t \in \mathsf{relv}(T)$.*

By the definition of the treewidth of a set of atoms – specifically by the fact that in the Gaifman graph all variables occurring together in an atom form a clique – and the fact that for the existential interface of each node component its variables occur together in some $\mathsf{cia}(\mathbf{S})$-atom, the number of variables in any existential interface is bound by the treewidth of the CQs defined in condition (b). Thus, we get the following corollary.

▶ **Corollary 21.** *Let $\mathcal{P}$ be a class of wdPTs that satisfies tractability condition (b) for some constant c. Then, for every $p \in \mathcal{P}$, the component width of p is at most $c + 1$.*

4 The FPT algorithm

Having defined the tractability conditions, we now show how they are used in the FPT-algorithm for p-EVAL($\mathcal{P}$) outlined in Algorithm 1.

The missing ingredient of Algorithm 1 that we have not yet introduced is $stop(\mathbf{S}, \mathbf{D})$ for a node component $\mathbf{S}$ and a database $\mathbf{D}$ which we explain now. Recall that we said earlier that the intention of the node components is to ensure a mapping to be maximal not by testing for extensions to the complete node, but to do these tests for smaller, independent units.

The idea how to realize this is to store in $\mathbf{D}'$ for each node component $\mathbf{S}$ those variable assignments ν to the variables in its existential interface such that there exists no extension $\nu' \colon \mathbf{S} \to \mathbf{D}$ of $\nu \cup \mu$. These are the values stored in $stop(\mathbf{S}_i, \mathbf{D})$.

In more detail, for a wdPT $((T, r), \lambda, \mathcal{X})$, a subtree T' of T containing r, a child node $t \in ch(T')$, node component $\mathbf{S} \in \mathcal{NC}_t$, a database $\mathbf{D}$, and a mapping $\mu \colon \mathsf{fvar}(T') \to \mathsf{dom}(\mathbf{D})$, consider the set $extend(\mathbf{S}, \mathbf{D}) = \{\eta \colon \mathcal{I}_t^{\exists}(\mathbf{S}) \to \mathsf{dom}(\mathbf{D}) \mid \text{there exists } \eta' \colon \mathbf{S} \to \mathbf{D} \text{ extending } \eta \text{ and } \mu|_{\mathsf{var}(\mathbf{S})}\}$. So $extend$ contains exactly those mappings on $\mathcal{I}_t^{\exists}(\mathbf{S})$ that *can* be extended in a way that is compatible with μ and maps $\mathbf{S}$ into $\mathbf{D}$. We thus set $stop(\mathbf{S}, \mathbf{D}) = \{\nu \colon \mathcal{I}_t^{\exists}(\mathbf{S}) \to \mathsf{dom}(\mathbf{D}) \mid \nu \notin extend(\mathbf{S}, \mathbf{D})\}$.

With this in place, we describe the idea of Algorithm 1. Recall that, given μ, we have to find a mapping μ' extending μ that is (1) a pp-solution, and (2) maximal. Because of the existential variables, there may be exponentially many subtrees T' of T containing r with $\mathsf{fvar}(T') = \mathsf{dom}(\mu)$, each being a potential candidate for witnessing (1) and (2). After removing all irrelevant nodes in line 1 (they might make evaluation unnecessarily hard), we thus check each of these subtrees (line 2).

If the required mapping μ' exists, then, as discussed earlier, for each child node of T' there exists at least one node component to which μ' cannot be extended. Not knowing which node components these are, the algorithm iterates over all possible combinations (line 4). In lines 5–7, the algorithm now checks whether there exists an extension of μ that maps all of $\lambda(T')$ into $\mathbf{D}$ (ensured by adding $\lambda(T')$ to q), but none of the node components $\mathbf{S}_1, \ldots, \mathbf{S}_n$. The latter property is equivalent to asking that μ' must assign a value to the existential interface variables of each $\mathbf{S}_i$ that cannot be extended. This is guaranteed by adding the atoms $\mathsf{cia}(\mathbf{S}_i)$ to q and providing in $\mathbf{D}'$ exactly the values from $\mathit{stop}(\mathbf{S}_i, \mathbf{D})$.

In order to see that this indeed gives an FPT algorithm in case tractability conditions (a) and (b) are satisfied, note that condition (b) ensures that the arity of each of the new relations for the atoms $\mathsf{cia}(\mathbf{S})$ is at most $c + 1$ (cf. Corollary 21). Thus the size of these relations (and thus the number of possible mappings in $\mathit{stop}(\mathbf{S}, \mathbf{D})$) is at most $|\mathsf{dom}(\mathbf{D})|^{(c+1)}$. Next, condition (a) ensures that for each mapping $\nu \colon \mathcal{I}_t^{\exists}(\mathbf{S}) \to \mathsf{dom}(\mathbf{D})$ deciding membership in $\mathit{stop}(\mathbf{S}, \mathbf{D})$ is in PTIME. Observe that the variables in $\mathcal{I}_t(\mathbf{S})$ are not considered in the computation of the treewidth since a fixed value is provided for them, thus they can be treated as constants. Finally, condition (b) also ensures that the test in line 7 is feasible in polynomial time. Again, since a fixed value is provided for the domain of μ, these variables can be treated as constants.

We note that the algorithm is an extension and refinement of the FPT algorithm presented in [15]. An inspection of [15] reveals that the conditions provided there imply our tractability conditions (a) and (b), but there is no implication in the other direction. In fact, our conditions explicitly describe the crucial properties of their restrictions that make the problem to be in FPT. From Algorithm 1 we thus derive the following result.

▶ **Theorem 22.** *Let $\mathcal{P}$ be a decidable class of wdPTs. If the tractability conditions (a) and (b) hold for $\mathcal{P}$, then* p-Eval($\mathcal{P}$) *can be solved in* FPT.

The correctness of the algorithm follows immediately from the previous discussion. For the runtime, in addition to what was already discussed, the number of loop-iterations in lines 2 and 4 is bounded by a function in the size of p, which is the parameter for the problem.

5 Optimality of the Tractability Conditions

We now show that both tractability criteria are necessary, and thus finish the proof of Theorem 18. We provide individual results for both conditions. In addition, we show that also the bound on the component width is necessary (and not just a side effect), which will turn out to be a useful result for proving that tractability condition (b) is necessary.

▶ **Lemma 23.** *Let $\mathcal{P}$ be a decidable class of simple wdPTs of bounded arity such that tractability condition (a) is not satisfied. Then* p-Eval($\mathcal{P}$) *is* coW[1]*-hard.*

Proof. For a wdPT $p \in \mathcal{P}$, let the *relevant components set* $\mathsf{rcs}(p)$ contain all the sets $\mathbf{S} \setminus \mathcal{I}_t(\mathbf{S})$ as defined in tractability condition (a). Moreover, let $\mathsf{rcs}(\mathcal{P}) = \bigcup_{p \in \mathcal{P}} \mathsf{rcs}(p)$. We will – by an FPT-reduction – reduce p-HOM($\mathsf{rcs}(\mathcal{P})$) to the complement of p-Eval($\mathcal{P}$). The result then follows from Theorem 7, as $\mathsf{rcs}(\mathcal{P})$ does not have bounded treewidth by assumption.

Consider an instance $\mathbf{E}, \mathbf{F}$ of p-HOM($\mathsf{rcs}(\mathcal{P})$). In a first step, find $p = ((T, r), \lambda, \mathcal{X}) \in \mathcal{P}$, a node $t \in \mathsf{relv}(T)$ such that $t \neq r$, and a node component $\mathbf{S} \in \mathcal{NC}_t$ such that $\mathbf{E} = \mathbf{S} \setminus \mathcal{I}_t(\mathbf{S})$. They exist by assumption and, since $\mathcal{P}$ is decidable, can be computed.

Since t is relevant, either for $t' = t$ or some descendant t' of t we have $\mathsf{fvar}(t') \setminus \mathsf{fvar}(\mathsf{branch}(t')) \neq \emptyset$. Among all possible candidates, pick some t' at a minimal distance to t.

We next define a database $\mathbf{D}$ over the set of relation symbols in p. For the description of $\mathbf{D}$, for all relation symbols R occurring in any atom $R(\vec{v}) \in \lambda(T)$, we will assume that $\vec{v}$ contains only variables, i.e. elements from $\mathcal{V}ar$. We implicitly assume that for all positions where $\vec{v}$ contains a constant, all atoms in $R^{\mathbf{D}}$ contain the same constant as in $\vec{v}$. Recall that we deal with simple wdPTs, thus each relation symbol occurs at most once within $\lambda(T)$. In the following, let $d \in \mathcal{C}onst$ be some fresh value not occurring in $\mathsf{dom}(\mathbf{F})$.

For each relation symbol R mentioned outside of $\lambda(\mathsf{cbranch}(t'))$, let $R^{\mathbf{D}} = \emptyset$.

For each relation symbol R mentioned in $\lambda(\mathsf{branch}(t))$, let $R^{\mathbf{D}} = \{R(d, \ldots, d)\}$.

For each relation symbol R mentioned in $\lambda(\mathsf{cbranch}(t') \setminus \mathsf{branch}(t)) \setminus \mathbf{S}$, let k be the arity of R and $R^{\mathbf{D}} = \{R(\vec{v}) \mid \vec{v} \in (\mathsf{dom}(\mathbf{F}) \cup \{d\})^k\}$.

For each relation symbol R mentioned in $\mathbf{S}$, observe that there exists a relation symbol R' in $\mathbf{E}$ that was derived from R when computing $\mathbf{S} \setminus \mathcal{I}_t(\mathbf{S})$. The idea is now to use $R^{\mathbf{D}}$ to simulate $R'^{\mathbf{F}}$ by padding the additional fields with d. Thus let k be the arity of R, let m be the arity of R', let $\{i_1, \ldots, i_\ell\} \subseteq \{1, \ldots, k\}$ be those positions of R containing values from $\mathcal{I}_t(\mathbf{S})$, and $\{o_1, \ldots, o_m\} = \{1, \ldots, k\} \setminus \{i_1, \ldots, i_\ell\}$ those positions of R that contain values from $\mathsf{var}(\mathbf{S}) \setminus \mathcal{I}_t(\mathbf{S})$. Then, for every $R'(a_{o_1}, \ldots, a_{o_m}) \in \mathbf{F}$, let $R^{\mathbf{D}}$ contain the atom $R(b_1, \ldots, b_k)$ where, for $1 \leq \alpha \leq k$, we have $b_\alpha = a_{o_j}$ if $\alpha = o_j$ for some $1 \leq j \leq m$ and $b_\alpha = d$ if $\alpha = i_j$ for some $1 \leq j \leq \ell$. This completes the definition of $\mathbf{D}$.

Finally, we define the mapping μ as $\mu(x) = d$ for all $x \in \mathsf{fvar}(\mathsf{branch}(t))$.

With the description of the reduction complete, we claim that $\mu \in p(\mathbf{D})$ if and only if there is no homomorphism from $\mathbf{E}$ to $\mathbf{F}$. We prove this property in two steps. First, we show that $\mu \in p(\mathbf{D})$ only depends on whether μ can be extended to t or not. After this we show that such an extension of μ exists if and only if there is a homomorphism $h\colon \mathbf{E} \to \mathbf{F}$.

First, observe that the only possible extension μ' of μ such that $\mu'(\tau) \in \mathbf{D}$ for every $\tau \in \lambda(\mathsf{branch}(t))$ is μ' mapping every variable in $\mathsf{var}(\mathsf{branch}(t))$ to d. Moreover, for all nodes $t'' \neq t$ in $ch(\mathsf{branch}(t))$ the mapping μ' cannot be extended to $\lambda(t'')$, since for all relation symbols R mentioned in $\lambda(t'')$ we have $R^{\mathbf{D}} = \emptyset$. Thus μ' is a pp-solution, and is a maximal pp-solution if and only if there exists no extension μ'' of μ' with $\mu''(\tau) \in \mathbf{D}$ for all $\tau \in \lambda(t)$.

Clearly, if μ' is a maximal pp-solution, then $\mu \in p(\mathbf{D})$. To see that $\mu \notin p(\mathbf{D})$ if μ' is not a maximal pp-solution, assume that there exists the above mentioned extension μ'' of μ'. Then μ'' can be obviously extended to μ''' with $\mu'''(\tau) \in \mathbf{D}$ for all $\tau \in \mathsf{cbranch}(t')$ since for all atoms on $(\mathsf{cbranch}(t') \setminus \mathsf{cbranch}(t)) \cup \{t'\}$, every possible atom over $\mathsf{dom}(\mathbf{D})$ is contained in $\mathbf{D}$. Since $\mathsf{dom}(\mu''')$ contains at least one free variable not in $\mathsf{dom}(\mu')$, this shows $\mu \notin p(\mathbf{D})$.

It thus remains to show that the extension μ'' of μ' exists if and only if there is a homomorphism $h\colon \mathbf{E} \to \mathbf{F}$. To see that this is the case, observe that by construction every such homomorphism h in combination with μ' gives a mapping from $\mathbf{S}$ into $\mathbf{D}$, and vice versa, every mapping $\mu\colon \mathbf{S} \to \mathbf{D}$ restricted to $\mathsf{dom}(\mathbf{E})$ gives the desired homomorphism. For the remaining atoms in $\lambda(t) \setminus \mathbf{S}$, observe that every possible mapping sends them into $\mathbf{D}$, since $\mathbf{D}$ again contains every possible atom for these relations. ◀

To simplify the proof that tractability condition (b) is necessary, we first show that having bounded component width is a necessary condition on its own.

▶ **Lemma 24.** *Let $\mathcal{P}$ be a decidable class of simple wdPTs of bounded arity. If there does not exist some constant c such that for every $p \in \mathcal{P}$ the component width is bounded by c, then* p-Eval($\mathcal{P}$) *is* coW[1]*-hard.*

Proof. The proof is an FPT-reduction of the problem of model checking FO sentences ϕ_k of the form $\phi_k = \forall x_1 \ldots \forall x_k \exists y \bigwedge_{i=1}^{k} E_i(x_i, y)$. Model checking for this class of sentences, parameterized by their size, is W[1]-hard [5]. Thus consider a formula ϕ_k and a database $\mathbf{E}$.

First, compute a wdPT $p = (T, \lambda, \mathcal{X}) \in \mathcal{P}$ with a component width of at least k. W.l.o.g. we assume that p contains only binary atoms: Since we assume a bounded arity, binary atoms can be easily encoded into atoms of higher arity. Consider the relevant node $t \in T$ and a node component $\mathbf{S} \in \mathcal{NC}_t$ such that the component width of $\mathbf{S}$ is at least k. Since we assume relations to be of some bounded arity, $\mathbf{S}$ cannot be of type (1) (Definition 12). W.l.o.g. we thus assume that $\mathbf{S}$ is of type (2).

Since t is relevant, either for $t' = t$ or some descendant t' of t we have $\mathsf{fvar}(t') \setminus \mathsf{fvar}(\mathsf{branch}(t')) \neq \emptyset$. Choose one such t' at a minimal distance to t.

For the description of the atoms in $\mathbf{D}$ for a relation symbol R occurring in an atom $R(\vec{v}) \in \lambda(T)$, we will assume that $\vec{v}$ contains only variables, i.e. elements from $\mathcal{V}ar$. We implicitly assume that for all positions where $\vec{v}$ contains a constant, all the atoms in $R^{\mathbf{D}}$ contain the same constant as in $\vec{v}$. Recall that we are dealing with simple wdPTs, thus each relation symbol R occurs at most once within $\lambda(T)$.

For each relation symbol R mentioned outside of $\lambda(\mathsf{cbranch}(t'))$, let $R^{\mathbf{D}} = \emptyset$.

For each relation symbol R mentioned in $\lambda(\mathsf{cbranch}(t')) \setminus \mathbf{S}$, let ℓ be the arity of R and $R^{\mathbf{D}} = \{R(\vec{v}) \mid \vec{v} \in \mathsf{dom}(\mathbf{E})^{\ell}\}$.

For the relation symbols R mentioned in $\mathbf{S}$, proceed as follows. Choose k interface variables $v_1, \ldots, v_k \in \mathcal{I}_t(\mathbf{S})$. Let $L = \mathsf{var}(\mathbf{S}) \setminus \mathsf{var}(\mathsf{branch}(t))$ be the "local variables" of $\mathbf{S}$. Observe that $\mathbf{S}$ being a node component of type (2) implies $L \neq \emptyset$. This is because of $\mathsf{var}(\mathbf{S}) \cap \mathsf{var}(\mathsf{branch}(t)) = \mathcal{I}_t(\mathbf{S})$, and the fact that to be a node component of type (2), in the Gaifman graph, all variables in $\mathcal{I}_t(\mathbf{S})$ must be connected to some variable from the node component that is not in $\mathcal{I}_t(\mathbf{S})$. Thus there must exist at least one "local variable". By the same reasoning, for each of the variables v_i, there must exist at least one atom $R_i(v_i, z_i)$ or $R_i(z_i, v_i)$ for some $z_i \in L$. We will assume $R_i(v_i, z_i)$ in the following, the other case is analogous. Now for each v_i, fix one such atom. Based on this, we define the following atoms to be contained in $\mathbf{D}$:

For each of the selected atoms $R_i(v_i, z_i)$, let $R_i^{\mathbf{D}} = E_i^{\mathbf{E}}$, i.e., we let R_i simulate exactly E_i. For every atom $R(z, z') \in \mathbf{S}$ such that $z, z' \in L$, define $R^{\mathbf{D}} = \{R(d, d) \mid d \in \mathsf{dom}(\mathbf{E})\}$. For the remaining atoms $R(z, z') \in \mathbf{S}$, define $R^{\mathbf{D}} = \{R(a, b) \mid a, b \in \mathsf{dom}(\mathbf{E})\}$.

Finally, μ is an arbitrary mapping $\mathsf{fvar}(\mathsf{branch}(t)) \to \mathsf{dom}(\mathbf{E})$.

It now follows by the same arguments as in the proof of Lemma 23 that we have $\mu \notin p(\mathbf{D})$ if and only if for every extension μ' of μ to $\mathsf{var}(\mathsf{branch}(t))$, there exists an extension ν of μ' such that $\nu(\tau) \in \mathbf{D}$ for all $\tau \in \mathbf{S}$.

To complete this proof, we thus only need to show that such an extension exists if and only if ϕ_k is satisfied. First, assume that ϕ_k is satisfied. Then, for all $z \in L$, define $\nu(z)$ to be the value of y in ϕ_k. This clearly maps $\mathbf{S}$ into $\mathbf{D}$. Next, assume that ϕ_k is not satisfied. Then there exists some assignment to $x_1, \ldots, x_k$ such that no suitable value for y exists. Then for the mapping μ' assigning exactly those values to the selected interface variables $v_1, \ldots, v_k$, there exists no extension of μ' to $\mathbf{S}$. This is because L defines a connected component in the Gaifman graph and because the definition of $\mathbf{D}$ forces all variables in L that occur together in some atom in $\mathbf{S}$ to be mapped to the same value by μ'. Thus μ' has to map all "local variables" in $\mathbf{S}$ to the same value, which would provide a suitable value for y, leading to a contradiction. This concludes the proof. ◀

▶ **Lemma 25.** *Let $\mathcal{P}$ be a decidable class of simple wdPTs of bounded arity such that tractability condition (b) is not satisfied. Then* p-Eval($\mathcal{P}$) *is either* coW[1]- *or* W[1]*-hard.*

Proof. First, assume that there exist some constant that is, for all $p \in \mathcal{P}$, an upper bound on the component width. Otherwise, p-Eval($\mathcal{P}$) is coW[1]-hard by Lemma 24. In particular, we may thus assume that all relations in all instances of $(\lambda(T') \cup \bigcup_{i=1}^{n} \{\mathsf{cia}(\mathbf{S}_i)\}) \setminus \mathsf{fvar}(T')$ for all $p \in \mathcal{P}$ are of bounded arity.

Let $solcheck(\mathcal{P})$ be the class of all sets of atoms $(\lambda(T') \cup \bigcup_{i=1}^{n} \{\mathsf{cia}(\mathbf{S}_i)\}) \setminus \mathsf{fvar}(T')$ for $\mathcal{P}$ as defined in tractability condition (b). We reduce p-HOM($solcheck(\mathcal{P})$) to p-Eval($\mathcal{P}$) via an FPT reduction. The result then follows directly by Theorem 7, since if (b) is false then $solcheck(\mathcal{P})$ has unbounded treewidth. The rest of this proof gives the desired reduction.

Let $\mathbf{E}, \mathbf{F}$ be an instance of p-HOM($solcheck(\mathcal{P})$). We construct a wdPT p, a database $\mathbf{D}$, and a mapping μ such that $\mu \in p(\mathbf{D})$ if and only if there is a homomorphism from $\mathbf{E}$ to $\mathbf{F}$.

First of all, find a $p = ((T, r), \lambda, \mathcal{X}) \in \mathcal{P}$ and a subtree T' of T containing r such that $\mathbf{E} = (\lambda(T') \cup \bigcup_{i=1}^{n} \{\mathsf{cia}(\mathbf{S}_i)\}) \setminus \mathsf{fvar}(T')$ for some combination $(\mathbf{S}_1, \ldots, \mathbf{S}_n) \in \mathcal{NC}_{t_1} \times \cdots \times \mathcal{NC}_{t_n}$ where $\{t_1, \ldots, t_n\} = ch(T') \cap \mathsf{relv}(T)$. To define a database $\mathbf{D}$ and a mapping μ such that $\mu \in p(\mathbf{D})$ if and only if a homomorphism from $\mathbf{E}$ to $\mathbf{F}$ exists, we need to define the following sets of nodes first. Let $K = ch(T') \cap \mathsf{relv}(T) = \{t_1, \ldots, t_n\}$. For each $t_i \in K$, we define the set N_i of nodes as follows. If $\mathsf{fvar}(t_i) \setminus \mathsf{fvar}(\mathsf{branch}(t_i)) \neq \emptyset$ (i.e., t_i contains some "new" free variable), then $N_i = \emptyset$. Otherwise, let $s_i \in T$ be a descendant of t_i such that $\mathsf{fvar}(s_i) \setminus \mathsf{fvar}(\mathsf{branch}(t_i)) \neq \emptyset$ and such that this property holds for no other node $s_i' \in T$ on the path from t_i to s_i. Then $N_i = \mathsf{cbranch}(s_i) \setminus \mathsf{cbranch}(t_i)$. Finally, let $N = \bigcup_{i=1}^{n} N_i$. Now all notions are in place to describe the database $\mathbf{D}$. While doing so, we implicitly assume that for all positions where an atom $R(\vec{v})$ of p contains a constant, all the atoms in $R^{\mathbf{D}}$ contain the same constant as in $\vec{v}$. I.e., we only describe the values for "variable positions" of $\vec{v}$. Recall that we are dealing with simple wdPTs, thus each relation symbol R occurs at most once within $\lambda(T)$.

For all atoms $R(\vec{y}) \in \lambda(T) \setminus (\lambda(T') \cup \lambda(K) \cup \lambda(N))$, let $R^{\mathbf{D}} = \emptyset$, i.e., for all atoms neither in T' nor in any of the relevant child nodes of T' (or their extensions to some node with a "new" free variable), no matching values exist in the database.

For all atoms $R(\vec{y}) \in \lambda(T')$, we want to use them to simulate in $\mathbf{D}$ the relations in $\mathbf{F}$. Observe that for each such atom, there exists an atom $R'(\vec{z}) \in \mathbf{E}$ that was derived from $R(\vec{y})$ by removing the free variables $\mathsf{fvar}(T')$. Thus, for each atom $R'(\vec{a}) \in R'^{\mathbf{F}}$, we add one atom $R(\vec{b})$ to $R^{\mathbf{D}}$ where $\vec{b}$ contains a fixed domain value $d \in \mathsf{dom}(\mathbf{F})$ at all positions $\vec{y}$ contains a free variable, and the corresponding value from $\vec{a}$ where the variable also occurs in $R'(\vec{z}')$. I.e., $R^{\mathbf{D}}$ is designed in such a way that all variables $x \in \mathsf{fvar}(T')$ can only be mapped to d.

The definition for the atoms in K is more involved. Consider some $t_i \in K$. Let $\vec{v}$ contain the existential interface variables of the node component $\mathbf{S}_i \in \mathcal{NC}_{t_i}$ selected for the construction of $\mathbf{E}$, and assume $\mathsf{cia}(\mathbf{S}_i) = R_{\mathsf{cia}}(\vec{v})$.

For all atoms $R(\vec{y}) \in \lambda(t_i) \setminus \mathbf{S}_i$, set $R^{\mathbf{D}} = \{R(\vec{a}) \mid \vec{a} \in \mathsf{dom}(\mathbf{F})^k\}$ where k is the arity of R. For the atoms in $\mathbf{S}_i$, we distinguish between $\mathbf{S}_i$ being of type (1) or of type (2).

If $\mathbf{S}_i$ is of type (1), i.e., $\mathbf{S}_i$ is of the form $\mathbf{S}_i = \{R(\vec{y})\}$ for some $R(\vec{y}) \in \lambda(t_i)$, define $R^{\mathbf{D}} = \{R(\vec{a}) \mid \vec{a} \in \mathsf{dom}(\mathbf{F})^k$ and $R_{\mathsf{cia}}(\vec{a}_{\vec{v}}) \notin \mathbf{F}\}$, where $\vec{a}_{\vec{v}}$ is the restriction of $\vec{a}$ to those positions in $\vec{y}$ with variables from $\vec{v}$ (and thus not containing variables from $\mathsf{fvar}(T')$).

If $\mathbf{S}_i$ is of type (2), we distinguish two types of variables: those that occur in $\mathcal{I}_t(\mathbf{S}_i)$, and those that do not appear in any node $t' \in \mathsf{branch}(t_i)$. We call these latter variables *new* variables and use as their domain the set $\mathsf{dom}(\mathbf{F})^{|\vec{v}|}$, i.e., the set of all possible assignments of values from $\mathbf{F}$ to the variables in $\vec{v}$ from $R_{\mathsf{cia}}(\vec{v})$. We assume that the encoding of the assignments $\vec{a} \in \mathsf{dom}(\mathbf{F})^{|\vec{v}|}$ is such that we can look up the value that is given to a variable $v_i \in \vec{v}$ by $\vec{a}$. For the remaining variables in $\mathsf{var}(\mathbf{S}_i)$, i.e., the variables not in $\vec{v}$, we will use values from $\mathsf{dom}(\mathbf{F})$. For each atom $R(\vec{y}) \in \mathbf{S}_i$, the values in $R^{\mathbf{D}}$ are defined as follows:

Let $\vec{z} = \vec{y} \cap (\mathsf{var}(\mathbf{S}_i) \setminus \vec{v})$ (because $\mathbf{S}_i$ is of type (2), $\vec{z} \setminus \mathsf{fvar}(T') \neq \emptyset$). Then $R^{\mathbf{D}}$ contains an atom for each tuple satisfying all of the following four properties.

1. All variables in $\mathsf{fvar}(T')$ get the value d.
2. All the variables in $\vec{z} \setminus \mathsf{fvar}(T')$ get assigned the same value. Denote this value by a, and recall that a represents an assignment $\vec{a} \in \mathsf{dom}(\mathbf{F})^{|\vec{v}|}$.

3. For all $v_i \in \vec{v} \cap \vec{y}$, the value of v_i is consistent with the vector $\vec{a}$ represented by a.

4. We have $R_{\mathsf{cia}}(\vec{a}) \notin \mathbf{F}$.

Because their arity is assumed to be bounded, all these relations can be constructed in polynomial time. To conclude the definition of $\mathbf{D}$, for all atoms $R(\vec{y}) \in \lambda(N)$, set $R^{\mathbf{D}} = \{R(\vec{a}) \mid \vec{a} \in \mathsf{dom}(\mathbf{D})^k\}$, where k is the arity of R and $\mathsf{dom}(\mathbf{D})$ is implicitly defined to consist of all values mentioned in the definition of $\mathbf{D}$.

Finally, let μ be the mapping defined on all variables $x \in \mathsf{fvar}(T')$ as $\mu(x) = d$. To prove $\mu \in p(\mathbf{D})$ if and only if homomorphism $\mathbf{E} \to \mathbf{F}$ exists, we first show the following claim.

▷ Claim 26. Let $R_{\mathsf{cia}}(\vec{v})$ be an interface atom and $\mathbf{S}_i$ the corresponding interface component. Then, for a mapping $\mu' \colon \vec{v} \to \mathsf{dom}(\mathbf{F})$, we have that $R_{\mathsf{cia}}(\mu'(\vec{v})) \in \mathbf{F}$ if and only if there is no extension μ'' of $\mu' \cup \mu$ to $\mathsf{var}(\lambda(t_i))$ that maps all atoms in $\mathbf{S}_i$ into $\mathbf{D}$ (since $\mathbf{S}_i \subseteq \lambda(t_i)$, this implies that there exists no extension μ'' of $\mu' \cup \mu$ that maps $\lambda(t_i)$ into $\mathbf{D}$).

Proof. For node components of type (1), the claim is immediate. So let us assume for the rest of the proof that $\mathbf{S}_i$ is of type (2).

First let $R_{\mathsf{cia}}(\mu'(\vec{v})) \in \mathbf{F}$. Let us assume an arbitrary extension μ'' of $\mu' \cup \mu$ to $\mathsf{var}(\mathbf{S}_i)$. If μ'' does not satisfy conditions 1., 2., and 3. for all $R(\vec{y}) \in \mathbf{S}_i$, then clearly for this particular atom there exists no atom in $\mathbf{D}$ to which it can be mapped by μ''. We may thus assume that μ'' satisfies the first three conditions for all atoms $R(\vec{y}) \in \mathbf{S}_i$. Then all variables in $\mathsf{var}(\mathbf{S}_i) \setminus (\vec{v} \cup \mathsf{fvar}(T'))$ take the same value under μ'', and this value corresponds exactly to the tuple $\mu'(\vec{v})$. But then μ'' does not satisfy condition 4. for any $R(\vec{y}) \in \mathbf{S}_i$ since $R_{\mathsf{cia}}(\mu'(\vec{v})) \in R^{\mathbf{F}}_{\mathsf{cia}}$ by assumption. Thus $R^{\mathbf{D}}$ does not contain any atom onto which μ'' could map $R(\vec{y})$ and thus μ'' cannot exist which completes the first direction.

For the other direction, assume that no extension μ'' of $\mu' \cup \mu$ maps all atoms in $\mathbf{S}_i$ into $\mathbf{D}$. Then this is in particular true for those assignments satisfying conditions 1., 2., and 3. Note that every such assignment maps all variables in $\vec{z} \setminus \mathsf{fvar}(T')$ to the same value, representing a mapping on $\vec{v}$. Also, $\mu''|_{\vec{v}} = \mu'$. Since μ'' fails to map $\mathbf{S}_i$ into $\mathbf{D}$ because of 4., we get that $R_{\mathsf{cia}}(\mu'(\vec{v})) \in R^{\mathbf{F}}_{\mathsf{cia}}$, which completes the proof of the claim. ◁

We continue the proof that $\mu \in p(\mathbf{D})$ if and only if a homomorphism $\mathbf{E} \to \mathbf{F}$ exists. First observe that $\mu \in p(\mathbf{D})$ if and only if on the one hand there is an extension μ' of μ to $\mathsf{var}(T')$ that maps all atoms in $\lambda(T')$ into $\mathbf{D}$ (of course, in general every subtree T'' containing the root node of T with $\mathsf{fvar}(T'') = \mathsf{dom}(\mu)$ is a potential candidate, but given the construction of $\mathbf{D}$, the subtree T' is the only possible candidate) and, on the other hand, for all $t_i \in K$, we have that there does not exist an extension of μ' onto $\lambda(t_i) \cup \lambda(N_i)$. (In fact, extending the mapping to any descendant of t_i that contains some additional free variable would work. However, the only nodes with non-empty relations in $\mathbf{D}$ are those mentioned in N.)

By the construction of $\mathbf{D}$ for atoms in $\lambda(N)$, for every $t_i \in K$ it follows immediately that there exists an extension of μ' onto $\lambda(t_i) \cup \lambda(N_i)$ if and only if there exists an extension to $\lambda(t_i)$. This is because for the atoms in $\lambda(N)$ the database $\mathbf{D}$ contains all possible atoms, thus every extension μ'' of μ' onto $\lambda(t_i)$ can be further extended to all atoms in $\lambda(N_i)$.

Note that the existence of an extension of μ' onto $\lambda(t_i)$ is, by Claim 26, equivalent to μ' sending $R_{\mathsf{cia}}(\vec{v})$ into $\mathbf{F}$. So $\mu \in p(\mathbf{D})$ if and only if there is a homomorphism from $\mathbf{E}$ into $\mathbf{F}$. This completes the proof. ◀

6 Relationship with SPARQL and Conclusion

Our results give a fine understanding of the tractable classes of wdPTs in the presence of projection. In particular they show the different sources of hardness. As laid out in the introduction, there is a strong relationship between well-designed SPARQL queries and wdPTs: For every well-designed SPARQL query, an equivalent well-designed pattern tree can be computed in polynomial time, and vice versa, in a completely syntactic way.

Note that our characterization of tractable classes unfortunately cannot be immediately translated to well-designed SPARQL queries. This is because our characterization only applies to classes of *simple* well-designed pattern trees. However, RDF triples and SPARQL triple patterns, in the relational model, are usually represented with a single (ternary) relation. Thus, there is no direct translation to and from simple (well-designed) pattern trees. As a consequence, our result does not imply an immediate characterization of the tractable classes of well-designed {AND, OPTIONAL}-SPARQL queries.

Nevertheless, our results also give interesting insights to SPARQL with projections. First, Algorithm 1 directly applies to queries in which relation symbols appear several times and thus in particular for well-designed pattern trees resulting from the translation of well-designed SPARQL queries. Moreover, our result determines completely the tractable classes that can be characterized by analyzing only the underlying graph structure of the queries, i.e., the Gaifman graph. Indeed, since simple queries can simulate all other queries sharing the same Gaifman graph by duplicating relations, Gaifman graph based techniques have exactly the same limits as simple queries. Thus, our work gives significant information on limits of tractability for SPARQL queries in the same way as e.g. [12, 4, 5] did in similar contexts.

Let us mention one major stumbling block towards a characterization of non-simple well-designed pattern trees with projections: In the proof of Lemma 24, we have used a reduction from quantified conjunctive queries. Unfortunately, the tractable classes for the non-simple fragment for that problem is not well understood which limits our result to simple queries since we are using the respective results from [5]. Note that we might have been able to give a more fine-grained result in sorted logics by using [6], but since this would, in our opinion, not have been very natural in our setting, we did not pursue this direction. Thus a better understanding of non-simple pattern trees would either need progress on quantified conjunctive queries or a reduction from another problem that is better understood.

One prominent operator of SPARQL that we did not consider is UNION, whose correspondence in pattern trees are sets of pattern trees, so-called pattern forests. While the extension to simple pattern forests is immediate (since no two trees share any relation symbols), it is not clear how to approach the possible repetition of relation symbol within different trees in forests of simple pattern trees in combination with projection.

Finally, another interesting class of queries are weakly well-designed pattern trees. While the tractability conditions can be easily adapted to provide FPT algorithms for these queries, providing a characterization of the tractable classes is much harder due to the fact that relevant nodes need not have a descendant introducing a "new" free variable.

References

1 Marcelo Arenas, Gonzalo I. Diaz, and Egor V. Kostylev. Reverse Engineering SPARQL Queries. In *Proc. WWW 2016*, pages 239–249. ACM, 2016.

2 Marcelo Arenas and Jorge Pérez. Querying semantic web data with SPARQL. In *Proc. PODS 2011*, pages 305–316. ACM, 2011.

3 Pablo Barceló, Markus Kröll, Reinhard Pichler, and Sebastian Skritek. Efficient Evaluation and Static Analysis for Well-Designed Pattern Trees with Projection. *ACM Trans. Database Syst.*, 43(2):8:1–8:44, 2018.

4 Hubie Chen. The tractability frontier of graph-like first-order query sets. In Thomas A. Henzinger and Dale Miller, editors, *Proc. CSL-LICS 2014*, pages 31:1–31:9. ACM, 2014.

5 Hubie Chen and Víctor Dalmau. Decomposing Quantified Conjunctive (or Disjunctive) Formulas. In *Proc. LICS 2012*, pages 205–214. IEEE Computer Society, 2012.

6 Hubie Chen and Dániel Marx. Block-Sorted Quantified Conjunctive Queries. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Proc. ICALP 2013*, volume 7966 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 2013.

7 Arnaud Durand and Stefan Mengel. Structural Tractability of Counting of Solutions to Conjunctive Queries. *Theory of Computing Systems*, pages 1–48, 2014.

8 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

9 Martin Grohe. The Parameterized Complexity of Database Queries. In Peter Buneman, editor, *Proc. PODS 2001*, pages 82–92. ACM, 2001.

10 Martin Grohe. Parameterized Complexity for the Database Theorist. *SIGMOD Record*, 31(4):86–96, 2002.

11 Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM*, 54(1), 2007.

12 Martin Grohe, Thomas Schwentick, and Luc Segoufin. When is the evaluation of conjunctive queries tractable? In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proc. STOC 2001*, pages 657–666. ACM, 2001.

13 Mark Kaminski and Egor V. Kostylev. Beyond Well-designed SPARQL. In *Proc. ICDT 2016*, volume 48 of *LIPIcs*, pages 5:1–5:18. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016.

14 Egor V. Kostylev, Juan L. Reutter, and Martín Ugarte. CONSTRUCT queries in SPARQL. In *Proc. ICDT 2015*, volume 31 of *LIPIcs*, pages 212–229. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2015.

15 Markus Kröll, Reinhard Pichler, and Sebastian Skritek. On the Complexity of Enumerating the Answers to Well-Designed Pattern Trees. In *Proc. ICDT 2016*, volume 48 of *LIPIcs*, pages 22:1–22:18. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016.

16 Andrés Letelier, Jorge Pérez, Reinhard Pichler, and Sebastian Skritek. Static analysis and optimization of semantic web queries. *ACM Trans. Database Syst.*, 38(4):25, 2013.

17 Dániel Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. In Leonard J. Schulman, editor, *Proc. STOC 2010*, pages 735–744. ACM, 2010.

18 Christos H. Papadimitriou and Mihalis Yannakakis. On the Complexity of Database Queries. *J. Comput. Syst. Sci.*, 58(3):407–427, 1999.

19 Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3), 2009.

20 François Picalausa and Stijn Vansummeren. What are real SPARQL queries like? In *Proc. SWIM 2011*, page 7. ACM, 2011.

21 Reinhard Pichler and Sebastian Skritek. Containment and equivalence of well-designed SPARQL. In *Proc. PODS 2014*, pages 39–50. ACM, 2014.

22 Miguel Romero. The Tractability Frontier of Well-designed SPARQL Queries. In *Proc. PODS 2018*, pages 295–306. ACM, 2018.

Boolean Tensor Decomposition for Conjunctive Queries with Negation

Mahmoud Abo Khamis
RelationalAI, Berkeley, USA

Hung Q. Ngo
RelationalAI, Berkeley, USA

Dan Olteanu
Department of Computer Science, University of Oxford, UK

Dan Suciu
Department of Computer Science and Engineering, University of Washington, USA

Abstract

We propose an approach for answering conjunctive queries with negation, where the negated relations have bounded degree. Its data complexity matches that of the InsideOut and PANDA algorithms for the positive subquery of the input query and is expressed in terms of the fractional hypertree width and the submodular width respectively. Its query complexity depends on the structure of the conjunction of negated relations; in general it is exponential in the number of join variables occurring in negated relations yet it becomes polynomial for several classes of queries.

This approach relies on several contributions. We show how to rewrite queries with negation on bounded-degree relations into equivalent conjunctive queries with not-all-equal (NAE) predicates, which are a multi-dimensional analog of disequality ($\neq$). We then generalize the known color-coding technique to conjunctions of NAE predicates and explain it via a Boolean tensor decomposition of conjunctions of NAE predicates. This decomposition can be achieved via a probabilistic construction that can be derandomized efficiently.

2012 ACM Subject Classification Theory of computation → Database query processing and optimization (theory); Information systems → Database query processing

Keywords and phrases color-coding, combined complexity, negation, query evaluation

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.21

Related Version An extended online version of this work includes the missing proofs [1], see `https://arxiv.org/abs/1712.07445`.

Funding This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 682588. This project is also supported in part by NSF grants AITF-1535565 and III-1614738.

Acknowledgements The authors would like to thank the anonymous reviewers for their suggestions that helped improve the readability of this paper.

1 Introduction

This paper considers the problem of answering conjunctive queries with negation of the form

$$Q(\boldsymbol{X}_F) \leftarrow \mathsf{body} \wedge \bigwedge_{S \in \overline{\mathcal{E}}} \neg R_S(\boldsymbol{X}_S), \tag{1}$$

where body is the body of an arbitrary conjunctive query, $\boldsymbol{X}_F = (X_i)_{i \in F}$ denotes a tuple of variables (or attributes) indexed by a set F of positive integers, and $\overline{\mathcal{E}}$ is the set of hyperedges

of a multi-hypergraph[1] $\overline{\mathcal{H}} = (\overline{\mathcal{V}}, \overline{\mathcal{E}})$. Every hyperedge $S \in \overline{\mathcal{E}}$ corresponds to a bounded-degree relation R_S on attributes $\boldsymbol{X}_S$. For instance, the equality (=) relation is a bounded-degree (binary) relation, because every element in the active domain has degree one; the edge relation E of a graph with bounded maximum degree is also a bounded-degree relation. Section 2 formalizes this notion of bounded degree.

We exemplify using three Boolean queries[2] over a directed graph $G = ([n], E)$ with n nodes and $N = |E|$ edges: the k-walk query[3], the k-path query, and the induced (or chordless) k-path query. They have the same body and encode graph problems of increasing complexity:

$$W \leftarrow E(X_1, X_2) \wedge E(X_2, X_3) \wedge \cdots \wedge E(X_k, X_{k+1}).$$

$$P \leftarrow E(X_1, X_2) \wedge E(X_2, X_3) \wedge \cdots \wedge E(X_k, X_{k+1}) \wedge \bigwedge_{\substack{i,j \in [k+1] \\ i+1<j}} X_i \neq X_j. \tag{2}$$

$$I \leftarrow E(X_1, X_2) \wedge E(X_2, X_3) \wedge \cdots \wedge E(X_k, X_{k+1}) \wedge \bigwedge_{\substack{i,j \in [k+1] \\ i+1<j}} (\neg E(X_i, X_j) \wedge X_i \neq X_j)). \tag{3}$$

The hypergraph $\overline{\mathcal{H}}$ for the k-walk query W is empty since it has no negated relations. This query can be answered in $O(kN \log N)$ time using for instance the Yannakakis dynamic programming algorithm [34]. The k-path query P has the hypergraph $\overline{\mathcal{H}} = ([k+1], \{(i,j) \mid i, j \in [k+1], i+1 < j\})$. It can be answered in $O(k^k N \log N)$-time [30] and even better in $2^{O(k)} N \log N$-time using the color-coding technique [8]. The induced k-path query I has the hypergraph $\overline{\mathcal{H}}$ similar to that of P, but every edge (i,j) has now multiplicity two due to the negated edge relation and also the disequality. This query is W[2]-hard [10]. However, if the graph G has a maximal degree that is bounded by some constant d, then the query can be answered in $O(f(k,d) \cdot N \log N)$-time for some function f that depends exponentially on k and d [30]. Our results imply the above complexities for the three queries.

1.1 Main Contribution

In this paper we propose an approach to answering conjunctive queries with negation on bounded-degree relations of arbitrary arities. Our approach is the first to exploit the bounded degree of the negated relations. The best known algorithms for positive queries such as InsideOut [2] and PANDA [3] can also answer queries with negation, albeit with much higher complexity since already one negation can increase their worst-case runtime. For example, the Boolean path queries with a disequality between the two end points takes linear time with our approach, but quadratic time with existing approaches [2, 3]. The data complexity of our approach matches that of InsideOut and PANDA for the positive subquery $Q(\boldsymbol{X}_F) \leftarrow \mathsf{body}$. To lower its query complexity, we use a range of techniques including color-coding, probabilistic construction of Boolean tensor decompositions, and derandomization of this construction.

▶ **Theorem 1.1.** *Any query Q of the form (1), where for each $S \in \overline{\mathcal{E}}$ the relation R_S has bounded degree and $f(\cdot)$ is a function of Q, can be answered over a database of size N in time $O(f(Q) \cdot \log N \cdot (N^{\mathsf{fhtw}_F(\mathsf{body})} + |\mathsf{output}|))$ using a reduction to* InsideOut *and $O(f(Q) \cdot (\mathsf{poly}(\log N) \cdot N^{\mathsf{subw}_F(\mathsf{body})} + \log N \cdot |\mathsf{output}|))$ using a reduction to* PANDA.

[1] In a multi-hypergraph, each hyperedge S can occur multiple times. All hypergraphs in this paper are multi-hypergraphs.
[2] We denote Boolean queries $Q(\boldsymbol{X}_F)$ where $F = \emptyset$ by Q instead of $Q()$. We also use $[n] = \{1, \ldots, n\}$.
[3] Unlike a path, in a walk some vertices may repeat.

The complexities of InsideOut [2] and PANDA [3] depend on the fractional hypertree width fhtw [17] and respectively the submodular width subw [23]. The widths fhtw_F and subw_F are fhtw and respectively subw computed on the subset of hypertree decompositions of the positive subquery of Q for which the set F of free variables form a connected subtree. The dependency of the function f on the structure of Q, and in particular on the hypergraph $\overline{\mathcal{H}}$ of the negated relations in Q, is an important result of this paper.

Theorem 1.1 draws on three contributions:

1. A rewriting of queries of the form (1) into equivalent conjunctive queries with not-all-equal predicates, which are a multi-dimensional analog of disequality $\neq$ (Proposition 4.3);
2. A generalization of color-coding [8] from cliques of disequalities to arbitrary conjunctions of not-all-equal predicates; and
3. An alternative view of color coding via Boolean tensor decomposition of conjunctions of not-all-equal predicates (Lemma 5.1). This decomposition admits a probabilistic construction that can be derandomized efficiently (Corollary 5.7).

Contribution 1 (Section 4) gives a rewrite of the query Q into an equivalent disjunction of queries Q_i of the form (cf. Proposition 4.3)

$$Q_i(\boldsymbol{X}_F) \leftarrow \mathsf{body}_i \wedge \bigwedge_{S \in \mathcal{E}_i} \mathsf{NAE}(\boldsymbol{Z}_S).$$

For each query Q_i, body_i may be different from body in Q, since fresh variables $\boldsymbol{Z}_S$ and unary predicates may be introduced. Its fractional hypertree and submodular widths remain however at most that of body. We thus rewrite the conjunction of the negated relations into a much simpler conjunction of NAE predicates without increasing the data complexity of Q. The number of such queries Q_i depends exponentially on the arities and the degrees of the negated relations, which is the reason why we need the constant bound on these degrees.

Contribution 2 (Section 5) is based on the observation that a conjunction of NAE predicates can be answered by an adaptation of the color-coding technique [8], which has been used so far for checking cliques of disequalities. The crux of this technique is to randomly color each value in the active domain with one color from a set whose size is much smaller than the size of the active domain, and to use these colors instead of the values themselves to check the disequalities. We generalize this idea to conjunctions of NAE predicates and show that such conjunctions can be expressed equivalently as disjunctions of simple queries over the different possible colorings of the variables in these queries.

Contribution 3 (Section 5) explains color coding by providing an alternative view of it: Color coding is a (Boolean) tensor decomposition of the (Boolean) tensor defined by the conjunction $\bigwedge_S \mathsf{NAE}(\boldsymbol{Z}_S)$. As a tensor, $\bigwedge_S \mathsf{NAE}(\boldsymbol{Z}_S)$ is a multivariate function over variables in the set $U = \bigcup_S \boldsymbol{Z}_S$. The tensor decomposition rewrites it into a disjunction of conjunctions of *univariate* functions over individual variables Z_i (Lemma 5.1). That is,

$$\bigwedge_S \mathsf{NAE}(\boldsymbol{Z}_S) \equiv \bigvee_{j \in [r]} \bigwedge_{i \in U} f_i^{(j)}(Z_i),$$

where r is the (Boolean tensor) rank of the tensor decomposition, and for each $j \in [r]$, the inner conjunction $\bigwedge_{i \in U} f_i^{(j)}(Z_i)$ can be thought of as a rank-1 tensor of inexpensive Boolean univariate functions $f_i^{(j)}(\cdot)$ $(\forall i \in U)$. The key advantages of this tensor decomposition are that (i) the addition of univariate conjuncts to body_i does not increase its (fractional hypertree and submodular) width and (ii) the dependency of the rank r on the database size N is only a $\log N$ factor. Lemma 5.1 shows that the rank r depends on two quantities:

$r = P(\mathcal{G}, c) \cdot |\mathcal{F}|$. The first is the chromatic polynomial of the hypergraph of $\bigwedge_S \mathsf{NAE}(\boldsymbol{Z}_S)$ using c colors. The second is the size of a family of hash functions that represent proper c-colorings of homomorphic images of the input database. The number c of needed colors is at most the number $|U|$ of variables in $\bigwedge_S \mathsf{NAE}(\boldsymbol{Z}_S)$. We show it to be the maximum chromatic number of a hypergraph defined by any homomorphic image of the database.

We give a probabilistic construction of the tensor decomposition that generalizes the construction used by the color-coding technique. It selects a color distribution dependent on the query structure, which allows the rank of $\bigwedge_S \mathsf{NAE}(\boldsymbol{Z}_S)$ to take a wide range of query complexity asymptotics, from polynomial to exponential in the query size. This is more refined than the previously known bound [8], which amounts to a tensor rank that is exponential in the query size. We further derandomize this construction by adapting ideas from derandomization for k-restrictions [6] (with k being related to the Boolean tensor rank).

Section 6 shows how to use the Boolean tensor decomposition in conjunction with InsideOut [2] and PANDA [3] to evaluate queries of the form (1) with the complexity given by Theorem 1.1. The query complexity captured by the function f is given by the number of NAE predicates and the rank of the tensor decomposition of their conjunction.

2 Preliminaries

In this paper we consider arbitrary conjunctive queries with negated relations of the form (1). We make use of the following naming convention. Capital letters with subscripts such as X_i or A_j denote variables. For any set S of positive integers, $\boldsymbol{X}_S = (X_i)_{i \in S}$ denote a tuple of variables indexed by S. Given a relation R over variables $\boldsymbol{X}_S$ and $J \subseteq S$, $\pi_J R$ denotes the projection of R onto variables $\boldsymbol{X}_J$, i.e., we write $\pi_J R$ instead of $\pi_{\boldsymbol{X}_J} R$. If X_i is a variable, then the corresponding lower-case x_i denotes a value from the active domain $\mathsf{Dom}(X_i)$ of X_i. Bold-face $\boldsymbol{x}_S = (x_i)_{i \in S}$ denotes a tuple of values in $\prod_{i \in S} \mathsf{Dom}(X_i)$.

We associate a hypergraph $\mathcal{H}(R)$ with a relation $R(\boldsymbol{X}_S)$ as follows. The vertex set is $\{(i, v) \mid i \in S, v \in \mathsf{Dom}(X_i)\}$. Each tuple $\boldsymbol{x}_S = (x_i)_{i \in S} \in R$ corresponds to a (hyper)edge $\{(i, x_i) | i \in S\}$. $\mathcal{H}(R)$ is a $|S|$-uniform hypergraph (all hyperedges have size $|S|$).

2.1 Hypergraph coloring and bounded-degree relations

Hypergraph coloring. Let $\mathcal{G} = (U, \mathcal{A})$ denote a multi-hypergraph and k be a positive integer. A *proper c-coloring* of $\mathcal{G}$ is a mapping $h : U \to [c]$ such that for every edge $S \in \mathcal{A}$, there exists $u, v \in S$ with $u \neq v$ such that $h(u) \neq h(v)$. The *chromatic polynomial* $P(\mathcal{G}, c)$ of $\mathcal{G}$ is the number of proper c-colorings of $\mathcal{G}$ [18]. A vertex (edge) coloring of $\mathcal{G}$ is an assignment of colors to the vertices (edges) of $\mathcal{G}$ so that no two adjacent vertices (incident edges) have the same color. The *chromatic number* $\chi(\mathcal{G})$ and the *chromatic index* $\chi'(\mathcal{G})$ are the smallest numbers of colors needed for a vertex coloring and respectively an edge coloring of $\mathcal{G}$. Coloring a (hyper)graph is equivalent to coloring it without singleton edges.

Bounded-degree relation. The *maximum degree* of a vertex in a hypergraph $\mathcal{G} = (U, \mathcal{A})$ is denoted by $\Delta(\mathcal{G})$: $\Delta(\mathcal{G}) = \max_{v \in U} |\{S \in \mathcal{A} \mid v \in S\}|$. For a relation $R_S(\boldsymbol{X}_S)$, its maximum degree $\Delta(\mathcal{H}(R_S))$ is the maximum number of tuples in R_S with the same value for a variable $X \in \boldsymbol{X}_S$: $\Delta(\mathcal{H}(R_S)) = \max_{\substack{i \in S \\ v \in \mathsf{Dom}(X_i)}} |\{\boldsymbol{x}_S \in R_S \mid x_i = v\}|$. We will use a slightly different notion of *degree* of a relation denoted by $\deg(R_S)$, which also accounts for the arity $|S|$ of the relation R_S. Proposition 2.3 connects the two notions.

▶ **Definition 2.1** (Matching). *A k-ary relation $M(\boldsymbol{X}_S)$ is called a (k-dimensional)* matching *if for every two tuples $\boldsymbol{x}_S, \boldsymbol{x}'_S \in M$, either $\boldsymbol{x}_S = \boldsymbol{x}'_S$, i.e., $\boldsymbol{x}_S$ and $\boldsymbol{x}'_S$ are the same tuple, or it holds that $x_i \neq x'_i, \forall i \in S$.*

▶ **Definition 2.2** (Degree). *The* degree *of a relation $R_S(\boldsymbol{X}_S)$, denoted by $\deg(R_S)$, is the smallest integer d for which R_S can be written as the disjoint union of d matchings. The degree $\deg(R_S)$ is* bounded *if there is a constant d_S such that $\deg(R_S) \leq d_S$.*

It is easy to see that $\deg(R_S) = \chi'(\mathcal{H}(R_S))$. If R_S is a binary relation, then $\mathcal{H}(R_S)$ is a bipartite graph and $\deg(R_S) = \chi'(\mathcal{H}(R_S)) = \Delta(\mathcal{H}(R_S))$. This follows from König's line coloring theorem [20], which states that the chromatic index of a bipartite graph is equal to its maximum degree. When the arity k is higher than two, to the best of our knowledge there does not exist such a nice characterization of the chromatic index of R_S in terms of the maximum degree of individual vertices in the graph, although there has been some work on bounding the chromatic index of (linear) uniform hypergraphs [5, 22, 13, 29, 29]. In our setting, we are willing to live with sub-optimal decomposition of a bounded-degree relation into matchings as long as it can be done in linear time.

▶ **Proposition 2.3.** *Let $R_S(\boldsymbol{X}_S)$ denote a k-ary relation of size N and $\ell = \Delta(\mathcal{H}(R_S))$. Then:*

- $\ell \leq \deg(R_S) \leq k(\ell - 1) + 1$;
- *We can compute in $O(N)$-time disjoint k-ary matchings $M_1, \ldots, M_{k\ell-k+1}$ such that $R_S = \bigcup_{j=1}^{k(\ell-1)+1} M_j$.*

Proof. The fact that $\ell \leq \deg(R_S)$ is obvious. To show that $\deg(R_S) \leq k(\ell - 1) + 1$, note that any edge in $\mathcal{H}(R_S)$ is adjacent to at most $k(\ell - 1)$ other edges of $\mathcal{H}(R_S)$, hence greedy coloring can color the edges of $\mathcal{H}(R_S)$ in time $O(N)$ using $k(\ell - 1) + 1$ colors. ◀

The two notions of degree of a relation are thus equivalent up to a constant factor given by the arity of the relation.

2.2 FAQ, width parameters, and corresponding algorithms

▶ **Definition 2.4** (The FAQ problem [2]). *The input to* FAQ *is a set of functions and the output is a function which is a series of aggregations (e.g. sums) over the product of input functions. In particular, the input to* FAQ *consists of the following:*

- *A multi-hypergraph $\mathcal{H} = (\mathcal{V} = [n], \mathcal{E})$.*
- *Each vertex $i \in \mathcal{V} = [n]$ corresponds to a variable X_i over a discrete domain $\mathsf{Dom}(X_i)$.*
- *For each hyperedge $S \in \mathcal{E}$, there is a corresponding input function (also called a* factor*) $\psi_S : \prod_{i \in S} \mathsf{Dom}(X_i) \to \mathbf{D}$ for some fixed $\mathbf{D}$.*
- *A number $f \in [n]$. Let $F := [f]$. Variables in $\boldsymbol{X}_F$ are called* free variables*, while variables in $\boldsymbol{X}_{[n]-F}$ are called* bound variables*.*
- *For each $i \in [n] - F$, there is a commutative semiring $(\mathbf{D}, \bigoplus^{(i)}, \bigotimes)$. (All the semirings share the same $\mathbf{D}$ and $\bigotimes$ but can potentially have different $\bigoplus^{(i)}$).*[4]

The FAQ *problem is to compute the following function $\varphi(\boldsymbol{x}_F) : \prod_{i \in F} \mathsf{Dom}(X_i) \to \mathbf{D}$*

$$\varphi(\boldsymbol{x}_F) := \bigoplus_{x_{f+1}}^{(f+1)} \cdots \bigoplus_{x_n}^{(n)} \bigotimes_{S \in \mathcal{E}} \psi_S(\boldsymbol{x}_s). \tag{4}$$

[4] More generally, instead of $(\mathbf{D}, \oplus^{(i)}, \otimes)$ being a semiring, we also allow some $\oplus^{(i)}$ to be identical to $\otimes$.

Consider the conjunctive query $Q(\boldsymbol{X}_F) \leftarrow \bigwedge_{S\in\mathcal{E}} R_S(\boldsymbol{X}_S)$ where $\boldsymbol{X}_F$ is the set of free variables. The FAQ framework models each input relation R_S as a Boolean function $\psi_S(\boldsymbol{x}_S)$, called a "factor", in which $\psi_S(\boldsymbol{x}_S) = \mathsf{true}$ iff $\boldsymbol{x}_S \in R_S$. Then, computing the output $Q(\boldsymbol{X}_F)$ is equivalent to computing the Boolean function $\varphi(\boldsymbol{x}_F)$ defined as $\varphi(\boldsymbol{x}_F) = \bigvee_{x_{f+1}} \cdots \bigvee_{x_n} \bigwedge_{S\in\mathcal{E}} \psi_S(\boldsymbol{x}_S)$. Instead of Boolean functions, this expression can be defined in SumProd form over functions on a commutative semiring $(\boldsymbol{D}, \oplus, \otimes)$:

$$\varphi(\boldsymbol{x}_F) = \bigoplus_{x_{f+1}} \cdots \bigoplus_{x_n} \bigotimes_{S\in\mathcal{E}} \psi_S(\boldsymbol{x}_S). \tag{5}$$

The semiring $(\{\mathsf{true}, \mathsf{false}\}, \vee, \wedge)$ was used for Q above.

We next define tree decompositions and the fhtw and subw parameters. We refer the reader to the recent survey by Gottlob et al. [15] for more details and a historical context. In what follows, the hypergraph $\mathcal{H}$ should be thought of as the hypergraph of the input FAQ query, although the notions of tree decomposition and width parameters are defined independently of queries.

A *tree decomposition* of a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is a pair (T, χ), where T is a tree and $\chi : V(T) \to 2^{\mathcal{V}}$ maps each node t of the tree to a subset $\chi(t)$ of vertices such that:

(a) Every hyperedge $S \in \mathcal{E}$ is a subset of some $\chi(t)$, $t \in V(T)$ (i.e. every edge is covered by some bag);

(b) For every vertex $v \in \mathcal{V}$, the set $\{t \mid v \in \chi(t)\}$ is a non-empty (connected) sub-tree of T. This is called the *running intersection property*.

The sets $\chi(t)$ are often called the *bags* of the tree decomposition. Let $\mathsf{TD}(\mathcal{H})$ denote the set of all tree decompositions of $\mathcal{H}$. When $\mathcal{H}$ is clear from context, we use TD for brevity.

▶ **Definition 2.5** (F-connex tree decomposition [9, 32]). *Given a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ and a set $F \subseteq \mathcal{V}$, a tree decomposition (T, χ) of $\mathcal{H}$ is* F-connex *if there is a subset $V' \subseteq V(T)$ that forms a connected subtree of T and satisfies $\bigcup_{t\in V'} \chi(t) = F$. We use TD_F to denote the set of all F-connex tree decompositions of $\mathcal{H}$. (Note that when $F = \emptyset$, $\mathsf{TD}_F = \mathsf{TD}$.)*

To define width parameters, we use the polymatroid characterization from [3]. A function $f : 2^{\mathcal{V}} \to \mathbb{R}_+$ is called a (non-negative) *set function* on $\mathcal{V}$. A set function f on $\mathcal{V}$ is *modular* if $f(S) = \sum_{v\in S} f(\{v\})$ for all $S \subseteq \mathcal{V}$, it is *monotone* if $f(X) \le f(Y)$ whenever $X \subseteq Y \subseteq \mathcal{V}$, and it is *submodular* if $f(X \cup Y) + f(X \cap Y) \le f(X) + f(Y)$ for all $X, Y \subseteq \mathcal{V}$. A monotone, submodular set function $h : 2^{\mathcal{V}} \to \mathbb{R}_+$ with $h(\emptyset) = 0$ is called a *polymatroid*. Let Γ_n denote the set of all polymatroids on $\mathcal{V} = [n]$.

Given a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, define the set of *edge dominated* set functions, denoted by $\mathsf{ED}_{\mathcal{H}}$ or ED when $\mathcal{H}$ is clear from the context, as follows:

$$\mathsf{ED} := \{h \mid h : 2^{\mathcal{V}} \to \mathbb{R}_+, h(S) \le 1, \forall S \in \mathcal{E}\}. \tag{6}$$

We can now define the submodular width and fractional hypertree width of a given hypergraph $\mathcal{H}$ (or of a given FAQ query with hypergraph $\mathcal{H}$): [5]

$$\mathsf{fhtw}(\mathcal{H}) := \min_{(T,\chi)\in\mathsf{TD}} \max_{h\in\mathsf{ED}\cap\Gamma_n} \max_{t\in V(T)} h(\chi(t)), \quad \mathsf{fhtw}_F(\mathcal{H}) := \min_{(T,\chi)\in\mathsf{TD}_F} \max_{h\in\mathsf{ED}\cap\Gamma_n} \max_{t\in V(T)} h(\chi(t)),$$

$$\mathsf{subw}(\mathcal{H}) := \max_{h\in\mathsf{ED}\cap\Gamma_n} \min_{(T,\chi)\in\mathsf{TD}} \max_{t\in V(T)} h(\chi(t)), \quad \mathsf{subw}_F(\mathcal{H}) := \max_{h\in\mathsf{ED}\cap\Gamma_n} \min_{(T,\chi)\in\mathsf{TD}_F} \max_{t\in V(T)} h(\chi(t)).$$

[5] Although this definition for fhtw differs from the original one [17, 15], the two definitions have been shown to be equivalent [3].

It is known that $\mathsf{subw}(\mathcal{H}) \leq \mathsf{fhtw}(\mathcal{H})$, and there are classes of hypergraphs with bounded subw and unbounded fhtw [23]. Furthermore, fhtw is strictly less than other width notions such as (generalized) hypertree width and tree width.

▶ **Theorem 2.6** ([2]). *Given an* FAQ φ *over a single semiring with hypergraph* $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ *and free variables* $F \subseteq \mathcal{V}$ *over a database of size* N*, the* InsideOut *algorithm can answer* φ *in time* $O(|\mathcal{E}| \cdot |\mathcal{V}|^2 \cdot \log N \cdot (N^{\mathsf{fhtw}_F(\mathcal{H})} + |\mathsf{output}|))$.

▶ **Theorem 2.7** ([3]). *Given an* FAQ φ *over the Boolean semiring with hypergraph* $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ *and free variables* $F \subseteq \mathcal{V}$ *over a database of size* N*, the* PANDA *algorithm can answer* φ *in time* $O(|\mathcal{V}| \cdot 2^{2^{|\mathcal{V}|}} \cdot (\mathsf{poly}(\log N) \cdot N^{\mathsf{subw}_F(\mathcal{H})} + \log N \cdot |\mathsf{output}|))$.

3 Example

We illustrate our approach using the following Boolean query[6]:

$$C \leftarrow R(X,Y), S(Y,Z), \neg T(X,Z) \tag{7}$$

where all input relations have sizes upper bounded by N and thus the *active* domain of any variable X has size at most N. The query C can be answered trivially in time $O(N^2)$ by joining R and S first, and then, for each triple (x,y,z) in the join, by verifying whether $(x,z) \notin T$ with a (hash) lookup. Suppose we know that the degree of relation T is less than two. Can we do better than $O(N^2)$ in that case? The answer is YES.

Rewriting to not-all-equal predicates

By viewing T as a bipartite graph of maximum degree two, it is easy to see that T can be written as a disjoint union of two relations $M_1(X,Z)$ and $M_2(X,Z)$ that represent *matchings* in the following sense: for any $i \in [2]$, if $(x,z) \in M_i$ and $(x',z') \in M_i$, then either $(x,z) = (x',z')$ or $x \neq x'$ and $z \neq z'$. Let $\mathsf{Dom}(Z)$ denote the active domain of the variable Z. Define, for each $i \in [2]$, a singleton relation $W_i(Z) \leftarrow \mathsf{Dom}(Z) \wedge \neg(\pi_Z M_i)(Z)$. Clearly, $|W_i| \leq N$ and given M_i, W_i can be computed in $O(N)$ preprocessing time. For each $i \in [2]$, create a new variable X_i with domain $\mathsf{Dom}(X_i) = \mathsf{Dom}(X)$. Then,

$$\neg M_i(X,Z) \equiv W_i(Z) \vee \exists X_i \big[M_i(X_i,Z) \wedge \mathsf{NAE}(X,X_i) \big]. \tag{8}$$

The predicate NAE stands for not-all-equal: It is the negation of the conjunction of pairwise equality on its variables. For arity two as in the rewriting of $\neg M_i(X,Z)$, $\mathsf{NAE}(X,X_i)$ stands for the disequality $X \neq X_i$.

From $T = M_1 \vee M_2$ and (8), we can rewrite the original query C from (7) into a disjunction of Boolean conjunctive queries without negated relations but with one or two extra existential variables that are involved in *disequalities* ($\neq$): $C \equiv \bigvee_{i \in [4]} C_i$, where

$$\begin{aligned}
C_1 &\leftarrow R(X,Y) \wedge S(Y,Z) \wedge W_1(Z) \wedge W_2(Z).\\
C_2 &\leftarrow R(X,Y) \wedge S(Y,Z) \wedge W_1(Z) \wedge M_2(X_2,Z) \wedge X \neq X_2.\\
C_3 &\leftarrow R(X,Y) \wedge S(Y,Z) \wedge W_2(Z) \wedge M_1(X_1,Z) \wedge X \neq X_1.\\
C_4 &\leftarrow R(X,Y) \wedge S(Y,Z) \wedge M_1(X_1,Z) \wedge M_2(X_2,Z) \wedge X \neq X_1 \wedge X \neq X_2.
\end{aligned}$$

[6] If R, S, and T would record direct train connections between cities, then this query would ask whether there exists a pair of cities with no direct train connection but with connections via another city.

It takes linear time to compute the matching decomposition of T into M_1 and M_2 since: (1) the relation T is a bipartite graph with degree at most two, and it is thus a union of even cycles and paths; and (2) we can traverse the cycles and paths and add alternative edges to M_1 and M_2. In general, when the maximum degree is higher and when T is not a binary predicate, Proposition 2.3 shows how to decompose a relation into high-dimensional matchings efficiently. The number of queries C_i depends exponentially on the arities and degrees of the negated relations.

Boolean tensor decomposition

The acyclic query C_1 can be answered in $O(N \log N)$ time using for instance InsideOut [2]; this algorithm first sorts the input relations in time $O(N \log N)$. The query C_2 can be answered as follows. Let $\forall i \in [\log N], f_i : \mathsf{Dom}(X) \to \{0,1\}$ denote the function such that $f_i(X)$ is the ith bit of X in its binary representation. Then, by noticing that

$$X \neq X_2 \equiv \bigvee_{b \in \{0,1\}} \bigvee_{i \in [\log N]} f_i(X) = b \wedge f_i(X_2) \neq b \tag{9}$$

we can break up the query C_2 into the disjunction of $2 \log N$ acyclic queries of the form

$$C_2^{b,i} \leftarrow R(X,Y) \wedge S(Y,Z) \wedge W_1(Z) \wedge M_2(X_2,Z) \wedge f_i(X) = b \wedge f_i(X_2) \neq b. \tag{10}$$

For a fixed b, both $f_i(X) = b$ and $f_i(X_2) \neq b$ are singleton relations on X and X_2, respectively. Then, C_2 can be answered in time $O(N \log^2 N)$. The same applies to C_3. We can use the same trick to answer C_4 in time $O(N \log^3 N)$. However, we can do better than that by observing that when viewed as a Boolean tensor in (9), the disequality tensor has the Boolean rank bounded by $O(\log N)$. In order to answer C_4 in time $O(N \log^2 N)$, we will show that the three-dimensional tensor $(X \neq X_1) \wedge (X \neq X_2)$ has the Boolean rank bounded by $O(\log N)$ as well. To this end, we extend the color-coding technique. We can further shave off a $\log N$ factor in the complexities of C_2, C_3, and C_4, as explained in Section 6.

Construction of the Boolean tensor decomposition

We next explain how to compute a tensor decomposition for the conjunction of disequalities in C_4. We show that there exists a family $\mathcal{F}$ of functions $f : \mathsf{Dom}(X) \to \{0,1\}$ satisfying the following conditions:

- **(i)** $|\mathcal{F}| = O(\log |\mathsf{Dom}(X)|) = O(\log N)$,
- **(ii)** For every triple $(x, x_1, x_2) \in \mathsf{Dom}(X)^3$ for which $x \neq x_1 \wedge x \neq x_2$, there is a function $f \in \mathcal{F}$ such that $f(x) \neq f(x_1) \wedge f(x) \neq f(x_2)$, and
- **(iii)** $\mathcal{F}$ can be constructed in time $O(N \log N)$.

We think of each function f as a "coloring" that assigns a "color" in $\{0,1\}$ to each element of $\mathsf{Dom}(X)$. Assuming (i) to (iii) hold, it follows that

$$X \neq X_1 \wedge X \neq X_2 \equiv \bigvee_{(c,c_1,c_2)} \bigvee_{f \in \mathcal{F}} f(X) = c \wedge f(X_1) = c_1 \wedge f(X_2) = c_2, \tag{11}$$

where (c, c_1, c_2) ranges over all triples in $\{0,1\}^3$ such that $c \neq c_1$ and $c \neq c_2$. Given this Boolean tensor decomposition, we can solve C_4 in time $O(N \log^2 N)$.

We prove (i) to (iii) using a combinatorial object called the *disjunct matrices*. These matrices are the central subject of combinatorial group testing [24, 12].

▶ **Definition 3.1** (k-disjunct matrix). *A $t \times N$ binary matrix $\boldsymbol{A} = (a_{ij})$ is called a* k-disjunct matrix *if for every column $j \in [N]$ and every set $S \subseteq [N]$ such that $|S| \leq k$ and $j \notin S$, there exists a row $i \in [t]$ for which $a_{ij} = 1$ and $a_{ij'} = 0$ for all $j' \in S$.*

It is known that for every integer $k < \sqrt{N}$, there exists a k-disjunct matrix (or equivalently a combinatorial group testing [24]) with $t = O\left(k^2 \log N\right)$ rows that can be constructed in time $O(k^2 N \log N)$ [31]. (If $k \geq \sqrt{N}$, we can just use the identity matrix.) In particular, for $N = |\mathsf{Dom}(X)|$ and $k = 2$, a 2-disjunct matrix $\boldsymbol{A} = (a_{ij})$ of size $O(\log N) \times N$ can be constructed in time $O(N \log N)$. From the matrix we define the function family $\mathcal{F}$ by associating a function f_i to each row i of the matrix, and every member $x \in \mathsf{Dom}(X)$ to a distinct column j_x of the matrix. Define $f_i(x) = a_{i,j_x}$ and (i)–(iii) straightforwardly follow.

4 Untangling bounded-degree relations

In this section we introduce a rewriting of queries of the form (1) into queries with so-called *not-all-equal* predicates, under the assumption that the relation $R_S(\boldsymbol{X}_S)$ for every hyperedge $S \in \overline{\mathcal{E}}$ has bounded degree $\deg(R_S)$.

▶ **Definition 4.1** (Not-all-equal). *Let $k \geq 2$ be an integer, and S be a set of k integers. The relation $\mathsf{NAE}_k(\boldsymbol{X}_S)$, or $\mathsf{NAE}(\boldsymbol{X}_S)$ for simplicity, holds true iff not all variables in $\boldsymbol{X}_S$ are equal:* $\mathsf{NAE}(\boldsymbol{X}_S) = \neg \bigwedge_{\{i,j\} \in \binom{S}{2}} X_i = X_j$.

The disequality ($\neq$) relation is exactly NAE_2. The negation of a matching is connected to NAE predicates as follows.

▶ **Proposition 4.2.** *Let $M(\boldsymbol{X}_S)$ be a k-ary matching, where $k = |S| \geq 2$. For any $i, j \in S$, define the unary relation $W_i(X_i) \leftarrow \mathsf{Dom}(X_i) \wedge \neg(\pi_i M)(X_i)$ and the binary relation $M_{ij} = \pi_{i,j} M$. For any $\ell \in S$, it holds that*

$$\neg M(\boldsymbol{X}_S) \equiv \left(\bigvee_{i \in S \setminus \{\ell\}} W_i(X_i) \right) \vee \exists \boldsymbol{Y}_{S \setminus \{\ell\}} \left[\mathsf{NAE}(X_\ell, \boldsymbol{Y}_{S \setminus \{\ell\}}) \wedge \bigwedge_{j \in S \setminus \{\ell\}} M_{\ell j}(Y_j, X_j) \right]. \quad (12)$$

Proof. The intuition for this rewriting is as follows. A value $x_i \in \mathsf{Dom}(X_i)$ occurs in at most one tuple in the matching M. Therefore, any value in a tuple determines the rest of the tuple. The rewriting in (12) first turns every tuple in M into a tuple whose values are all the same, i.e., all-equal values. The negation of M consists of tuples with at least two different values, i.e., not-all-equal values.

We next prove that the rewriting is correct.

In one direction, consider a tuple $\boldsymbol{x}_S \notin M$, i.e., $\neg M(\boldsymbol{x}_S)$ holds, and suppose $x_i \notin W_i$ for all $i \in S \setminus \{\ell\}$. This means, for every $i \in S \setminus \{\ell\}$, there is a unique tuple $\boldsymbol{t}^{(i)} = (t_j^{(i)})_{j \in S} \in M$ such that $x_i = t_i^{(i)}$. Define $y_j = t_\ell^{(j)}$ for all $j \in S \setminus \{\ell\}$. The tuple $\boldsymbol{y}_{S \setminus \{\ell\}}$ satisfies $(y_j, x_j) = (t_\ell^{(j)}, t_j^{(j)}) \in M_{\ell j}$, for all $j \in S \setminus \{\ell\}$. Moreover, one can verify that $\mathsf{NAE}(x_\ell, \boldsymbol{y}_{S \setminus \{\ell\}})$ holds. In particular, if $y_j = x_\ell$ for all $j \in S \setminus \{\ell\}$, then all tuples $\boldsymbol{t}^{(j)} \in M$ are the same tuple (since M is a matching) and that tuple is $\boldsymbol{x}_S$. Hence $\boldsymbol{x}_S \in M$ which is a contradiction.

Conversely, suppose there exists a tuple $(\boldsymbol{x}_S, \boldsymbol{y}_{S \setminus \{\ell\}})$ satisfying the right hand side of (12). If $x_i \in W_i$ for any $i \in S \setminus \{\ell\}$, then $\boldsymbol{x}_S \notin M$, i.e., $\boldsymbol{x}_S$ satisfies the left hand side of (12). Now, suppose $x_i \notin W_i$ for all $i \in S \setminus \{\ell\}$. Suppose to the contrary that $\boldsymbol{x}_S \in M$. Then, for all $j \in S \setminus \{\ell\}$ we have $y_j = x_\ell$ since $M_{\ell j}(y_j, x_j)$ must hold. This means that $\mathsf{NAE}(x_\ell, \boldsymbol{y}_{S \setminus \{\ell\}}) = \neg \wedge_{j \in S \setminus \{\ell\}} x_\ell = y_j$ does not hold. This contradicts our hypothesis. ◀

We use the connection to NAE predicates to decompose a query containing a conjunction of negated bounded-degree relations into a disjunction of positive terms, as given next by Proposition 4.3. We call this rewriting *untangling*.

Let fhtw_F and subw_F denote the fractional hypertree width and respectively the submodular width of the conjunctive query $Q(\boldsymbol{X}_F) \leftarrow \mathsf{body}$ (These notions are defined in Section 2.2).

▶ **Proposition 4.3.** *Let Q be the query defined in Eq. (1): $Q(\boldsymbol{X}_F) \leftarrow \mathit{body} \wedge \bigwedge_{S\in\overline{\mathcal{E}}} \neg R_S(\boldsymbol{X}_S)$. We can compute in linear time a collection of B hypergraphs $\mathcal{H}_i = (\mathcal{V}_i, \mathcal{E}_i)$ such that*

$$Q(\boldsymbol{X}_F) \equiv \bigvee_{i\in[B]} Q_i(\boldsymbol{X}_F), \qquad \text{where } \forall i \in [B] : Q_i(\boldsymbol{X}_F) \leftarrow \mathit{body}_i \wedge \bigwedge_{S\in\mathcal{E}_i} \mathsf{NAE}(\boldsymbol{Z}_S), \tag{13}$$

and body_i is the body of a conjunctive query satisfying

$$\mathsf{fhtw}_F(\mathit{body}_i) \le \mathsf{fhtw}_F(\mathit{body}), \qquad \text{and} \qquad \mathsf{subw}_F(\mathit{body}_i) \le \mathsf{subw}_F(\mathit{body}).$$

Furthermore, the number B of queries is bounded by $B \le \prod_{S\in\overline{\mathcal{E}}} (|S|)^{|S|(\deg(R_S)-1)+1}$.

Proof. From Proposition 2.3, each relation $R_S(\boldsymbol{X}_S)$ can be written as a disjoint union of $D_S \le |S|(\deg(R_S)-1)+1$ matchings M_S^ℓ, $\ell \in [D_S]$. These matchings can be computed in linear time. Hence, the second half of the body of query Q can be rewritten equivalently as

$$\bigwedge_{S\in\overline{\mathcal{E}}} \neg R_S(\boldsymbol{X}_S) \equiv \bigwedge_{S\in\overline{\mathcal{E}}} \neg \bigvee_{\ell\in[D_S]} M_S^\ell(\boldsymbol{X}_S) \equiv \bigwedge_{\substack{S\in\overline{\mathcal{E}}\\ \ell\in[D_S]}} \neg M_S^\ell(\boldsymbol{X}_S).$$

To simplify notation, let $\overline{\mathcal{E}}_1$ denote the multiset of edges obtained from $\overline{\mathcal{E}}$ by duplicating the edge $S \in \overline{\mathcal{E}}$ exactly D_S times. Furthermore, for the ℓ-th copy of S, associate the matching M_S^ℓ with the copy of S in $\overline{\mathcal{E}}_1$; use M_S to denote the matching corresponding to that copy. Then, we can write Q equivalently $Q(\boldsymbol{X}_F) \leftarrow \mathsf{body} \wedge \bigwedge_{S\in\overline{\mathcal{E}}_1} \neg M_S(\boldsymbol{X}_S)$.

For each $S \in \overline{\mathcal{E}}_1$, fix an arbitrary integer $\ell_S \in S$. From Proposition 4.2, the negation of M_S can be written as

$$\neg M_S(\boldsymbol{X}_S) \equiv \Big(\bigvee_{i\in S\setminus\{\ell_S\}} W_i^S(X_i)\Big) \vee \exists \boldsymbol{Y}^S_{S\setminus\{\ell_S\}} \Big[\bigwedge_{j\in S\setminus\{\ell_S\}} (\pi_{\ell_S,j} M_S)(Y_j^S, X_j) \wedge \mathsf{NAE}(X_{\ell_S}, \boldsymbol{Y}^S_{S\setminus\{\ell_S\}})\Big],$$

where W_i^S is a unary relation on variable X_i, and $\boldsymbol{Y}^S_{S\setminus\{\ell_S\}} = (Y_i^S)_{i\in S\setminus\{\ell_S\}}$ is a tuple of fresh variables, only associated with (the copy of) S. In particular, if S and S' are two distinct items in the multiset $\overline{\mathcal{E}}_1$, then Y_i^S and $Y_i^{S'}$ are two distinct variables.

Each negated term $\neg M_S(\boldsymbol{X}_S)$ is thus expressed as a disjunction of $|S|$ positive terms. We can then express the conjunction of $|\overline{\mathcal{E}}_1|$ negated terms as the disjunction of $\prod_{S\in\overline{\mathcal{E}}_1} |S|$ conjunctions. For this, define a collection of tuples $\mathcal{T} = \prod_{S\in\overline{\mathcal{E}}_1} S$. In particular, every member $\boldsymbol{T} \in \mathcal{T}$ is a tuple $\boldsymbol{T} = (t_S)_{S\in\overline{\mathcal{E}}_1}$ where $t_S \in S$. The second half of the body of query Q can be rewritten equivalently as

$$\begin{aligned}
&\bigwedge_{S\in\overline{\mathcal{E}}} \neg R_S(\boldsymbol{X}_S) \equiv \bigwedge_{S\in\overline{\mathcal{E}}_1} \neg M_S(\boldsymbol{X}_S)\\
&\equiv \bigwedge_{S\in\overline{\mathcal{E}}_1} \left(\bigvee_{i\in S\setminus\{\ell_S\}} W_i^S(X_i) \vee \exists \boldsymbol{Y}^S_{S\setminus\{\ell_S\}} \Big[\bigwedge_{j\in S\setminus\{\ell_S\}} (\pi_{\ell_S,j} M_S)(Y_j^S, X_j) \wedge \mathsf{NAE}\big(X_{\ell_S}, \boldsymbol{Y}^S_{S\setminus\{\ell_S\}}\big)\Big] \right)\\
&\equiv \bigvee_{\boldsymbol{T}\in\mathcal{T}} \bigwedge_{\substack{S\in\overline{\mathcal{E}}_1\\ t_S\neq\ell_S}} W_{t_S}^S(X_{t_S}) \wedge \bigwedge_{\substack{S\in\overline{\mathcal{E}}_1\\ t_S=\ell_S}} \exists \boldsymbol{Y}^S_{S\setminus\{\ell_S\}} \Big[\bigwedge_{j\in S\setminus\{\ell_S\}} (\pi_{\ell_S,j} M_S)(Y_j^S, X_j) \wedge \mathsf{NAE}\big(X_{\ell_S}, \boldsymbol{Y}^S_{S\setminus\{\ell_S\}}\big)\Big]
\end{aligned}$$

The original query Q is equivalent to the disjunction

$$Q(\boldsymbol{X}_F) \equiv \bigvee_{\boldsymbol{T}\in\mathcal{T}} Q_{\boldsymbol{T}}(\boldsymbol{X}_F)$$

of up to $\prod_{S\in\overline{\mathcal{E}}_1}|S|$ queries $Q_{\boldsymbol{T}}$ defined by

$$\underbrace{\mathsf{body} \wedge \bigwedge_{\substack{S\in\overline{\mathcal{E}}_1\\ t_S\neq\ell_S}} W^S_{t_S}(X_{t_S}) \wedge \bigwedge_{\substack{S\in\overline{\mathcal{E}}_1\\ t_S=\ell_S\\ j\in S\setminus\{\ell_S\}}} (\pi_{\ell_S,j}M_S)(Y^S_j, X_j)}_{\mathsf{body}_i} \wedge \bigwedge_{\substack{S\in\overline{\mathcal{E}}_1\\ t_S=\ell_S}} \mathsf{NAE}\big(X_{\ell_S}, \boldsymbol{Y}^S_{S\setminus\{\ell_S\}}\big) \tag{14}$$

In the above definition of $Q_{\boldsymbol{T}}$, let us denote all but the last conjunction of NAE predicates by body_i. It holds that $\mathsf{fhtw}_F(\mathsf{body}_i) \le \mathsf{fhtw}_F(\mathsf{body})$, and $\mathsf{subw}_F(\mathsf{body}_i) \le \mathsf{subw}_F(\mathsf{body})$ [1]. We now turn to the conjunction of NAE predicates in (14). Since each $S \in \overline{\mathcal{E}}$ is repeated at most $|S|(\deg(R_S)-1)+1$ times in $\overline{\mathcal{E}}_1$, it follows that the number $\prod_{S\in\overline{\mathcal{E}}_1}|S|$ of conjunctive queries $Q_{\boldsymbol{T}}$ is at most $\prod_{S\in\overline{\mathcal{E}}}|S|^{|S|(\deg(R_S)-1)+1}$. ◀

5 Boolean tensor decomposition

Thanks to the untangling result in Proposition 4.3, we only need to concentrate on answering queries of the form (13). To deal with the conjunction of NAE predicates, this section describes the construction of a Boolean tensor decomposition of a conjunction $\bigwedge_{S\in\mathcal{A}}\mathsf{NAE}(\boldsymbol{X}_S)$ of NAE predicates. The multi-hypergraph of this conjunction has the query variables as vertices and the NAE predicates as hyperedges.

▶ **Lemma 5.1.** *Let $\mathcal{G}=(U,\mathcal{A})$ be the multi-hypergraph of a conjunction $\bigwedge_{S\in\mathcal{A}}\mathsf{NAE}(\boldsymbol{X}_S)$, N an upper bound on the domain sizes for variables $(X_i)_{i\in U}$, and c a positive integer. Suppose there exists a family $\mathcal{F}$ of functions $f:[N]\to[c]$ satisfying the following property*

$$\text{for any proper } N\text{-coloring } h: U\to[N] \text{ of } \mathcal{G} \text{ there exists a function } f\in\mathcal{F} \text{ such that } f\circ h \text{ is a proper } c\text{-coloring of } \mathcal{G}. \tag{15}$$

Then, the following holds:

$$\bigwedge_{S\in\mathcal{A}}\mathsf{NAE}(\boldsymbol{X}_S) \equiv \bigvee_{g}\bigvee_{f\in\mathcal{F}}\bigwedge_{i\in U} f(X_i)=g(i), \tag{16}$$

where g ranges over all proper c-colorings of $\mathcal{G}$. In particular, the Boolean tensor rank of the left-hand side of (16) is bounded by $r=P(\mathcal{G},c)\cdot|\mathcal{F}|$.

Proof. Let $\boldsymbol{x}_U$ denote any tuple satisfying the LHS of (16). Define $h:U\to[N]$ by setting $h(i)=x_i$. Then h is a proper N-coloring of $\mathcal{G}$, which means there exists $f\in\mathcal{F}$ such that $g=f\circ h$ is a proper c-coloring of $\mathcal{G}$. Then the conjunct on the RHS corresponding to this particular pair (g,f) is satisfied.

Conversely, let $\boldsymbol{x}_U$ denote any tuple satisfying the RHS of (16). Then, there is a pair (g,f) whose corresponding conjunct on the RHS of (16) is satisfied, i.e., $f(x_i)=g(i)$ for all $i\in U$. Recall that g is a proper c-coloring of $\mathcal{G}$. If there exists $S\in\mathcal{A}$ such that $\mathsf{NAE}(\boldsymbol{x}_S)$ does *not* hold, then $x_i=x_j$ for all $i,j\in S$, implying $g(i)=f(x_i)=f(x_j)=g(j)$ for all $i,j\in S$, contradicting the fact that g is a proper coloring.

For the Boolean tensor rank statement, note that (16) is a Boolean tensor decomposition of the formula $\bigwedge_{S\in\mathcal{A}}\mathsf{NAE}(\boldsymbol{X}_S)$, because $f(X_i)=g(i)$ is a unary predicate on variable X_i. This predicate is of size bounded by N. ◀

To explain how Lemma 5.1 can be applied, we exemplify two techniques, showing the intimate connections of our Boolean tensor decomposition problem to combinatorial group testing and perfect hashing.

▶ **Example 5.2** (Connection to group testing)**.** Consider the case when the graph $\mathcal{G}$ is a *k-star*, i.e., a tree with a center vertex and k leaf vertices. Let $\boldsymbol{A}$ be a $O(k^2 \log N) \times N$ binary k-disjunct matrix, which can be constructed in time $O(kN \log N)$ (This is due to known results on k-restriction and error codes, recalled in the extended paper [1]). We can assume $k < \sqrt{N}$ to avoid triviality. Consider a family $\mathcal{F}$ of functions $f : [N] \to \{0, 1\}$ constructed as follows: there is a function f for every row i of $\boldsymbol{A}$, where $f(j) = a_{ij}$, for all $j \in [N]$. The family $\mathcal{F}$ has size $O(k^2 \log N)$. We show that $\mathcal{F}$ satisfies condition (15). Let $h : U \to [N]$ denote any coloring of the star. Let $j \in [N]$ be the color h assigns to the center, and S be the set of colors assigned to the leaf nodes. Clearly $j \notin S$. Hence, there is a function $f \in \mathcal{F}$ for which $f(j) = 1$ and $f(j') = 0$ for all $j' \in S$, implying $f \circ h$ is a proper 2-coloring of $\mathcal{G}$.

A consequence of our observation is that for a k-star $\mathcal{G}$ the conjunction $\bigwedge_{S \in \mathcal{A}} \mathsf{NAE}(\boldsymbol{X}_S)$ has Boolean rank bounded by $O(k^2 \log N)$.

▶ **Example 5.3** (Connection to perfect hashing)**.** Consider now the case when the graph $\mathcal{G}$ is a k-clique. Let $\mathcal{F}$ denote any (N, c, k)-perfect hash family, i.e., a family of hash functions from $[N] \to [c]$ such that for every subset $S \subseteq [N]$ of size k, there is a function f in the family for which its image is also of size k. It is easy to see that this hash family satisfies (15). From [6], it is known that we can construct in polytime an (N, k^2, k)-perfect hash family of size $O(k^4 \log N)$. However, it is not clear what the runtime exponent of their construction is. What we need for our application is that the construction should run in linear data complexity and in polynomial query complexity. We use a result from [31] to exhibit such a construction in Theorem 5.6; furthermore, our hash family has size only $O(k^2 \log N)$.

We next construct the smallest family $\mathcal{F}$ satisfying Lemma 5.1. We first bound the size of $\mathcal{F}$ using the probabilistic method [7] and then specify how to derandomize the probabilistic construction of $\mathcal{F}$ to obtain a deterministic algorithm. For this, we need some terminology.

Every coloring $h : U \to [N]$ of $\mathcal{G} = (U, \mathcal{A})$ induces a homomorphic image $h(\mathcal{G}) = (h(U), h(\mathcal{A}))$, which is the graph on vertex set $h(U)$ and edge set $h(\mathcal{A})$ defined by

$$h(U) = \{h(v) \mid v \in U\} \subseteq [N], \qquad h(\mathcal{A}) = \big\{h(S) = \{h(v) \mid v \in S\} \mid S \in \mathcal{A}\big\} \subseteq 2^{[N]}.$$

Here, we overload notation to allow h range over sets and graphs. Let $\mathsf{col}(\mathcal{G}, N)$ denote the set of proper N-colorings h of $\mathcal{G}$. Each such proper N-coloring is a homomorphic image of $\mathcal{G}$. Define c as the maximum chromatic number over all homomorphic images of $\mathcal{G}$: $c = \max_{h \in \mathsf{col}(\mathcal{G}, N)} \chi(h(\mathcal{G}))$. For a given $h \in \mathsf{col}(\mathcal{G}, N)$, let $g : h(U) \to [c]$ be a proper c-coloring of $h(\mathcal{G})$. The *multiplicity* of a color $i \in [c]$ is the number of vertices colored i by g. The *signature* of g is the vector $\boldsymbol{\mu}(g) = (\mu_i)_{i \in [c]}$, where μ_i is the multiplicity of color i. Let $T_c(h)$ denote the collection of all signatures of proper c-colorings of $h(\mathcal{G})$. For a given signature $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_c) \in T_c(h)$, let $n(\boldsymbol{\mu}, h)$ denote the number of proper c-colorings of $h(\mathcal{G})$ whose signature is $\boldsymbol{\mu}$.

▶ **Example 5.4.** Suppose $\mathcal{G}$ is the k-clique and $c = k$. Then, every proper k-coloring of $h(\mathcal{G})$ has signature $\boldsymbol{\mu} = \mathbf{1}_k = (1, 1, \ldots, 1)$: $T_c(h)$ has only one member, but $n(\mathbf{1}, h) = k!$. If $\mathcal{G}$ is the k-star then $c = 2$ and for any $h \in \mathsf{col}(\mathcal{G}, N)$, $h(\mathcal{G})$ is an ℓ-star for some $\ell \in [k]$. Then, $T_2(h)$ has two signatures: $\boldsymbol{\mu} = (\ell, 1)$ and $\boldsymbol{\mu}' = (1, \ell)$; furthermore, $n(\boldsymbol{\mu}, h) = n(\boldsymbol{\mu}', h) = 1$.

▶ **Definition 5.5** (Strongly Explicit Construction)**.** *A family $\mathcal{F}$ of functions $f : [N] \to [c]$ is said to be* strongly explicit *if there is an algorithm that, given an index to a function f in $\mathcal{F}$ and a number $j \in [N]$, returns $f(j)$ in $\mathsf{poly}(\log |\mathcal{F}|, \log N)$-time.*

The next theorem gives two upper bounds on the size of a family of hash functions satisfying (15) that we use to define the rank of our Boolean tensor decomposition: The first bound is for such families in general, whereas the second is for strongly explicit families that we can use effectively.

▶ **Theorem 5.6.** *Let $\mathcal{G} = (U, \mathcal{A})$ be a multi-hypergraph, $c = \max_{h \in \mathsf{col}(\mathcal{G},N)} \chi(h(\mathcal{G}))$, and $\boldsymbol{p} = (p_1, \ldots, p_c) \in \mathbb{R}^c_+$ be a fixed non-negative real vector such that $\|\boldsymbol{p}\|_1 = 1$. Define*

$$\theta(\boldsymbol{p}) = \min_{h \in \mathsf{col}(\mathcal{G},N)} \sum_{\boldsymbol{\mu} \in T_c(h)} n(\boldsymbol{\mu}, h) \prod_{i=1}^{c} p_i^{\mu_i} \tag{17}$$

Then, the following hold:

(a) *There exists a family $\mathcal{F}$ of functions $f : [N] \to [c]$ satisfying (15) such that*

$$|\mathcal{F}| \le \left\lceil \frac{\ln P(\mathcal{G}, N)}{\theta(\boldsymbol{p})} \right\rceil \le \frac{|U| \log N}{\theta(\boldsymbol{p})}. \tag{18}$$

(b) *There is a strongly explicit family $\mathcal{F}'$ of functions $f : [N] \to [c]$ satisfying (15) such that*

$$|\mathcal{F}'| = O\left(\frac{|U|^3 \cdot \log |U| \cdot \log N}{\theta(\boldsymbol{p})} \right). \tag{19}$$

The next corollary follows immediately from Lemma 5.1 and Theorem 5.6.

▶ **Corollary 5.7.** *Let $\mathcal{G} = (U, \mathcal{A})$ be a multi-hypergraph, $c = \max_{h \in \mathsf{col}(\mathcal{H},N)} \chi(h(\mathcal{G}))$, and*

$$\theta^* = \max_{\boldsymbol{p}: \|\boldsymbol{p}\|_1 = 1, \boldsymbol{p} \ge \boldsymbol{0}} \theta(\boldsymbol{p}), \tag{20}$$

where $\theta(\boldsymbol{p})$ is defined in (17). The following hold:

(a) *The Boolean rank of the function $\bigwedge_{F \in \mathcal{A}} \mathsf{NAE}(\boldsymbol{X}_F)$ is upper bounded by $\frac{P(\mathcal{G},c) \cdot \ln P(\mathcal{G},N)}{\theta^*}$.*

(b) *Given $\boldsymbol{p}$, there is a strongly explicit Boolean tensor decomposition of $\bigwedge_{F \in \mathcal{A}} \mathsf{NAE}(\boldsymbol{X}_F)$ whose rank is upper bounded by $P(\mathcal{G}, c) \cdot \frac{|U|^3 \cdot \log |U| \cdot \log N}{\theta(\boldsymbol{p})}$.*

To apply the above result, we need to specify $\boldsymbol{p}$ to maximize $\theta(\boldsymbol{p})$. We do not know how to compute the optimizer $\boldsymbol{p}^*$ in closed form. We next discuss several observations that allow us to bound θ^* from below or compute it exactly. In the following, for any tuple $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_\ell)$ of positive integers, let $K_{\boldsymbol{\mu}}$ denote the complete ℓ-partite graph defined as follows. For every $i \in [\ell]$ there is an independent set I_i of size μ_i. All independent sets are disjoint. The vertex set is $\bigcup_{i \in [\ell]} I_i$ and the vertices not belonging to the same independent set are connected. Without loss of generality, we assume $\mu_1 \ge \cdots \ge \mu_\ell$ when specifying the graph $K_{\boldsymbol{\mu}}$. For example, $K_{\boldsymbol{1}_k}$ is the k-clique, and $K_{(k,1)}$ is the k-star.

▶ **Proposition 5.8.** *The following hold:*

(a) *Given a multi-hypergraph $\mathcal{G} = (U, \mathcal{A})$ with $|U| = k$ and $c = \max_{h \in \mathsf{col}(\mathcal{G},N)} \chi(h(\mathcal{G}))$, it holds that $\theta^* \ge \frac{1}{c^c} \ge \frac{1}{k^k}$.*

(b) *Suppose $\mathcal{G} = K_{\boldsymbol{\mu}}$ for some positive integer tuple $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_\ell)$, where $\mu_1 \ge \cdots \ge \mu_\ell \ge 1$. Let S_ℓ denote the set of all permutations of $[\ell]$, and $\mathsf{FP}(\boldsymbol{\mu})$ denote the number of permutations $\pi \in S_\ell$ for which $\mu_i = \mu_{\pi(i)}, \forall i \in [\ell]$. Then,*

$$\theta^* = \max_{\boldsymbol{p}} \sum_{\pi \in S_\ell} \prod_{i=1}^{\ell} p_i^{\mu_{\pi(i)}} \ge \sum_{\pi \in S_\ell} \prod_{i=1}^{\ell} \left(\frac{\mu_i}{\|\boldsymbol{\mu}\|_1} \right)^{\mu_{\pi(i)}} \ge \mathit{FP}(\boldsymbol{\mu}) \prod_{i=1}^{\ell} \left(\frac{\mu_i}{\|\boldsymbol{\mu}\|_1} \right)^{\mu_i}. \tag{21}$$

▶ **Corollary 5.9.** *Let $\ell \in [k]$ be an integer. Let $\boldsymbol{\mu} = (k-\ell, \mathbf{1}_\ell)$. Then, when $\mathcal{G} = K_{\boldsymbol{\mu}}$ we have*

$$\theta^* \geq \frac{\ell!}{k^\ell}\left(\frac{k-\ell}{k}\right)^{k-\ell} \geq \frac{\ell!}{e^\ell}\frac{1}{k^\ell},$$

where $e = 2.7..$ is the base of the natural log. In particular, $\mathcal{G}$ is a $(k-1)$-star when $\ell = 1$ and the bound is $\theta^* \geq \frac{1}{ek}$. When $\ell = k$, then $\mathcal{G}$ is a k-clique and the bound is $\theta^* = k!/k^k$.

For any constant $\ell \in [k]$, the bound for θ^* is $\Omega(1/k^\ell)$; in particular, the lower bound for θ^* ranges anywhere between $\Omega(1/k)$, $\Omega(1/k^2)$, up to $\Omega(k!/k^k)$. There is a spectrum of these bounds, leading to a spectrum of Boolean tensor ranks for our decomposition.

▶ **Example 5.10.** From (18) and the above corollary, it follows that when $\mathcal{G}$ is a k-star, the corresponding Boolean rank is bounded by $O(k^2 \log N)$, matching the group testing connection from Example 5.2. The reason is twofold. We need two colors to color a k-star and the chromatic polynomial of a k-star using two colors is two. The size of the family $\mathcal{F}$ of hash functions is upper bounded by $\frac{|U|\log N}{\theta^*}$ where θ^* is at least $\frac{1}{ek}$ and $|U| = k+1$. Then, $|\mathcal{F}| \leq e \cdot k \cdot (k+1)\log N = O(k^2 \log N)$. This matches the tailor-made construction from Example 5.2. However, our strongly explicit construction in Theorem 5.6(b) yields a slightly larger Boolean tensor decomposition of rank $O(k^4 \log k \log N)$.

When applying part (b) of Proposition 5.8 to the problem of detecting k-paths in a graph, i.e., the query P in the introduction, we obtain the Boolean rank $O(\frac{k^{k+3}}{k!} \cdot \log k \cdot \log N)$. This is because (1) we would need two colors and the chromatic polynomial for the k-path hypergraph using two colors is two, and (2) the size of the family of strongly explicit functions is $O(\frac{(k+1)^3 \log(k+1) \log N}{\theta^*})$ with $\theta^* = k!/k^k$.

6 How to use the tensor decomposition

Sections 4 and 5 introduced two rewriting steps. The first step transforms a conjunctive query with negation of the form (1) into a disjunction of conjunctive queries with NAE predicates of the form (13). The second step transforms a conjunction of NAE predicates into a disjunction of conjunctions of one-variable-conditions of the form (16). The first step exploited the bounded degrees of the negated relations to bound from above the number of disjuncts and independently of the database size. The second step uses a generalization of the color-coding technique to further rewrite a conjunction of NAE predicates into a Boolean tensor decomposition whose rank depends on the structure of the multi-hypergraph of the conjunction. Both rewriting steps preserve the equivalence of the queries.

In this section, we show that the query obtained after the two rewriting steps can be evaluated efficiently. This query has the form $Q(\boldsymbol{X}_F) \leftarrow \bigvee_{j\in[B]} Q_j(\boldsymbol{X}_F)$ where $\forall j \in [B]$:

$$Q_j(\boldsymbol{X}_F) \leftarrow \bigvee_{g\in \mathsf{col}(\mathcal{G}_j, c_j)} \bigvee_{f\in\mathcal{F}_j} \underbrace{\left[\bigwedge_{S\in\mathcal{E}_j} R_S(\boldsymbol{X}_S) \wedge \bigwedge_{i\in U_j} f(X_i) = g(i)\right]}_{Q_j^{(g,f)}(\boldsymbol{X}_F)} \tag{22}$$

In particular, we will show that the data complexity of any conjunctive query with negation of the form (1) is the same as for its positive subquery $Q(\boldsymbol{X}_F) \leftarrow \mathsf{body}$.

The subsequent development in this section uses the InsideOut algorithm and the FAQ framework (see Section 2.2 and [2]). For each $j \in [B]$, we distinguish two multi-hypergraphs for the query $Q_j(\boldsymbol{X}_F)$: $\mathcal{H}_j = (V_j, \mathcal{E}_j)$ and associated relations $(R_S)_{S\in\mathcal{E}_j}$ for $\bigwedge_{S\in\mathcal{E}_j} R_S(\boldsymbol{X}_S)$;

and $\mathcal{G}_j = (U_j, \mathcal{A}_j)$ for $\bigwedge_{i \in U_j} f(X_i) = g(i)$, where $U_j \subseteq V_j$. For the rest of this section, we will fix some $j \in [B]$ and drop the subscript j for brevity. In particular, we will use $\mathcal{H} = (V, \mathcal{E}), \mathcal{G} = (U, \mathcal{A}), \mathcal{F}$ to denote $\mathcal{H}_j = (V_j, \mathcal{E}_j), \mathcal{G}_j = (U_j, \mathcal{A}_j), \mathcal{F}_j$ respectively.

A better semiring for shaving off a $\log N$ factor

Let $r = P(\mathcal{G}, c) \cdot |\mathcal{F}|$ denote the Boolean tensor rank in the decomposition (16). If we were only interested in bounding the rank, we can use the bound on $|\mathcal{F}|$ from Part (a) of Theorem 5.6. However, for the purpose of using the Boolean tensor decomposition in an algorithm, we have to be able to explicitly and efficiently construct the family $\mathcal{F}$ of functions. We thus need to use the bound on $|\mathcal{F}|$ from Part (b) of Theorem 5.6. To facilitate the explanations below, define $w = |\mathcal{F}|/\log N$ so that the Boolean rank is decomposed into $r = P(\mathcal{G}, c) \cdot w \cdot \log N$; that is, $w = \frac{|U|^3 \cdot \log |U|}{\theta(\boldsymbol{p})}$ from Part (b) of Theorem 5.6.

By Theorem 2.6, we can answer query (22) by running r instantiations of InsideOut, each of which computes $Q_j^{(g,f)}$ for some fixed pair (g, f), and then take the disjunction of $Q_j^{(g,f)}$ over g and f. The runtime is

$$O(P(\mathcal{G}, c) \cdot w \cdot (|\mathcal{E}| + |U|) \cdot |V|^2 \cdot (\log N)^2 \cdot (N^{\mathsf{fhtw}_F(\mathcal{H})} + |\mathsf{output}|)). \tag{23}$$

The atoms $f(X_i) = g(i)$ are singleton factors, i.e., factors on one variable, and thus do not increase the fractional hypertree width or the submodular width of the query.

These r instantiations of InsideOut are run on sum-product instances over the Boolean semiring. We can however reformulate the problem as sum-product over a different semiring, which helps reduce the runtime. The new semiring $(\boldsymbol{D}, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ is defined as follows. The domain $\boldsymbol{D}$ is set to $\boldsymbol{D} = \{0,1\}^r$, the collection of all r-bit vectors. The "addition" and "multiplication" operators $\oplus$ and $\otimes$ are bit-wise max and min (essentially, bit-wise $\vee$ and $\wedge$). The additive identity is $\mathbf{0} = \mathbf{0}_r$, the r-bit all-0 vector. The multiplicative identity is $\mathbf{1} = \mathbf{1}_r$, the r-bit all-1 vector. To each input relation R_S, we associate a function $\psi_S(\boldsymbol{X}_S) : \prod_{i \in S} \mathsf{Dom}(X_i) \to \boldsymbol{D}$, where $\psi_S(\boldsymbol{x}_S) = \mathbf{1}$ if $\boldsymbol{x}_S \in R_S$ and $\mathbf{0}$ otherwise. Also, define $|U|$ extra singleton factors $\overline{\psi}_i : \mathsf{Dom}(X_i) \to \boldsymbol{D}$ $(\forall i \in U)$, where

$$\overline{\psi}_i(x_i) = (b_{g,f})_{g \in \mathsf{col}(\mathcal{G},c), f \in \mathcal{F}}, \quad \text{where} \quad b_{g,f} = \begin{cases} 1 & \text{if } f(x_i) = g(i) \\ 0 & \text{if } f(x_i) \neq g(i). \end{cases} \tag{24}$$

▶ **Proposition 6.1.** *The query* (22) *is equivalent to the following* SumProd *expression*

$$\varphi(\boldsymbol{x}_F) = \bigoplus_{x_{|F|+1}} \cdots \bigoplus_{x_{|V|}} \bigotimes_{S \in \mathcal{E}} \psi_S(\boldsymbol{x}_S) \otimes \bigotimes_{i \in U} \overline{\psi}_i(x_i). \tag{25}$$

The runtime of InsideOut *for the expression* $\varphi(\boldsymbol{x}_F)$ *is*

$$O(P(\mathcal{G}, c) \cdot w \cdot (|\mathcal{E}| + |U|) \cdot |V|^2 \cdot \log N \cdot (N^{\mathsf{fhtw}_F(\mathcal{H})} + |\mathsf{output}|)). \tag{26}$$

Proof. For any $\boldsymbol{x}_F$, we have $Q(\boldsymbol{x}_F) = \mathsf{true}$ iff $\varphi(\boldsymbol{x}_F) \neq \mathbf{0}$. This is because for each $\boldsymbol{x}_F$, the value $\varphi(\boldsymbol{x}_F) \in \mathbf{D}$ is an r-bit vector where each bit represents the answer to $Q_j^{(g,f)}(\boldsymbol{X}_F)$ for some pair (g, f) (There are exactly $r = P(\mathcal{G}, c) \cdot |\mathcal{F}|$ such pairs).

The runtime of InsideOut follows from the observation that each operation $\oplus$ or $\otimes$ can be done in $O(r/\log N)$-time, because those are bit-wise $\vee$ or $\wedge$ and the r-bit vector can be stored in $O(r/\log N)$ words in memory. Bit-wise $\vee$ and $\wedge$ of two words are done in $O(1)$-time. ◀

We can further lower the data complexity of our approach using PANDA (See Section 2.2 and [3]). By Theorem 2.7, the complexity from (26) becomes:

$$O(P(\mathcal{G}, c) \cdot w \cdot |V| \cdot 2^{2^{|V|}} \cdot (\mathsf{poly}(\log N) \cdot N^{\mathsf{subw}_F(\mathcal{H})} + \log N \cdot |\mathsf{output}|)). \tag{27}$$

The data complexity for conjunctive queries with negation

We are now ready to prove Theorem 1.1. From Proposition 4.3, we untangle Q into a disjunction of B different queries Q_j for $j \in [B]$ of the form (13) where $B \leq \prod_{S \in \overline{\mathcal{E}}} (|S|)^{|S|(d_S-1)+1} = O(1)$ in data complexity. From (16), each of these queries is equivalent to query (22). For a fixed $g \in \mathsf{col}(\mathcal{G}, c)$ and $f \in \mathcal{F}$, the inner conjunction $Q_j^{(g,f)}(\boldsymbol{X}_F)$ in (22) has at most the widths fhtw_F and subw_F of body in the original query (1). Query (22) can be solved in time (26) using InsideOut or (27) using PANDA.

7 Related Work

Color-coding. The color-coding technique [8] underlies existing approaches to answering queries with disequalities [27, 9, 21], the homomorphic embedding problem [14], and motif finding and counting in computational biology [4]. This technique has been originally proposed for checking cliques of inequalities. It is typically used in conjunction with a dynamic programming algorithm, whose analysis involves combinatorial arguments that make it difficult to apply and generalize to problems beyond the path query from Eq. (2). For example, it is unclear how to use color coding to recover the Plehn and Voigt result [30] for the induced path query from Eq. (3). In this paper, we generalize the technique to arbitrary conjunctions of NAE predicates and from graph coloring to hypergraph coloring.

Queries with disequalities. Our work also generalizes prior work on answering queries with disequalities, which are a special case of queries with negated relations of bounded degree.

Papadimitriou and Yannakakis [27] showed that any acyclic join query Q with an arbitrary set of disequalities on k variables can be evaluated in time $2^{O(k \log k)} \cdot |D| \cdot |Q(D)| \cdot \log^2 |D|$ over any database D. This builds on, yet uses more colors than the color-coding technique.

Bagan et al [9] extended this result to free-connex acyclic queries; they also shaved off a $\log |D|$ factor by using a RAM model of computation differently from ours, where sorting can be done in linear time.

Koutris et al [21] introduced a practical algorithm for conjunctive queries with disequalities: Given a select-project-join (SPJ) plan for the conjunctive query without disequalities, the disequalities can be solved uniformly using an extended projection operator. The reliance on SPJ plans is a limitation, since it is known that such plans are suboptimal for join processing [25] and are inadequate to achieve the fhtw and subw complexity bounds. Our approach uses the InsideOut [2] and PANDA [3] query evaluation algorithms and inherits their low data complexity, thus achieving both bounds as stated in Theorem 1.1.

Differently from prior work and in line with our work, Koutris et al [21] also investigated query structures for which the combined complexity becomes polynomial: This is the case for queries whose augmented hypergraphs have bounded treewidth (an augmented hypergraph is the hypergraph of the skeleton conjunctive query extended with one hyperedge per disequality). Koutris et al [21] further proposed an alternative query answering approach that uses the probabilistic construction of the original color-coding technique coupled with any query evaluation algorithm. When restricted to queries with disequalities, our query complexity analysis is more refined than [21] as the number of colors used in our generalization of color-coding is sensitive to the query structure.

Tensor decomposition. Our Boolean tensor decomposition for conjunctions of NAE predicates draws on the general framework of tensor decomposition used in signal processing and machine learning [19, 33]. It is a special case of sum-product decomposition and a powerful

tool. Typical dynamic programming algorithms solve subproblems by *combining* relations and *eliminating* variables [34, 26, 2]. The sum-product decomposition is the dual approach that *decomposes* a formula and *introduces* new variables. The PANDA algorithm [3] achieves a generalization of the submodular width by rewriting a conjunction as a sum-product over tree decompositions. By combining PANDA with our Boolean tensor decomposition, we can answer queries with negation in time defined by the submodular width.

While close in spirit to k-restrictions [6], our approach to derandomization of the construction of the Boolean tensor decomposition is different since we would like to execute it in time defined by the fhtw-bound for computing body. Our derandomization uses a code-concatenation technique where the outer-code is a linear error-correcting code on the Gilbert-Varshamov boundary [31] that can be constructed in linear time. As a byproduct, the code enables an efficient construction of an (N, k^2, k)-perfect hash family of size $O(k^2 \log N)$. To the best of our knowledge, the prior constructions yield families of size $O(k^4 \log N)$ [6].

Data sparsity. We connect two notions of sparsity in this work. One is the bounded degree of the input relations that are negated in the query. There are notions of sparsity beyond bounded degree, cf. [28] for an excellent and comprehensive course on sparsity. The most refined sparsity notion is that of *nowhere denseness* [16], which characterizes the *input monotone* graph classes on which FO model checking is fixed-parameter tractable. We leave as future work the generalization of our work to queries with negated nowhere-dense relations.

The second notion of sparsity used in this work is given by the Boolean tensor rank of the Boolean tensor decomposition of the conjunction of NAE predicates. We note that the relation represented by such a conjunction is not necessarily nowhere dense.

8 Concluding remarks

In this paper, we studied the complexity of answering conjunctive queries with negation on relations of bounded degree. We give an approach that matches the data complexity of the best known query evaluation algorithms InsideOut [2] and PANDA [3].

An intriguing venue of future research is to further lower the query complexity of our approach. Proposition 5.8 presented lower bounds on θ^* that are dependent on the structure of the multi-hypergraph $\mathcal{G}$ of the input query. It is an intriguing open problem to give a lower bound on θ^* that is dependent on some known parameter of $\mathcal{G}$. The extended version of this paper [1] discusses two further ideas on how to reduce the query complexity:

- Cast coloring as a join of "coloring predicates" and apply the InsideOut algorithm on the resulting query with the coloring predicates taken into account; and
- Exploit symmetry to answer the k-path query in time $2^{O(k)} N \log N$ [8] instead of $O(k^k N \log N)$.

Our approach extends immediately to unions of conjunctive queries with negated relations and of degree bounds on the positive relations. In the latter case, we can achieve a runtime depending on the *degree-aware* version of the submodular width [3].

We finally note that our Boolean tensor decomposition technique cannot be generalized to more powerful semirings such as the sum-product semiring over the reals due to an intrinsic computational difficulty: The counting version of the (induced) k-path query from Section 1 is #W[1]-hard [11, 14].

References

1 Mahmoud Abo Khamis, Hung Q. Ngo, Dan Olteanu, and Dan Suciu. Boolean Tensor Decomposition for Conjunctive Queries with Negation. *CoRR*, abs/1712.07445, 2017. `arXiv:1712.07445`.

2 Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. FAQ: Questions Asked Frequently. In *PODS*, pages 13–28, 2016.

3 Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. What Do Shannon-type Inequalities, Submodular Width, and Disjunctive Datalog Have to Do with One Another? In *PODS*, pages 429–444, 2017.

4 Noga Alon, Jehoshua Bruck, Joseph Naor, Moni Naor, and Ron M. Roth. Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs. *IEEE Trans. Information Theory*, 38(2):509–516, 1992.

5 Noga Alon and Jeong Han Kim. On the degree, size, and chromatic index of a uniform hypergraph. *J. Combin. Theory Ser. A*, 77(1):165–170, 1997.

6 Noga Alon, Dana Moshkovitz, and Shmuel Safra. Algorithmic construction of sets for k-restrictions. *ACM Trans. Algorithms*, 2(2):153–177, 2006.

7 Noga Alon and Joel Spencer. *The Probabilistic Method*. John Wiley, 1992.

8 Noga Alon, Raphael Yuster, and Uri Zwick. Color-Coding. *J. ACM*, 42(4):844–856, 1995.

9 Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On Acyclic Conjunctive Queries and Constant Delay Enumeration. In *CSL*, pages 208–222, 2007.

10 Yijia Chen and Jörg Flum. On Parameterized Path and Chordless Path Problems. In *CCC*, pages 250–263, 2007.

11 Yijia Chen, Marc Thurley, and Mark Weyer. Understanding the Complexity of Induced Subgraph Isomorphisms. In *ICALP*, pages 587–596, 2008.

12 Ding-Zhu Du and Frank K. Hwang. *Combinatorial group testing and its applications*, volume 12 of *Series on Applied Mathematics*. World Scientific Publishing Co. Inc., second edition, 2000.

13 V. Faber. Linear Hypergraph Edge Coloring. *ArXiv e-prints*, 2016. `arXiv:1603.04938`.

14 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

15 Georg Gottlob, Gianluigi Greco, Nicola Leone, and Francesco Scarcello. Hypertree Decompositions: Questions and Answers. In *PODS*, pages 57–74, 2016.

16 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding First-Order Properties of Nowhere Dense Graphs. *J. ACM*, 64(3):17:1–17:32, 2017.

17 Martin Grohe and Dániel Marx. Constraint Solving via Fractional Edge Covers. *ACM Trans. Alg.*, 11(1):4, 2014.

18 Tommy R. Jensen and Bjarne Toft. *Graph coloring problems*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons Inc., New York, 1995. A Wiley-Interscience Publication.

19 Tamara G. Kolda and Brett W. Bader. Tensor Decompositions and Applications. *SIAM Rev.*, 51(3):455–500, 2009.

20 D. König. Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre. *Math. Ann.*, 77:453–465, 1916.

21 Paraschos Koutris, Tova Milo, Sudeepa Roy, and Dan Suciu. Answering Conjunctive Queries with Inequalities. *Theory Comput. Syst.*, 61(1):2–30, 2017.

22 Valentas Kurauskas and Katarzyna Rybarczyk. On the chromatic index of random uniform hypergraphs. *SIAM J. Discrete Math.*, 29(1):541–558, 2015.

23 Dániel Marx. Tractable Hypergraph Properties for Constraint Satisfaction and Conjunctive Queries. *J. ACM*, 60(6):42:1–42:51, November 2013. `doi:10.1145/2535926`.

24 Hung Q. Ngo and Ding-Zhu Du. A Survey on Combinatorial Group Testing Algorithms with Applications to DNA Library Screening. In *DIMACS*, volume 55, pages 171–182. Amer. Math. Soc., 2000.

25 Hung Q. Ngo, Christopher Ré, and Atri Rudra. Skew Strikes Back: New Developments in the Theory of Join Algorithms. In *SIGMOD Rec.*, pages 5–16, 2013.

26 Dan Olteanu and Jakub Závodnỳ. Size Bounds for Factorised Representations of Query Results. *TODS*, 40(1):2:1–2:44, 2015.

27 Christos H. Papadimitriou and Mihalis Yannakakis. On the Complexity of Database Queries. *J. Comput. Syst. Sci.*, 58(3):407–427, 1999.

28 Michał Pilipczuk and Sebastian Siebertz. Sparsity. Technical report, University of Warsaw, December 2017. URL: `https://www.mimuw.edu.pl/~mp248287/sparsity/`.

29 Nicholas Pippenger and Joel Spencer. Asymptotic behavior of the chromatic index for hypergraphs. *J. Combin. Theory Ser. A*, 51(1):24–42, 1989.

30 Jürgen Plehn and Bernd Voigt. Finding Minimally Weighted Subgraphs. In *Graph-Theoretic Concepts in Computer Science*, pages 18–29, 1990.

31 Ely Porat and Amir Rothschild. Explicit Nonadaptive Combinatorial Group Testing Schemes. *IEEE Trans. Information Theory*, 57(12):7982–7989, 2011.

32 Luc Segoufin. Enumerating with Constant Delay the Answers to a Query. In *ICDT*, pages 10–20, 2013.

33 Nicholas D. Sidiropoulos, Lieven De Lathauwer, Xiao Fu, Kejun Huang, Evangelos E. Papalexakis, and Christos Faloutsos. Tensor Decomposition for Signal Processing and Machine Learning. *Trans. Sig. Proc.*, 65(13):3551–3582, 2017.

34 Mihalis Yannakakis. Algorithms for Acyclic Database Schemes. In *VLDB*, pages 82–94, 1981.

Constant-Delay Enumeration for Nondeterministic Document Spanners

Antoine Amarilli
LTCI Paris, France
Télécom ParisTech, France
Université Paris-Saclay, France

Pierre Bourhis
CNRS Lille, CRIStAL UMR 9189, France
Inria Lille, France

Stefan Mengel
CNRS, CRIL UMR 8188, Lens, France

Matthias Niewerth
University of Bayreuth, Germany

Abstract

We consider the information extraction framework known as *document spanners*, and study the problem of efficiently computing the results of the extraction from an input document, where the extraction task is described as a sequential *variable-set automaton* (VA). We pose this problem in the setting of enumeration algorithms, where we can first run a preprocessing phase and must then produce the results with a small delay between any two consecutive results. Our goal is to have an algorithm which is tractable in combined complexity, i.e., in the sizes of the input document and the VA; while ensuring the best possible data complexity bounds in the input document size, i.e., constant delay in the document size. Several recent works at PODS'18 proposed such algorithms but with linear delay in the document size or with an exponential dependency in size of the (generally nondeterministic) input VA. In particular, Florenzano et al. suggest that our desired runtime guarantees cannot be met for general sequential VAs. We refute this and show that, given a nondeterministic sequential VA and an input document, we can enumerate the mappings of the VA on the document with the following bounds: the preprocessing is linear in the document size and polynomial in the size of the VA, and the delay is independent of the document and polynomial in the size of the VA. The resulting algorithm thus achieves tractability in combined complexity and the best possible data complexity bounds. Moreover, it is rather easy to describe, in particular for the restricted case of so-called extended VAs.

2012 ACM Subject Classification Information systems → Information extraction; Theory of computation → Formal languages and automata theory; Theory of computation → Database query processing and optimization (theory)

Keywords and phrases enumeration, spanners, automata

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.22

Related Version A full version of the paper is available at `https://arxiv.org/abs/1807.09320`.

Funding *Pierre Bourhis*: Supported by the DeLTA ANR project (ANR-16-CE40-0007).
Matthias Niewerth: Supported by grant number MA 4938/4–1 from the Deutsche Forschungsgemeinschaft.

1 Introduction

Information extraction from text documents is an important problem in data management. One approach to this task has recently attracted a lot of attention: it uses *document spanners*, a declarative logic-based approach first implemented by IBM in their tool SystemT [26] and whose core semantics have then been formalized in [10]. The spanner approach uses variants of regular expressions (e.g. *regex formulas* with variables), compiles them to variants of finite automata (e.g., *variable-set automata*, for short *VAs*), and evaluates them on the input document to extract the data of interest. After this extraction phase, algebraic operations like joins, unions and projections can be performed. The formalization of the spanner framework in [10] has led to a thorough investigation of its properties by the theoretical database community [13, 15, 21, 14, 11].

We here consider the basic task in the spanner framework of efficiently computing the results of the extraction, i.e., computing without duplicates all tuples of ranges of the input document (called *mappings*) that satisfy the conditions described by a VA. As many algebraic operations can also be compiled into VAs [15], this task actually solves the whole data extraction problem for so-called *regular spanners* [10]. While the extraction task is intractable for general VAs [13], it is known to be tractable if we impose that the VA is *sequential* [15, 11], which requires that all accepting runs actually describe a well-formed mapping; we will make this assumption throughout our work. Even then, however, it may still be unreasonable in practice to materialize all mappings: if there are k variables to extract, then mappings are k-tuples and there may be up to n^k mappings on an input document of size n, which is unrealistic if n is large. For this reason, recent works [21, 11, 15] have studied the extraction task in the setting of *enumeration algorithms*: instead of materializing all mappings, we enumerate them one by one while ensuring that the *delay* between two results is always small. Specifically, [15, Theorem 3.3] has shown how to enumerate the mappings with delay linear in the input document and quadratic in the VA, i.e., given a document d and a functional VA A (a subclass of sequential VAs), the delay is $O(|A|^2 \times |d|)$.

Although this result ensures tractability in both the size of the input document and the automaton, the delay may still be long as $|d|$ is generally very large. By contrast, enumeration algorithms for database tasks often enforce stronger tractability guarantees in data complexity [27, 30], in particular *linear preprocessing* and *constant delay* (when measuring complexity in the RAM model with uniform cost measure [1]). Such algorithms consist of two phases: a *preprocessing phase* which precomputes an index data structure in linear data complexity, and an *enumeration phase* which produces all results so that the delay between any two consecutive results is always *constant*, i.e., independent from the input data. It was recently shown in [11] that this strong guarantee could be achieved when enumerating the mappings of VAs if we only focus on data complexity, i.e., for any *fixed* VA, we can enumerate its mappings with linear preprocessing and constant delay in the input document. However, the preprocessing and delay in [11] are exponential in the VA because they first determinize it [11, Propositions 4.1 and 4.3]. This is problematic because the VAs constructed from regex formulas [10] are generally nondeterministic.

Thus, to efficiently enumerate the results of the extraction, we would ideally want to have the best of both worlds: ensure that the *combined complexity* (in the sequential VA and in the document) remains polynomial, while ensuring that the *data complexity* (in the document) is as small as possible, i.e., linear time for the preprocessing phase and constant time for the delay of the enumeration phase. However, up to now, there was no known algorithm to satisfy these requirements while working on nondeterministic sequential VAs. Further, it was conjectured that such an algorithm is unlikely to exist [11] because the related task of *counting* the number of mappings is SpanL-hard for such VAs.

The question of nondeterminism is also unsolved for the related problem of enumerating the results of monadic second-order (MSO) queries on words and trees: there are several approaches for this task where the query is given as an automaton, but they require the automaton to be deterministic [6, 2] or their delay is not constant in the input document [19]. Hence, also in the context of MSO enumeration, it is not known whether we can achieve linear preprocessing and constant delay in data complexity while remaining tractable in the (generally non-deterministic) automaton. The result that we will show in the present paper will imply that we can achieve this for MSO queries on words when all free variables are first-order, with the query being represented as a generally non-deterministic sequential VA, or as a sequential regex-formula with capture variables: note that an extension to trees is investigated in our follow-up work [4].

Contributions. In this work, we show that nondeterminism is in fact not an obstacle to enumerating the results of document spanners: we present an algorithm that enumerates the mappings of a nondeterministic sequential VA in polynomial combined complexity while ensuring linear preprocessing and constant delay in the input document. This answers the open question of [11], and improves on the bounds of [15]. More precisely, we show:

▶ **Theorem 1.1.** *Let $2 \leq \omega \leq 3$ be an exponent for Boolean matrix multiplication. Let $\mathcal{A}$ be a sequential VA with variable set $\mathcal{V}$ and with state set Q, and let d be an input document. We can enumerate the mappings of $\mathcal{A}$ on d with preprocessing time in $O((|Q|^{\omega+1} + |\mathcal{A}|) \times |d|)$ and with delay $O(|\mathcal{V}| \times (|Q|^2 + |\mathcal{A}| \times |\mathcal{V}|^2))$, i.e., linear preprocessing and constant delay in the input document, and polynomial preprocessing and delay in the input VA.*

The existence of such an algorithm is surprising but in hindsight not entirely unexpected: remember that, in formal language theory, when we are given a word and a nondeterministic finite automaton, then we can evaluate the automaton on the word with tractable combined complexity by determinizing the automaton "on the fly", i.e., computing at each position of the word the set of states where the automaton can be. Our algorithm generalizes this intuition, and extends it to the task of enumerating mappings without duplicates: we first present it for so-called *extended sequential VAs*[1], a variant of sequential VAs introduced in [11], before generalizing it to sequential VAs. Our overall approach is to construct a kind of product of the input document with the extended VA, similarly to [11]. We then use several tricks to ensure the constant delay bound despite nondeterminism; in particular we precompute a *jump function* that allows us to skip quickly the parts of the document where no variable can be assigned. The resulting algorithm is rather simple and has no large hidden constants. Note that our enumeration algorithm does not contradict the counting hardness results of [11, Theorem 5.2]: while our algorithm *enumerates* mappings with constant delay and without duplicates, we do not see a way to adapt it to *count* the mappings efficiently. This is similar to the enumeration and counting problems for maximal cliques: we can enumerate maximal cliques with polynomial delay [28], but counting them is #P-hard [29].

To extend our result to sequential VAs that are not extended, one possibility would be to convert them to extended VAs, but this necessarily entails an exponential blowup [11, Proposition 4.2]. We avoid this by adapting our algorithm to work with non-extended sequential VAs directly. Our idea for this is to efficiently enumerate at each position the possible sets of markers that can be assigned by the VA: we do so by enumerating paths

[1] Note that, contrary to what the terminology suggests, VAs are not special cases of extended VAs. Further, while extended VAs can be converted in PTIME to VAs, the converse is not true as there are extended VAs for which the smallest equivalent VA has exponential size [11].

in the VA, relying on the fact that the VA is sequential so these paths are acyclic. The challenge is that the same set of markers can be captured by many different paths, but we explain how we can explore efficiently the set of distinct paths with a technique known as *flashlight search* [20, 25]: the key idea is that we can efficiently determine which partial sets of markers can be extended to the label of a path (Lemma 6.4).

Of course, our main theorem (Theorem 1.1) implies analogous results for all spanner formalisms that can be translated to sequential VAs. In particular, spanners are not usually written as automata by users, but instead given in a form of regular expressions called *regex-formulas*, see [10] for exact definitions. As we can translate sequential regex-formulas to sequential VAs in linear time [10, 15, 21], our results imply that we can also evaluate them:

▶ **Corollary 1.2.** *Let $2 \leq \omega \leq 3$ be an exponent for Boolean matrix multiplication. Let φ be a sequential regex-formula with variable set $\mathcal{V}$, and let d be an input document. We can enumerate the mappings of φ on d with preprocessing time in $O(|\varphi|^{\omega+1} \times |d|)$ and with delay $O(|\mathcal{V}| \times (|\varphi|^2 + |\varphi| \times |\mathcal{V}|^2))$, i.e., linear preprocessing and constant delay in the input document, and polynomial preprocessing and delay in the input regex-formula.*

Another direct application of our result is for so-called *regular spanners* which are unions of conjunctive queries (UCQs) posed on regex-formulas, i.e., the closure of regex-formulas under union, projection and joins. We again point the reader to [10, 15] for the full definitions. As such UCQs can in fact be evaluated by VAs, our result also implies tractability for such representations, as long as we only perform a bounded number of joins:

▶ **Corollary 1.3.** *For every fixed $k \in \mathbb{N}$, let k-UCQ denote the class of document spanners represented by UCQs over functional regex-formulas with at most k applications of the join operator. Then the mappings of a spanner in k-UCQ can be enumerated with linear preprocessing and constant delay in the document size, and with polynomial preprocessing and delay in the size of the spanner representation.*

Paper structure. In Section 2, we formally define spanners, VAs, and the enumeration problem that we want to solve on them. In Sections 3–5, we prove our main result (Theorem 1.1) for *extended* VAs, where the sets of variables that can be assigned at each position are specified explicitly. We first describe in Section 3 the main part of our preprocessing phase, which converts the extended VA and input document to a *mapping DAG* whose paths describe the mappings that we wish to enumerate. We then describe in Section 4 how to enumerate these paths, up to having precomputed a so-called *jump function* whose computation is explained in Section 5. Last, we adapt our scheme in Section 6 for sequential VAs that are not extended. We conclude in Section 7.

2 Preliminaries

Document spanners. We fix a finite alphabet Σ. A *document* $d = d_0 \cdots d_{n-1}$ is just a word over Σ. A *span* of d is a pair $[i, j\rangle$ with $0 \leq i \leq j \leq |d|$ which represents a substring (contiguous subsequence) of d starting at position i and ending at position $j - 1$. To describe the possible results of an information extraction task, we will use a finite set $\mathcal{V}$ of variables, and define a result as a *mapping* from these variables to spans of the input document. Following [11, 21] but in contrast to [10], we will not require mappings to assign all variables: formally, a *mapping* of $\mathcal{V}$ on d is a function μ from some domain $\mathcal{V}' \subseteq \mathcal{V}$ to spans of d. We define a *document spanner* to be a function assigning to every input document d a set of mappings, which denotes the set of results of the extraction task on the document d.

Variable-set automata. We will represent document spanners using *variable-set automata* (or *VAs*). The transitions of a VA can carry letters of Σ or *variable markers*, which are either of the form $x\vdash$ for a variable $x \in \mathcal{V}$ (denoting the start of the span assigned to x) or $\dashv x$ (denoting its end). Formally, a *variable-set automaton* $\mathcal{A}$ (or VA) is then defined to be an automaton $\mathcal{A} = (Q, q_0, F, \delta)$ where the transition relation δ consists of *letter transitions* of the form (q, a, q') for $q, q' \in Q$ and $a \in \Sigma$, and of *variable transitions* of the form $(q, x\vdash, q')$ or $(q, \dashv x, q')$ for $q, q' \in Q$ and $x \in \mathcal{V}$. A *configuration* of a VA is a pair (q, i) where $q \in Q$ and i is a position of the input document d. A *run* σ of $\mathcal{A}$ on d is then a sequence of configurations

$$(q_0, i_0) \xrightarrow{\sigma_1} (q_1, i_1) \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_m} (q_m, i_m)$$

where $i_0 = 0$, $i_m = |d|$, and where for every $1 \leq j \leq m$:

- Either σ_j is a letter of Σ, we have $i_j = i_{j-1} + 1$, we have $d_{i_{j-1}} = \sigma_j$, and (q_{j-1}, σ_j, q_j) is a letter transition of $\mathcal{A}$;
- Or σ_j is a variable marker, we have $i_j = i_{j-1}$, and (q_{j-1}, σ_j, q_j) is a variable transition of $\mathcal{A}$. In this case we say that the variable marker σ_j is *read* at position i_j.

As usual, we say that a run is *accepting* if $q_m \in F$. A run is *valid* if it is accepting, every variable marker is read at most once, and whenever an open marker $x\vdash$ is read at a position i then the corresponding close marker $\dashv x$ is read at a position i' with $i \leq i'$. From each valid run, we define a mapping where each variable $x \in \mathcal{V}$ is mapped to the span $[i, i'\rangle$ such that $x\vdash$ is read at position i and $\dashv x$ is read at position i'; if these markers are not read then x is not assigned by the mapping (i.e., it is not in the domain $\mathcal{V}'$). The *document spanner* of the VA $\mathcal{A}$ is then the function that assigns to every document d the set of mappings defined by the valid runs of $\mathcal{A}$ on d: note that the same mapping can be defined by multiple different runs. The task studied in this paper is the following: given a VA $\mathcal{A}$ and a document d, enumerate *without duplicates* the mappings that are assigned to d by the document spanner of $\mathcal{A}$. The enumeration must write each mapping as a set of pairs (m, i) where m is a variable marker and i is a position of d.

Sequential VAs. We cannot hope to efficiently enumerate the mappings of arbitrary VAs because it is already NP-complete to decide if, given a VA $\mathcal{A}$ and a document d, there are any valid runs of $\mathcal{A}$ on d [13]. For this reason, we will restrict ourselves to so-called *sequential* VAs [21]. A VA $\mathcal{A}$ is *sequential* if for every document d, every accepting run of $\mathcal{A}$ of d is also valid: this implies that the document spanner of $\mathcal{A}$ can simply be defined following the accepting runs of $\mathcal{A}$. If we are given a VA, then we can test in NL whether it is sequential [21, Proposition 5.5], and otherwise we can convert it to an equivalent sequential VA (i.e., that defines the same document spanner) with an unavoidable exponential blowup in the number of variables (not in the number of states), using existing results:

▶ **Proposition 2.1.** *Given a VA $\mathcal{A}$ on variable set $\mathcal{V}$, letting $k := |\mathcal{V}|$ and r be the number of states of $\mathcal{A}$, we can compute an equivalent sequential VA $\mathcal{A}'$ with $3^k r$ states. Conversely, for any $k \in \mathbb{N}$, there exists a VA $\mathcal{A}_k$ with 1 state on a variable set with k variables such that any sequential VA equivalent to $\mathcal{A}_k$ has at least 3^k states.*

Proof. This can be shown exactly like [13, Proposition 12] and [12, Proposition 3.9]. In short, the upper bound is shown by modifying $\mathcal{A}$ to remember in the automaton state which variables have been opened or closed, and by re-wiring the transitions to ensure that the run is valid: this creates 3^k copies of every state because each variable can be either unseen, opened, or closed. For the lower bound, [12, Proposition 3.9] gives a VA for which any equivalent sequential VA must remember the status of all variables in this way. ◀

All VAs studied in this work will be sequential, and we will further assume that they are *trimmed* in the sense that for every state q there is a document d and an accepting run of the VA where the state q appears. This condition can be enforced in linear time on any sequential VA: we do a graph traversal to identify the accessible states (the ones that are reachable from the initial state), we do another graph traversal to identify the co-accessible states (the ones from which we can reach a final state), and we remove all states that are not accessible or not co-accessible. We will implicitly assume that all sequential VAs have been trimmed, which implies that they cannot contain any cycle of variable transitions (as such a cycle would otherwise appear in a run, which would not be valid).

Extended VAs. We will first prove our results for a variant of sequential VAs introduced by [11], called sequential *extended VAs*. An extended VA on alphabet Σ and variable set $\mathcal{V}$ is an automaton $\mathcal{A} = (Q, q_0, F, \delta)$ where the transition relation δ consists of *letter transitions* as before, and of *extended variable transitions* (or *ev-transitions*) of the form (q, M, q') where M is a possibly empty set of variable markers. Intuitively, on ev-transitions, the automaton reads multiple markers at once. Formally, a *run* σ of $\mathcal{A}$ on $d = d_0 \cdots d_{n-1}$ is a sequence of configurations (defined like before) where letter transitions and ev-transitions alternate:

$$(q_0, 0) \xrightarrow{M_0} (q'_0, 0) \xrightarrow{d_0} (q_1, 1) \xrightarrow{M_1} (q'_1, 1) \xrightarrow{d_1} \cdots \xrightarrow{d_{n-1}} (q_n, n) \xrightarrow{M_n} (q'_n, n)$$

where (q'_i, d_i, q_{i+1}) is a letter transition of $\mathcal{A}$ for all $0 \leq i < n$, and (q_i, M_i, q'_i) is an ev-transition of $\mathcal{A}$ for all $0 \leq i \leq n$ where M_i is the set of variable markers *read* at position i. Accepting and valid runs are defined like before, and the extended VA is sequential if all accepting runs are valid, in which case its document spanner is defined like before.

Our definition of extended VAs is slightly different from [11] because we allow ev-transitions that read the empty set to change the automaton state. This allows us to make a small additional assumption to simplify our proofs: we require that the states of extended VAs are partitioned between *ev-states*, from which only ev-transitions originate (i.e., the q_i above), and *letter-states*, from which only letter transitions originate (i.e., the q'_i above); and we impose that the initial state is an ev-state and the final states are all letter-states. Note that transitions reading the empty set move from an ev-state to a letter-state, like all other ev-transitions. Our requirement can be imposed in linear time on any input extended VA by rewriting each state to one letter-state and one ev-state, and re-wiring the transitions and changing the initial/final status of states appropriately. This rewriting preserves sequentiality and guarantees that any path in the rewritten extended VA must alternate between letter transitions and ev-transitions. Hence, we implicitly make this assumption on all extended VAs from now on.

▶ **Example 2.2.** The top of Figure 1 represents a sequential extended VA $\mathcal{A}_0$ to extract email addresses. To keep the example readable, we simply define them as words (delimited by a space or by the beginning or end of document) which contain one at-sign "@" preceded and followed by a non-empty sequence of non-"@" characters. In the drawing of $\mathcal{A}_0$, the initial state q_0 is at the left, and the states q_{10} and q_{12} are final. The transitions labeled by Σ represent a set of transitions for each letter of Σ, and the same holds for Σ' which we define as $\Sigma' := \Sigma \setminus \{@, \sqcup\}$.

It is easy to see that, on any input document d, there is one mapping of $\mathcal{A}_0$ on d per email address contained in d, which assigns the markers $x\vdash$ and $\dashv x$ to the beginning and end of the email address, respectively. In particular, $\mathcal{A}_0$ is sequential, because any accepting run is valid. Note that $\mathcal{A}_0$ happens to have the property that each mapping is produced by exactly one accepting run, but our results in this paper do not rely on this property.

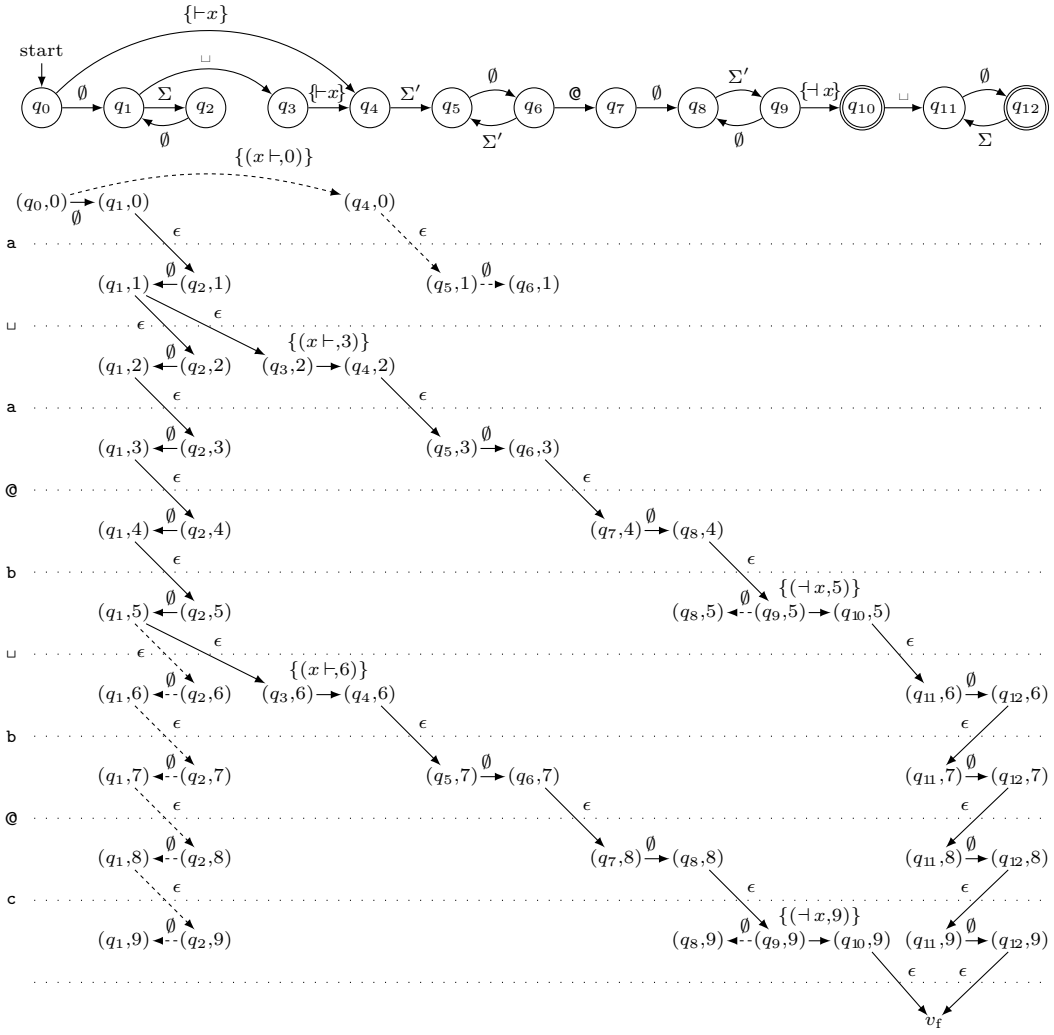


Figure 1 Example sequential extended VA $\mathcal{A}_0$ to extract e-mail addresses (see Example 2.2) and example mapping DAG on an example document (see Examples 3.3, 3.6, 3.7, and 3.10).

Matrix multiplication. The complexity bottleneck for some of our results will be the complexity of multiplying two Boolean matrices, which is a long-standing open problem, see e.g. [16] for a recent discussion. When stating our results, we will often denote by $2 \leq \omega \leq 3$ an exponent for Boolean matrix multiplication: this is a constant such that the product of two r-by-r Boolean matrices can be computed in time $O(r^\omega)$. For instance, we can take $\omega := 3$ if we use the naive algorithm for Boolean matrix multiplication, and it is obvious that we must have $\omega \geq 2$. The best known upper bound is currently $\omega < 2.3728639$, see [17].

3 Computing Mapping DAGs for Extended VAs

We start our paper by studying extended VAs, which are easier to work with because the set of markers that can be assigned at every position is explicitly written as the label of a single transition. We accordingly show Theorem 1.1 for the case of extended VAs in Sections 3–5. We will then cover the case of non-extended VAs in Section 6.

To show Theorem 1.1 for extended VAs, we will reduce the problem of enumerating the mappings captured by $\mathcal{A}$ to that of enumerating path labels in a special kind of directed acyclic graph (DAG), called a *mapping DAG*. This DAG is intuitively a variant of the product of $\mathcal{A}$ and of the document d, where we represent simultaneously the position in the document and the corresponding state of $\mathcal{A}$. We will no longer care in the mapping DAG about the labels of letter transitions, so we will erase these labels and call these transitions *ϵ-transitions*. As for the ev-transitions, we will extend their labels to indicate the position in the document in addition to the variable markers. We first give the general definition of a mapping DAG:

▶ **Definition 3.1.** *A* mapping DAG *consists of a set V of* vertices*, an* initial vertex $v_0 \in V$*, a* final vertex $v_\mathrm{f} \in V$*, and a set of* edges E *where each edge (s, x, t) has a* source vertex $s \in V$*, a* target vertex $t \in V$*, and a* label x *that may be ϵ (in which case we call the edge an* ϵ-edge*) or a finite (possibly empty) set of pairs (m, i), where m is a variable marker and i is a position. These edges are called* marker edges*. We require that the graph (V, E) is acyclic. We say that a mapping DAG is* normalized *if every path from the initial vertex to the final vertex starts with a marker edge, ends with an ϵ-edge, and alternates between marker edges and ϵ-edges.*

The mapping $\mu(\pi)$ *of a path π in the mapping DAG is the union of labels of the marker edges of π: we require of any mapping DAG that, for every path π, this union is disjoint. Given a set U of vertices of G, we write $\mathcal{M}(U)$ for the set of mappings of paths from a vertex of U to the final vertex; note that the same mapping may be captured by multiple different paths. The set of mappings* captured *by G is then $\mathcal{M}(G) := \mathcal{M}(\{v_0\})$.*

Intuitively, the ϵ-edges will correspond to letter transitions of $\mathcal{A}$ (with the letter being erased, i.e., replaced by ϵ), and marker edges will correspond to ev-transitions: their labels are a possibly empty finite set of pairs of a variable marker and position, describing which variables have been assigned during the transition. We now explain how we construct a mapping DAG from $\mathcal{A}$ and from a document d, which we call the *product DAG* of $\mathcal{A}$ and d:

▶ **Definition 3.2.** *Let $\mathcal{A} = (Q, q_0, F, \delta)$ be a sequential extended VA and let $d = d_0 \cdots d_{n-1}$ be an input document. The* product DAG *of $\mathcal{A}$ and d is the normalized mapping DAG whose vertex set is $Q \times \{0, \ldots, n\} \cup \{v_\mathrm{f}\}$ with $v_\mathrm{f} := (\bullet, n+1)$ for some fresh value $\bullet$. Its edges are:*

- *For every letter-transition (q, a, q') in δ, for every $0 \leq i < |d|$ such that $d_i = a$, there is an ϵ-edge from (q, i) to $(q', i+1)$;*
- *For every ev-transition (q, M, q') in δ, for every $0 \leq i \leq |d|$, there is a marker edge from (q, i) to (q', i) labeled with the (possibly empty) set $\{(m, i) \mid m \in M\}$.*
- *For every final state $q \in F$, an ϵ-edge from (q, n) to v_f.*

The initial vertex of the product DAG is $(q_0, 0)$ and the final vertex is v_f.

Note that, contrary to [11], we do not contract the ϵ-edges but keep them throughout our algorithm.

▶ **Example 3.3.** The mapping DAG for our example sequential extended VA $\mathcal{A}_0$ on the example document `a␣a@b␣b@c` is shown on Figure 1, with the document being written at the left from top to bottom. The initial vertex of the mapping DAG is $(q_0, 0)$ at the top left and its final vertex is v_f at the bottom. We draw marker edges horizontally, and ϵ-edges diagonally. To simplify the example, we only draw the parts of the mapping DAG that are reachable from the initial vertex. Edges are dashed when they cannot be used to reach the final vertex.

It is easy to see that this construction satisfies the definition:

▷ Claim 3.4. The product DAG of $\mathcal{A}$ and d is a normalized mapping DAG.

Proof sketch. The mapping DAG is acyclic and normalized because its edges follow the transitions of the extended VA, which we had preprocessed to distinguish letter-states and ev-states. Paths in the mapping DAG cannot contain multiple occurrences of the same label, because the labels in the mapping DAG include the position in the document. ◁

Further, the product DAG clearly captures what we want to enumerate. Formally:

▷ Claim 3.5. The set of mappings of $\mathcal{A}$ on d is exactly the set of mappings $\mathcal{M}(G)$ captured by the product DAG G.

▶ **Example 3.6.** The set of mappings captured by the example product DAG on Figure 1 is $\{\{(x\vdash, 3), (\dashv x, 5)\}, \{(x\vdash, 6), (\dashv x, 9)\}\}$, and this is indeed the set of mappings of the example extended VA $\mathcal{A}_0$ on the example document.

Our task is to enumerate $\mathcal{M}(G)$ *without duplicates*, and this is still non-obvious: because of nondeterminism, the same mapping in the product DAG may be witnessed by exponentially many paths, corresponding to exponentially many runs of the nondeterministic extended VA $\mathcal{A}$. We will present in the next section our algorithm to perform this task on the product DAG G. To do this, we will need to preprocess G by *trimming* it, and introduce the notion of *levels* to reason about its structure.

First, we present how to *trim* G. We say that G is *trimmed* if every vertex v is both *accessible* (there is a path from the initial vertex to v) and *co-accessible* (there is a path from v to the final vertex). Given a mapping DAG, we can clearly trim in linear time by two linear-time graph traversals. Hence, we will always implicitly assume that the mapping DAG is trimmed. If the mapping DAG may be empty once trimmed, then there are no mappings to enumerate, so our task is trivial. Hence, we assume in the sequel that the mapping DAG is non-empty after trimming. Further, if $\mathcal{V} = \emptyset$ then the only possible mapping is the empty mapping and we can produce it at that stage, so in the sequel we assume that $\mathcal{V}$ is non-empty.

▶ **Example 3.7.** For the mapping DAG of Figure 1, trimming eliminates the non-accessible vertices (which are not depicted) and the non-co-accessible vertices (i.e., those with incoming dashed edges).

Second, we present an invariant on the structure of G by introducing the notion of *levels*:

▶ **Definition 3.8.** *A mapping DAG G is* leveled *if its vertices $v = (q, i)$ are pairs whose second component i is a nonnegative integer called the* level *of the vertex and written* $\mathsf{level}(v)$*, and where the following conditions hold:*

- *For the initial vertex v_0 (which has no incoming edges), the level is 0;*
- *For every ϵ-edge from u to v, we have $\mathsf{level}(v) = \mathsf{level}(u) + 1$;*
- *For every marker edge from u to v, we have $\mathsf{level}(v) = \mathsf{level}(u)$. Furthermore, all pairs (m, i) in the label of the edge have $i = \mathsf{level}(v)$.*

The depth *D of G is the maximal level. The* width *W of G is the maximal number of vertices that have the same level.*

The following is then immediate by construction:

▷ Claim 3.9. The product DAG of $\mathcal{A}$ and d is leveled, and we have $W \leq |Q|$ and $D = |d| + 2$.

▶ **Example 3.10.** The example mapping DAG on Figure 1 is leveled, and the levels are represented as horizontal layers separated by dotted lines: the topmost level is level 0 and the bottommost level is level 10.

In addition to levels, we will need the notion of a *level set*:

▶ **Definition 3.11.** *A* level set Λ *is a non-empty set of vertices in a leveled normalized mapping DAG that all have the same level (written* $\mathsf{level}(\Lambda)$*) and which are all the source of some marker edge. The singleton* $\{v_\mathrm{f}\}$ *of the final vertex is also considered as a level set.*

In particular, letting v_0 be the initial vertex, the singleton $\{v_0\}$ is a level set. Further, if we consider a level set Λ which is not the final vertex, then we can follow marker edges from all vertices of Λ (and only such edges) to get to other vertices, and follow ϵ-edges from these vertices (and only such edges) to get to a new level set Λ' with $\mathsf{level}(\Lambda') = \mathsf{level}(\Lambda) + 1$.

4 Enumeration for Mapping DAGs

In the previous section, we have reduced our enumeration problem for extended VAs on documents to an enumeration problem on normalized leveled mapping DAGs. In this section, we describe our main enumeration algorithm on such DAGs and show the following:

▶ **Theorem 4.1.** *Let* $2 \leq \omega \leq 3$ *be an exponent for Boolean matrix multiplication. Given a normalized leveled mapping DAG* G *of depth* D *and width* W*, we can enumerate* $\mathcal{M}(G)$ *(without duplicates) with preprocessing* $O(|G| + D \times W^{\omega+1})$ *and delay* $O(W^2 \times (r+1))$ *where* r *is the size of each produced mapping.*

Remember that, as part of our preprocessing, we have ensured that the leveled normalized mapping DAG G has been trimmed. We will also preprocess G to ensure that, given any vertex, we can access its adjacency list (i.e., the list of its outgoing edges) in some sorted order on the labels, where we assume that $\emptyset$-edges come last. This sorting can be done in linear time on the RAM model [18, Theorem 3.1], so the preprocessing is in $O(|G|)$.

Our general enumeration algorithm is then presented as Algorithm 1. We explain the missing pieces next. The function Enum is initially called with $\Lambda = \{v_0\}$, the level set containing only the initial vertex, and with mapping being the empty set.

Algorithm 1 Main enumeration algorithm.

```
1: procedure ENUM(G, Λ, mapping)
2:     Λ' := JUMP(Λ)
3:     if Λ' is the singleton {v_f} of the final vertex then
4:         OUTPUT(mapping)
5:     else
6:         for (locmark, Λ'') in NEXTLEVEL(Λ') do
7:             ENUM(G, Λ'', locmark ∪ mapping)
```

For simplicity, let us assume for now that the Jump function just computes the identity, i.e., $\Lambda' := \Lambda$. As for the call NextLevel(Λ'), it returns the pairs $(\mathsf{locmark}, \Lambda'')$ where:

- The label set locmark is an edge label such that there is a marker edge labeled with locmark that starts at some vertex of Λ'
- The level set Λ'' is formed of all the vertices w at level $\mathsf{level}(\Lambda') + 1$ that can be reached from such an edge followed by an ϵ-edge. Formally, a vertex w is in Λ'' if and only if there is an edge labeled locmark from some vertex $v \in \Lambda$ to some vertex v', and there is an ϵ-edge from v' to w.

Remember that, as the mapping DAG is normalized, we know that all edges starting at vertices of the level set Λ' are marker edges (several of which may have the same label); and for any target v' of these edges, all edges that leave v' are ϵ-edges whose targets w are at the level $\mathsf{level}(\Lambda') + 1$.

It is easy to see that the NextLevel function can be computed efficiently:

▶ **Proposition 4.2.** *Given a leveled trimmed normalized mapping DAG G with width W, and a level set Λ', we can enumerate without duplicates all the pairs* $(\mathsf{locmark}, \Lambda'') \in \textsc{NextLevel}(\Lambda')$ *with delay $O(W^2 \times |\mathsf{locmark}|)$ in an order such that* $\mathsf{locmark} = \emptyset$ *comes last if it is returned.*

Proof. We simultaneously go over the sorted lists of the outgoing edges of each vertex of Λ', of which there are at most W, and we merge them. Specifically, as long as we are not done traversing all lists, we consider the smallest value of locmark (according to the order) that occurs at the current position of one of the lists. Then, we move forward in each list until the list is empty or the edge label at the current position is no longer equal to locmark, and we consider the set Λ'_2 of all vertices v' that are the targets of the edges that we have seen. This considers at most W^2 edges and reaches at most W vertices (which are at the same level as Λ'), and the total time spent reading edge labels is in $O(|\mathsf{locmark}|)$, so the process is in $O(W^2 \times |\mathsf{locmark}|)$ so far. Now, we consider the outgoing edges of all vertices $v' \in \Lambda'_2$ (all are ϵ-edges) and return the set Λ'' of the vertices w to which they lead: this only adds $O(W^2)$ to the running time because we consider at most W vertices v' with at most W outgoing edges each. Last, $\mathsf{locmark} = \emptyset$ comes last because of our assumption on the order of adjacency lists. ◀

The design of Algorithm 1 is justified by the fact that, for any level set Λ', the set $\mathcal{M}(\Lambda')$ can be partitioned based on the value of locmark. Formally:

▷ Claim 4.3. For any level set Λ of G which is not the final vertex, we have:

$$\mathcal{M}(\Lambda) \quad = \bigcup_{(\mathsf{locmark}, \Lambda'') \in \textsc{NextLevel}(\Lambda)} \mathsf{locmark} \cup \mathcal{M}(\Lambda'') \; . \tag{1}$$

Furthermore, this union is disjoint, non-empty, and none of its terms is empty.

Thanks to this claim, we could easily prove by induction that Algorithm 1 correctly enumerates $\mathcal{M}(G)$ when Jump is the identity function. However, this algorithm would not achieve the desired delay bounds: indeed, it may be the case that $\textsc{NextLevel}(\Lambda')$ only contains $\mathsf{locmark} = \emptyset$, and then the recursive call to Enum would not make progress in constructing the mapping, so the delay would not generally be linear in the size of the mapping. To avoid this issue, we use the Jump function to directly "jump" to a place in the mapping DAG where we can read a label different from $\emptyset$. Let us first give the relevant definitions:

▶ **Definition 4.4.** *Given a level set Λ in a leveled mapping DAG G, the* jump level $\mathsf{JL}(\Lambda)$ *of Λ is the first level $j \geq \mathsf{level}(\Lambda)$ containing a vertex v' such that some $v \in \Lambda$ has a path to v' and such that v' is either the final vertex or has an outgoing edge with a label which is $\neq \epsilon$ and $\neq \emptyset$. In particular we have $\mathsf{JL}(\Lambda) = \mathsf{level}(\Lambda)$ if some vertex in Λ already has an outgoing edge with such a label, or if Λ is the singleton set containing only the final vertex.*

The jump set *of Λ is then $\textsc{Jump}(\Lambda) := \Lambda$ if $\mathsf{JL}(\Lambda) = \mathsf{level}(\Lambda)$, and otherwise $\textsc{Jump}(\Lambda)$ is formed of all vertices at level $\mathsf{JL}(\Lambda)$ to which some $v \in \Lambda$ have a directed path whose last edge is labeled ϵ. This ensures that $\textsc{Jump}(\Lambda)$ is always a level set.*

The definition of Jump ensures that we can jump from Λ to $\textsc{Jump}(\Lambda)$ when enumerating mappings, and it will not change the result because we only jump over ϵ-edges and $\emptyset$-edges:

▷ Claim 4.5. For any level set Λ of G, we have $\mathcal{M}(\Lambda) = \mathcal{M}(\textsc{Jump}(\Lambda))$.

Claims 4.3 and 4.5 imply that Algorithm 1 is correct with this implementation of JUMP:

▶ **Proposition 4.6.** ENUM($\{v_0\}, \epsilon$) *correctly enumerates* $\mathcal{M}(G)$ *(without duplicates).*

What is more, Algorithm 1 now achieves the desired delay bounds, as we will show. Of course, this relies on the fact that the JUMP function can be efficiently precomputed and evaluated. We only state this fact for now, and prove it in the next section:

▶ **Proposition 4.7.** *Given a leveled mapping DAG G with width W, we can preprocess G in time $O(D \times W^{\omega+1})$ such that, given any level set Λ of G, we can compute the jump set* JUMP(Λ) *of Λ in time $O(W^2)$.*

We can now conclude the proof of Theorem 4.1 by showing that the preprocessing and delay bounds are as claimed. For the preprocessing, this is clear: we do the preprocessing in $O(|G|)$ presented at the beginning of the section (i.e., trimming, and computing the sorted adjacency lists), followed by that of Proposition 4.7. For the delay, we claim:

▷ Claim 4.8. Algorithm 1 has delay $O(W^2 \times (r+1))$, where r is the size of the mapping of each produced path. In particular, the delay is independent of the size of G.

Proof sketch. The time to call JUMP is in $O(W^2)$ by Proposition 4.7, and the time spent to move to the next iteration of the **for** loop with a label set locmark is in time $O(W^2 \times |\mathsf{locmark}|)$ using Proposition 4.2: now the operations in the loop body run in constant time if we represent mapping as a linked list so that we do not have to copy it when making the recursive call. As Proposition 4.2 ensures that $\emptyset$ comes last, when producing the first solution, we make at most $r+1$ calls to produce a solution of size r, and the time is in $O(W^2 \times (r+1))$. We adapt this argument to show that each successive solution is also produced within that bound: note that when we use $\emptyset$ in the **for** loop (which does not contribute to r) then the next call to ENUM either reaches the final vertex or uses a non-empty set which contributes to r. What is more, as $\emptyset$ is considered last, the corresponding call to ENUM is tail-recursive, so we can ensure that the size of the stack (and hence the time to unwind it) stays $\leq r+1$. ◁

Memory usage. We briefly discuss the *memory usage* of the enumeration phase, i.e., the maximal amount of working memory that we need to keep throughout the enumeration phase, not counting the precomputation phase. Indeed, in enumeration algorithms the memory usage can generally grow to be very large even if one adds only a constant amount of information at every step. We will show that this does not happen here, and that the memory usage throughout the enumeration remains polynomial in $\mathcal{A}$ and constant in the input document size.

All our memory usage during enumeration is in the call stack, and thanks to tail recursion elimination (see the proof of Claim 4.8) we know that the stack depth is at most $r+1$, where r is the size of the produced mapping as in the statement of Theorem 4.1. The local space in each stack frame must store Λ' and Λ'', which have size $O(W)$, and the status of the enumeration of NEXTLEVEL in Proposition 4.2, i.e., for every vertex $v \in \Lambda'$, the current position in its adjacency list: this also has total size $O(W)$, so the total memory usage of these structures over the whole stack is in $O((r+1) \times W)$. Last, we must also store the variables mapping and locmark, but their total size of the variables locmark across the stack is clearly r, and the same holds of mapping because each occurrence is stored as a linked list (with a pointer to the previous stack frame). Hence, the total memory usage is $O((r+1) \times W)$, i.e., $O((|\mathcal{V}|+1) \times |Q|)$ in terms of the extended VA.

5 Jump Function

The only missing piece in the enumeration scheme of Section 4 is the proof of Proposition 4.7. We first explain the preprocessing for the JUMP function, and then the computation scheme.

Preprocessing scheme. Recall the definition of the jump level $\mathsf{JL}(\Lambda)$ and jump set $\textsc{Jump}(\Lambda)$ of a level set Λ (Definition 4.4). We assume that we have precomputed in $O(|G|)$ the mapping level associating each vertex v to its level $\mathsf{level}(v)$, as well as, for each level i, the list of the vertices v such that $\mathsf{level}(v) = i$.

The first part of the preprocessing is then to compute, for every individual vertex v, the jump level $\mathsf{JL}(v) := \mathsf{JL}(\{v\})$, i.e., the minimal level containing a vertex v' such that v' is reachable from v and v' is either the final vertex or has an outgoing edge which is neither an ϵ-edge nor an $\emptyset$-edge. We claim:

▷ Claim 5.1. We can precompute in $O(D \times W^2)$ the jump level $\mathsf{JL}(v)$ of all vertices v of G.

Proof sketch. We do the computation along a reverse topological order: we have $\mathsf{JL}(v_\mathrm{f}) := \mathsf{level}(v_\mathrm{f})$ for the final vertex v_f, we have $\mathsf{JL}(v) := \mathsf{level}(v)$ if v has an outgoing edge which is not an ϵ-edge or an $\emptyset$-edge, and otherwise we have $\mathsf{JL}(v) := \min_{v \to w} \mathsf{JL}(w)$. ◁

The second part of the preprocessing is to compute, for each level i of G, the *reachable levels* $\mathsf{Rlevel}(i) := \{\mathsf{JL}(v) \mid \mathsf{level}(v) = i\}$, which we can clearly do in linear time in the number of vertices of G, i.e., in $O(D \times W)$. Note that the definition clearly ensures that we have $|\mathsf{Rlevel}(i)| \leq W$.

▶ **Example 5.2.** In Figure 1, the jumping level for nodes $(q_1, 3)$ and $(q_2, 3)$ is 6 and the jumping level for nodes $(q_5, 3)$ and $(q_6, 3)$ is 5. Hence, the set of reachable levels $\mathsf{Rlevel}(3)$ for level 3 is $\{5, 6\}$.

Last, the third step of the preprocessing is to compute a reachability matrix from each level to its reachable levels. Specifically, for any two levels $i < j$ of G, let $\mathsf{Reach}(i, j)$ be the Boolean matrix of size at most $W \times W$ which describes, for each (u, v) with $\mathsf{level}(u) = i$ and $\mathsf{level}(v) = j$, whether there is a path from u to v whose last edge is labeled ϵ. We can't afford to compute all these matrices, but we claim that we can efficiently compute a subset of them, which will be enough for our purposes:

▷ Claim 5.3. We can precompute in time $O(D \times W^{\omega+1})$ the matrices $\mathsf{Reach}(i, j)$ for all pairs of levels $i < j$ such that $j \in \mathsf{Rlevel}(i)$.

Proof sketch. We compute them in decreasing order on i: the matrix $\mathsf{Reach}(i, i+1)$ can be computed in time $O(W \times W)$ from the edge relation, and matrices $\mathsf{Reach}(i, j)$ with $j > i+1$ can be computed in time $O(W^\omega)$ as the product of $\mathsf{Reach}(i, i+1)$ and $\mathsf{Reach}(i+1, j)$: note that $\mathsf{Reach}(i+1, j)$ has been precomputed because $j \in \mathsf{Rlevel}(i)$ easily implies that $j \in \mathsf{Rlevel}(i+1)$. ◁

Evaluation scheme. We can now describe our evaluation scheme for the jump function. Given a level set Λ, we wish to compute $\textsc{Jump}(\Lambda)$. Let i be the level of Λ, and let j be $\mathsf{JL}(\Lambda)$ which we compute as $\min_{v \in \Lambda} \mathsf{JL}(v)$. If $j = i$, then $\textsc{Jump}(\Lambda) = \Lambda$ and there is nothing to do. Otherwise, by definition there must be $v \in \Lambda$ such that $\mathsf{JL}(v) = j$, so v witnesses that $j \in \mathsf{Rlevel}(i)$, and we know that we have precomputed the matrix $\mathsf{Reach}(i, j)$. Now $\textsc{Jump}(\Lambda)$ are the vertices at level j to which the vertices of Λ (at level i) have a directed path whose last edge is labeled ϵ, which we can simply compute in time $O(W^2)$ by unioning the lines that correspond to the vertices of Λ in the matrix $\mathsf{Reach}(i, j)$.

This concludes the proof of Proposition 4.7 and completes the presentation of our scheme to enumerate the set captured by mapping DAGs (Theorem 4.1). Together with Section 3, this proves Theorem 1.1 in the case of extended sequential VAs.

6 From Extended Sequential VAs to General Sequential VAs

In this section, we adapt our main result (Theorem 1.1) to work with sequential non-extended VAs rather than sequential extended VAs. Remember that we cannot tractably convert non-extended VAs into extended VAs [11, Proposition 4.2], so we must modify our construction in Sections 3–5 to work with sequential non-extended VAs directly. Our general approach will be the same: compute the mapping DAG and trim it like in Section 3, then precompute the jump level and jump set information as in Section 5, and apply the enumeration scheme of Section 4. The difficulty is that non-extended VAs may assign multiple markers at the same word position by taking multiple variable transitions instead of one single ev-transition. Hence, when enumerating all possible values for locmark in Algorithm 1, we need to consider all possible sequences of variable transitions. The challenge is that there may be many different transitions sequences that assign the same set of markers, which could lead to duplicates in the enumeration. Thus, our goal will be to design a replacement to Proposition 4.2 for non-extended VAs, i.e., enumerate possible values for locmark at each level without duplicates.

We start as in Section 3 by computing the product DAG G of $\mathcal{A}$ and of the input document $d = d_0 \cdots d_{n-1}$ with vertex set $Q \times \{0, \ldots, n\} \cup \{v_\mathrm{f}\}$ with $v_\mathrm{f} := (\bullet, n+1)$ for some fresh value $\bullet$, and with the following edge set:

- For every letter-transition (q, a, q') of $\mathcal{A}$, for every $0 \leq i < |d|$ such that $d_i = a$, there is an ϵ-edge from (q, i) to $(q', i+1)$;
- For every variable-transition (q, m, q') of $\mathcal{A}$ (where m is a marker), for every $0 \leq i \leq |d|$, there is an edge from (q, i) to (q', i) labeled with $\{(m, i)\}$.
- For every final state $q \in F$, an ϵ-edge from (q, n) to v_f.

The initial vertex of G is $(q_0, 0)$ and the final vertex is v_f. Note that the edge labels are now always singleton sets or ϵ; in particular there are no longer any $\emptyset$-edges.

We can then adapt most of Claim 3.4: the product DAG is acyclic because all letter-transitions make the second component increase, and because we know that there cannot be a cycle of variable-transitions in the input sequential VA $\mathcal{A}$ (remember that we assume VAs to be trimmed). We can also trim the mapping DAG in linear time as before, and Claim 3.5 also adapts to show that the resulting mapping DAG correctly captures the mappings that we wish to enumerate. Last, as in Claim 3.9, the resulting mapping DAG is still leveled, the depth D (number of levels) is still $|d| + 2$, and the width W (maximal size of a level) is still $\leq |Q|$; we will also define the *complete width* W_c of G in this section as the maximal size, over all levels i, of the sum of the number of vertices with level i and of the number of *edges* with a source vertex having level i: clearly we have $W_\mathrm{c} \leq |\mathcal{A}|$. The main change in Section 3 is that the mapping DAG is no longer normalized, i.e., we may follow several marker edges in succession (staying at the same level) or follow several ϵ-edges in succession (moving to the next level each time). Because of this, we change Definition 3.11 and redefine *level sets* to mean any non-empty set of vertices that are at the same level.

We then reuse the enumeration approach of Section 4 and 5. Even though the mapping DAG is no longer normalized, it is not hard to see that with our new definition of level sets we can reuse the jump function from Section 5 as-is, and we can also reuse the general approach of Algorithm 1. However, to accommodate for the different structure of the mapping DAG,

we will need a new definition for NextLevel: instead of following exactly one marker edge before an ϵ-edge, we want to be able to follow any (possibly empty) path of marker edges before an ϵ-edge. We formalize this notion as an *S^+-path*:

▶ **Definition 6.1.** *For S^+ a set of labels, an* S^+-path *in the mapping DAG G is a path of $|S^+|$ edges that includes no ϵ-edges and where the labels of the path are exactly the elements of S^+ in some arbitrary order. Recall that the definition of a mapping DAG ensures that there can be no duplicate labels on the path, and that the start and end vertices of an S^+-path must have the same level because no ϵ-edge is traversed in the path.*

For Λ a level set, NextLevel(Λ) *is the set of all pairs (S^+, Λ'') where:*

- *S^+ is a set of labels such that there is an S^+-path that goes from some vertex v of Λ to some vertex v' which has an outgoing ϵ-edge;*
- *Λ'' is the level set containing exactly the vertices w that are targets of these ϵ-edges, i.e., there is an S^+-path from some vertex $v \in \Lambda$ to some vertex v', and there is an ϵ-edge from v' to w.*

Note that these definitions are exactly equivalent to what we would obtain if we converted $\mathcal{A}$ to an extended VA and then used our original construction. This directly implies that the modified enumeration algorithm is correct (i.e., Proposition 4.6 extends). In particular, the modified algorithm still uses the jump pointers as computed in Section 5 to jump over positions where the only possibility is $S^+ = \emptyset$, i.e., positions where the sequential VA make no variable-transitions. The only thing that remains is to establish the delay bounds, for which we need to enumerate NextLevel efficiently without duplicates (and replace Proposition 4.2). To present our method for this, we will introduce the *alphabet size B* as the maximal number, over all levels j of the mapping DAG G, of the different labels that can occur in marker edges between vertices at level j; in our construction this value is bounded by the number of different markers, i.e., $B \leq 2\,|\mathcal{V}|$. We can now state the claim:

▶ **Theorem 6.2.** *Given a leveled trimmed mapping DAG G with complete width W_c and alphabet size B, and a level set Λ', we can enumerate without duplicates all the pairs $(S^+, \Lambda'') \in$* NextLevel(Λ') *with delay $O(W_c \times B^2)$ in an order such that $S^+ = \emptyset$ comes last if it is returned.*

With this runtime, the delay of Theorem 4.1 becomes $O((r+1) \times (W^2 + W_c \times B^2))$, and we know that $W_c \leq |\mathcal{A}|$, that $W \leq |Q|$, that $r \leq |\mathcal{V}|$, and that $B \leq 2\,|\mathcal{V}|$; so this leads to the overall delay of $O(|\mathcal{V}| \times (|Q|^2 + |\mathcal{A}| \times |\mathcal{V}|^2))$ in Theorem 1.1.

The idea to prove Theorem 6.2 is to use a general approach called *flashlight search* [20, 25]: we will use a search tree on the possible sets of labels on $\mathcal{V}$ to iteratively construct the set S^+ that can be assigned at the current position, and we will avoid useless parts of the search tree by using a lemma to efficiently check if a partial set of labels can be extended to a solution. To formalize the notion of extending a partial set, we will need the notion of *S^+/S^--paths*:

▶ **Definition 6.3.** *For S^- and S^+ two disjoint sets of labels, an* S^+/S^--path *in the mapping DAG G is a path of edges that includes no ϵ-edges, that includes no edges with a label in S^-, and where every label of S^+ is seen exactly once along the path.*

Note that, when $S^+ \cup S^-$ contains all labels used in G, then the notions of S^+/S^--path and S^+-path coincide, but if G contains some labels not in $S^+ \cup S^-$ then an S^+/S^--path is free to use them or not, whereas an S^+-path cannot use them. The key to prove Theorem 6.2 is to efficiently determine if S^+/S^--paths exist: we formalize this as a lemma which we will apply to the mapping DAG G restricted to the current level (in particular removing ϵ-edges):

▶ **Lemma 6.4.** *Let G be a mapping DAG with no ϵ-edges and let V be its vertex set. Given a non-empty set $\Lambda' \subseteq V$ of vertices of G and given two disjoint sets of labels S^+ and S^-, we can compute in time $O(|G| \times |S^+|)$ the set $\Lambda'_2 \subseteq V$ of vertices v such that there is an S^+/S^--path from one vertex of Λ' to v.*

Proof sketch. We first delete all edges from G with a label in S^-, add a fresh source vertex s_0, and remove all vertices that are not reachable from s_0. We then follow a topological sort of G to annotate each vertex v with the maximal set of labels of S^+ that can be seen along paths from s_0 to v: and we use a failure annotation $\emptyset$ when there are two such paths that can see two incomparable sets of labels of S^+. Indeed, as we argue, when this happens the vertex v can never be part of an S^+/S^--path because the definition of G imposes that each edge label occurs at most once on any path, so the partial paths from s_0 to v can never be completed with all missing labels from S^+. Hence, we can compute our set Λ'_2 simply by returning all the vertices annotated by the whole set S^+. ◀

We can now use Lemma 6.4 to prove Theorem 6.2:

Proof sketch of Theorem 6.2. We restrict our attention to the level $\mathsf{level}(\Lambda')$ of the mapping DAG G that contains the input level set Λ': in particular we remove all ϵ-edges. The resulting mapping DAG has size at most W_c, and we call $\mathcal{K}$ the set of labels that it uses, whose cardinality is at most the alphabet size B of G. We fix some arbitrary order on $\mathcal{K}$. Now, let us consider the full decision tree $T_{\mathcal{K}}$ on $\mathcal{K}$ following this order: it is a complete binary tree of height $|\mathcal{K}|$, each internal node at depth $0 \leq r < |\mathcal{K}|$ has two children reflecting on whether we take the r-th label of $\mathcal{K}$ or not, and each leaf n corresponds to a subset of $\mathcal{K}$ built according to the choices described on the path from the root of $T_{\mathcal{K}}$ to n. Our algorithm will explore $T_{\mathcal{K}}$ to find the sets S^+ of labels that we must enumerate for Λ' and G.

More precisely, we wish to determine the leaves of $T_{\mathcal{K}}$ that correspond to a set S^+ such that there is an S^+-path in G from a vertex of Λ' to a vertex with an outgoing ϵ-edge: we call this a *good* leaf. The naive way to find the good leaves would be to test them one after the other, but this would not ensure a good delay bound. Instead, we use the notion of S^+/S^--paths to only explore the relevant parts of $T_{\mathcal{K}}$. Following this idea, we say that an internal node n at depth $0 \leq r < |\mathcal{K}|$ of $T_{\mathcal{K}}$ is *good* if there is an S^+/S^- path from a vertex of Λ' to a vertex with an outgoing ϵ-edge, where S^+ and S^- respectively contain the labels of $\mathcal{K}$ that we decided to take and those that we decided not to take when going from the root of $T_{\mathcal{K}}$ to n. Note that S^+, S^- is a partition of the r first labels of $\mathcal{K}$ that uniquely defines n.

We can now use Lemma 6.4 as an oracle to determine, given any node n of the tree, whether n is good in this sense or not. This oracle makes it possible to find the good leaves of $T_{\mathcal{K}}$ efficiently, by starting at the root of $T_{\mathcal{K}}$ and doing a depth-first exploration of good nodes of the tree. We build $T_{\mathcal{K}}$ on-the-fly while doing so, to avoid materializing irrelevant parts of the tree. The exploration is guaranteed to find all good leaves, because the root of the tree is always good, and because the ancestors of a good leaf are always good. Further, it ensures that we always find one new good leaf after at most $O(|\mathcal{K}|)$ invocations of Lemma 6.4, because whenever we are at a good node then it must have a good child and therefore, by induction, a good descendant that is a leaf. We will find this leaf in our depth-first search with a number of oracle calls that is at most linear in the height of $T_{\mathcal{K}}$. Together with the delay bound of Lemma 6.4, this yields the claimed delay bound of $O(|W_c| \times B^2)$.

Last, it is clear that whenever we have found a good leaf corresponding to a set S^+, then we can compute the new level set Λ'' that we must return together with S^+, with the same delay bound. Indeed, we can simply do this by post-processing the set of vertices returned by the corresponding invocation of Lemma 6.4. ◀

Memory usage. The recursion depth of Algorithm 1 on general sequential VAs is unchanged, and we can still eliminate tail recursion for the case $\mathsf{locmark} = \emptyset$ as we did in Section 4.

The local space must now include the local space used by the enumeration scheme of NextLevel, of which there is an instance running at every level on the stack. We need to remember our current position in the binary search tree: assuming that the order of labels is fixed, it suffices to remember the current positive set P_n plus the last label in the order on $\mathcal{K}$ that we use, with all other labels being implicitly in N_n. This means that we store one label per level (the last label), plus the positive labels, so their total number in the stack is at most the total number of markers, i.e., $O(|\mathcal{V}|)$. Hence the structure of Theorem 6.2 has no effect on the memory usage.

The space usage must also include the space used for one call to the construction of Lemma 6.4, only one instance of which is running at every given time. This space usage is clearly in $O(|Q| \times |V|)$, so this additive term has again no impact on the memory usage. Hence, the memory usage of our enumeration algorithm is the same as in Section 4, i.e., $O((r+1) \times W)$, or $O((|\mathcal{V}|+1) \times |Q|)$ in terms of the VA.

7 Conclusion

We have shown that we can efficiently enumerate the mappings of sequential variable-set automata on input documents, achieving linear-time preprocessing and constant-delay in data complexity, while ensuring that preprocessing and delay are polynomial in the input VA even if it is not deterministic. This result was previously considered as unlikely by [11], and it improves on the algorithms in [15]: with our algorithm, the delay between outputs does not depend on the input document, whereas it had a linear dependency on the size of the input document in [15].

We will consider different directions for future works. A first question is how to cope with changes to the input document without recomputing our enumeration index structure from scratch. This question has been recently studied for other enumeration algorithms, see e.g. [3, 7, 8, 9, 19, 23, 24], but for atomic update operations: insertion, deletion, and relabelings of single nodes. However, as spanners operate on text, we would like to use bulk update operations that modify large parts of the text at once: cut and paste operations, splitting or joining strings, or appending at the end of a file and removing from the beginning, e.g., in the case of log files with rotation. It may be possible to show better bounds for these operations than the ones obtained by modifying each individual letter [24, 19].

A second question is to generalize our result from words to trees, but this is challenging: the run of a tree automaton is no longer linear in just one direction, so it is not easy to skip parts of the input similarly to the jump function of Section 5, or to combine computation that occurs in different branches. We believe that these difficulties can be solved and that a similar result can be shown for trees, but that the resulting algorithm is far more complex: this point, and the question of updates, are explored in our follow-up work [4].

Finally, it would be interesting to implement our algorithms and evaluate them on real-world data similarly to the work in [5, 22]. We believe that our techniques are rather simple and easily implementable, at least in the case of extended VAs. Moreover, since there are no large hidden constants in any of our constructions, we feel that they might be feasible in practice. Nevertheless, an efficient implementation would of course have to optimize implementation details that we could gloss over in our theoretical analysis since they make no difference in theory but might change practical behavior substantially.

References

1 Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The design and analysis of computer algorithms*. Addison-Wesley, 1974.

2 Antoine Amarilli, Pierre Bourhis, Louis Jachiet, and Stefan Mengel. A circuit-based approach to efficient enumeration. In *ICALP*, 2017. arXiv:1702.05589.

3 Antoine Amarilli, Pierre Bourhis, and Stefan Mengel. Enumeration on trees under relabelings. In *ICDT*, 2018. arXiv:1709.06185.

4 Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Enumeration on Trees with Tractable Combined Complexity and Efficient Updates. Under review, 2019. arXiv:1812.09519.

5 Marcelo Arenas, Francisco Maturana, Cristian Riveros, and Domagoj Vrgoc. A framework for annotating CSV-like data. *PVLDB*, 9(11), 2016. URL: http://www.vldb.org/pvldb/vol9/p876-arenas.pdf, doi:10.14778/2983200.2983204.

6 Guillaume Bagan. MSO queries on tree decomposable structures are computable with linear delay. In *CSL*, 2006.

7 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering Conjunctive Queries under Updates. In *PODS*, 2017. arXiv:1702.06370.

8 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering FO+MOD Queries Under Updates on Bounded Degree Databases. In *ICDT*, 2017. arXiv:1702.08764.

9 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering UCQs under Updates and in the Presence of Integrity Constraints. In *ICDT*, 2018. arXiv:1709.10039.

10 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Document Spanners: A Formal Approach to Information Extraction. *J. ACM*, 62(2), 2015. URL: https://pdfs.semanticscholar.org/8df0/ad1c6aa0df93e58071b8afe3371a16a3182f.pdf, doi:10.1145/2699442.

11 Fernando Florenzano, Cristian Riveros, Martín Ugarte, Stijn Vansummeren, and Domagoj Vrgoc. Constant Delay Algorithms for Regular Document Spanners. In *PODS*, 2018. arXiv:1803.05277.

12 Dominik D. Freydenberger. A Logic for Document Spanners. Unpublished extended version. URL: http://ddfy.de/sci/splog.pdf.

13 Dominik D. Freydenberger. A Logic for Document Spanners. In *ICDT*, 2017. URL: http://drops.dagstuhl.de/opus/volltexte/2017/7049/, doi:10.4230/LIPIcs.ICDT.2017.13.

14 Dominik D. Freydenberger and Mario Holldack. Document Spanners: From Expressive Power to Decision Problems. *Theory Comput. Syst.*, 62(4), 2018. doi:10.1007/s00224-017-9770-0.

15 Dominik D. Freydenberger, Benny Kimelfeld, and Liat Peterfreund. Joining Extractions of Regular Expressions. In *PODS*, 2018. arXiv:1703.10350.

16 François Le Gall. Improved output-sensitive quantum algorithms for Boolean matrix multiplication. In *SODA*, 2012. URL: https://pdfs.semanticscholar.org/91a5/dd90ed43a6e8f55f8ec18ceead7dd0a6e988.pdf.

17 François Le Gall. Powers of tensors and fast matrix multiplication. In *ISSAC*, 2014. arXiv:1401.7714.

18 Étienne Grandjean. Sorting, linear time and the satisfiability problem. *Annals of Mathematics and Artificial Intelligence*, 16(1), 1996.

19 Katja Losemann and Wim Martens. MSO queries on trees: Enumerating answers under updates. In *CSL-LICS*, 2014. URL: http://www.theoinf.uni-bayreuth.de/download/lics14-preprint.pdf.

20 Arnaud Mary and Yann Strozecki. Efficient Enumeration of Solutions Produced by Closure Operations. In *STACS*, 2016. URL: http://drops.dagstuhl.de/opus/volltexte/2016/5753/.

21 Francisco Maturana, Cristian Riveros, and Domagoj Vrgoc. Document Spanners for Extracting Incomplete Information: Expressiveness and Complexity. In *PODS*, 2018. arXiv:1707.00827.

22 Andrea Morciano. Engineering a runtime system for AQL. Master's thesis, Politecnico di Milano, 2017. URL: https://www.politesi.polimi.it/bitstream/10589/135034/1/2017_07_Morciano.pdf.

23 Matthias Niewerth. MSO queries on trees: Enumerating answers under updates using forest algebras. In *LICS*, 2018. doi:10.1145/3209108.3209144.

24 Matthias Niewerth and Luc Segoufin. Enumeration of MSO Queries on Strings with Constant Delay and Logarithmic Updates. In *PODS*, 2018. URL: http://www.di.ens.fr/~segoufin/Papers/Mypapers/enum-update-words.pdf.

25 Ronald C. Read and Robert E. Tarjan. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5(3), 1975.

26 IBM Research. SystemT, 2018. URL: https://researcher.watson.ibm.com/researcher/view_group.php?id=1264.

27 Luc Segoufin. A glimpse on constant delay enumeration (Invited talk). In *STACS*, 2014. URL: https://hal.inria.fr/hal-01070893/document.

28 Shuji Tsukiyama, Mikio Ide, Hiromu Ariyoshi, and I Shirakawa. A New Algorithm for Generating All the Maximal Independent Sets. *SIAM J. Comput.*, 6, September 1977. doi:10.1137/0206036.

29 L.G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2), 1979. URL: https://www.sciencedirect.com/science/article/pii/0304397579900446.

30 Kunihiro Wasa. Enumeration of enumeration algorithms. *CoRR*, 2016. arXiv:1605.05102.

Consistent Query Answering for Primary Keys in Logspace

Paraschos Koutris
University of Wisconsin-Madison, WI, USA
paris@cs.wisc.edu

Jef Wijsen
University of Mons, Belgium
jef.wijsen@umons.ac.be

Abstract

We study the complexity of consistent query answering on databases that may violate primary key constraints. A repair of such a database is any consistent database that can be obtained by deleting a minimal set of tuples. For every Boolean query q, CERTAINTY(q) is the problem that takes a database as input and asks whether q evaluates to true on every repair. In [Koutris and Wijsen, ACM TODS, 2017], the authors show that for every self-join-free Boolean conjunctive query q, the problem CERTAINTY(q) is either in **P** or **coNP**-complete, and it is decidable which of the two cases applies. In this paper, we sharpen this result by showing that for every self-join-free Boolean conjunctive query q, the problem CERTAINTY(q) is either expressible in symmetric stratified Datalog (with some aggregation operator) or **coNP**-complete. Since symmetric stratified Datalog is in **L**, we thus obtain a complexity-theoretic dichotomy between **L** and **coNP**-complete. Another new finding of practical importance is that CERTAINTY(q) is on the logspace side of the dichotomy for queries q where all join conditions express foreign-to-primary key matches, which is undoubtedly the most common type of join condition.

2012 ACM Subject Classification Information systems → Relational database model; Information systems → Inconsistent data; Information systems → Incomplete data; Information systems → Integrity checking

Keywords and phrases conjunctive queries, consistent query answering, Datalog, primary keys

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.23

Related Version A full version of this paper is available on arXiv [22], `https://arxiv.org/abs/1810.03386`.

1 Motivation

Consistent query answering (CQA) with respect to primary key constraints is the following problem. Given a database **db** that may violate its primary key constraints, define a repair as any consistent database that can be obtained by deleting a minimal set of tuples from **db**. For every Boolean query q, the problem CERTAINTY(q) takes a database as input and asks whether q evaluates to true on every repair of **db**. In this paper, we focus on CERTAINTY(q) for queries q in the class sjfBCQ, the class of self-join-free Boolean conjunctive queries. For all Boolean first-order queries q, CERTAINTY(q) is in **coNP** and can thus be solved by expressive formalisms like answer set programming [27] and binary integer programming [18]. These solutions, however, are likely to be inefficient when CERTAINTY(q) also belongs to a lower complexity class. In particular, given a query q in sjfBCQ, it is decidable [20] whether CERTAINTY(q) is in the low complexity class **FO**. Moreover, if CERTAINTY(q) is in **FO**, then it is possible to construct a first-order query for solving CERTAINTY(q), which is also called a *consistent first-order rewriting for* q. This construction is detailed in [20, Section 5] and has already been implemented [30].

In [20], the authors also show that for every query q in sjfBCQ, the problem CERTAINTY(q) is either in **P** or **coNP**-complete, and it is decidable (in polynomial time in the size of q) which of the two cases applies. The authors show how to construct a polynomial-time algorithm for CERTAINTY(q) when it does not lie on the **coNP**-hard side of the dichotomy. Unfortunately, unlike for consistent first-order rewritings, this construction is complex and does not tell us what language would be appropriate for implementing CERTAINTY(q) when it is in $\mathbf{P} \setminus \mathbf{FO}$. In this paper, we improve this situation: we show that if CERTAINTY(q) is in **P**, then it can be implemented in symmetric stratified Datalog, which has deterministic logspace data complexity [10]. We thus sharpen the complexity dichotomy of [20] as follows: for every query q in sjfBCQ, CERTAINTY(q) is either in **L** or **coNP**-complete. It is significant that Datalog is used as a target language, because this allows using optimized Datalog engines for solving CERTAINTY(q) whenever the problem lies on the logspace side of the dichotomy. Rewriting into Datalog is generally considered a desirable outcome when consistent first-order rewritings do not exist (see, e.g., [5, page 193]). It is also worth noting that the SQL:1999 standard introduced linear recursion into SQL, which has been implemented in varying ways in existing DBMSs [31]. Since the Datalog programs in this paper use only linear recursion, they may be partially or fully implementable in these DBMSs.

Throughout this paper, we use the term *consistent database* to refer to a database that satisfies all primary-key constraints, while the term *database* refers to both consistent and inconsistent databases. This is unlike most database textbooks, which tend to say that databases must always be consistent. The following definition introduces the main focus of this paper; the complexity dichotomy of Theorem 2 is the main result of this paper.

▶ **Definition 1.** *Let q be a Boolean query. Let $\mathcal{L}$ be some logic. A* consistent $\mathcal{L}$ rewriting for q *is a Boolean query P in $\mathcal{L}$ such that for every database* **db**, *P is true in* **db** *if and only if q is true in every repair of* **db**. *If q has a consistent $\mathcal{L}$ rewriting, then we say that* CERTAINTY(q) *is expressible in $\mathcal{L}$.*

▶ **Theorem 2.** *For every self-join-free Boolean conjunctive query q,* CERTAINTY(q) *is either* **coNP***-complete or expressible in SymStratDatalog*$^{\text{min}}$ *(and thus in* **L***).*

The language *SymStratDatalog*$^{\text{min}}$ will be defined in Section 3; informally, the superscript min means that the language allows selecting a minimum (with respect to some total order) from a finite set of values. Since CERTAINTY(q) is **L**-complete for some queries $q \in$ sjfBCQ, the logspace upper bound in Theorem 2 is tight. The proof of Theorem 2 relies on novel constructs and insights developed in this paper. Compared to [20], significant new contributions are the notion of *garbage set* and the helping Lemmas 11 and 22.

Our second significant result in this paper focuses on consistent query answering for foreign-to-primary key joins. In Section 9, we define a subclass of sjfBCQ that captures foreign-to-primary key joins, which is undoubtedly the most common type of join. We show that CERTAINTY(q) lies on the logspace side of the dichotomy for *all* queries q in this class. Thus, for the most common type of joins and primary key constraints, CQA is highly tractable, a result that goes against a widely spread belief that CQA would be impractical because of its high computational complexity.

Organization Section 2 discusses related work. Section 3 defines our theoretical framework, including the notion of *attack graph*. To guide the reader through the technical development, Section 4 provides a high-level outline of where we are heading in this paper, including examples of the different graphs used. Section 5 introduces a special subclass of sjfBCQ, called *saturated queries*, and shows that each problem CERTAINTY(q) can be first-order reduced to

some CERTAINTY(q') where q' is saturated. Section 6 introduces the notion of M-graph, a graph at the schema-level, and its data-level instantiation, called $\rightsquigarrow$-graph. An important result, Lemma 11, relates cycles in attack graphs to cycles in M-graphs, for saturated queries only. Section 7 introduces the notion of garbage set for a subquery. Informally, garbage sets contain facts that can never make the subquery hold true, and thus can be removed from the database without changing the answer to CERTAINTY(q). Section 8 focuses on cycles in the M-graph of a query, and shows that garbage sets for such cycles can be computed and removed in symmetric stratified Datalog. At the end of Section 8, we have all ingredients for the proof of our main theorem. Finally, Section 9 shows that foreign-to-primary key joins fall on the logspace side of the dichotomy. The proofs of lemmas and theorems appear in [22].

2 Related Work

Consistent query answering (CQA) starts from the seminal work by Arenas, Bertossi, and Chomicki [2], and is the topic of a monograph by Bertossi [7]. The term CERTAINTY(q) was coined in [33] to refer to CQA for Boolean queries q on databases that violate primary keys, one per relation, which are fixed by q's schema. The complexity classification of CERTAINTY(q) for all $q \in$ sjfBCQ started with the ICDT 2005 paper of Fuxman and Miller [13, 14], and has attracted much research since then. These previous works (see [35] for a survey) were generalized by [19, 20], where it was shown that the set {CERTAINTY(q) | $q \in$ sjfBCQ} exhibits a **P-coNP**-complete dichotomy. Furthermore, it was shown that membership of CERTAINTY(q) in **FO** is decidable for queries q in sjfBCQ. The current paper culminates this line of research by showing that the dichotomy is actually between **L** and **coNP**-complete, and – even stronger – between expressibility in symmetric stratified Datalog (with some aggregation operator) and **coNP**-complete.

The complexity of CERTAINTY(q) for self-join-free conjunctive queries with negated atoms was studied in [21]. Little is known about CERTAINTY(q) beyond self-join-free conjunctive queries. For UCQ (i.e., unions of conjunctive queries, possibly with self-joins), Fontaine [12] showed that a **P-coNP**-complete dichotomy in the set {CERTAINTY(q) | q is a Boolean query in UCQ} implies Bulatov's dichotomy theorem for conservative CSP [9]. This relationship between CQA and CSP was further explored in [26]. The complexity of CQA for aggregation queries with respect to violations of functional dependencies has been studied in [3].

The counting variant of CERTAINTY(q), which is called #CERTAINTY(q), asks to determine the number of repairs that satisfy some Boolean query q. In [28], the authors show a **FP**-#**P**-complete dichotomy in {#CERTAINTY(q) | $q \in$ sjfBCQ}. For conjunctive queries q with self-joins, the complexity of #CERTAINTY(q) has been established for the case that all primary keys consist of a single attribute [29]. In recent years, CQA has also been studied beyond the setting of relational databases, in ontology-based knowledge bases [8, 23] and in graph databases [6].

3 Preliminaries

We assume an infinite total order (**dom**, $\leq$) of *constants*. We assume a set of *variables* disjoint with **dom**. If $\vec{x}$ is a sequence containing variables and constants, then vars($\vec{x}$) denotes the set of variables that occur in $\vec{x}$. A *valuation* over a set U of variables is a total mapping θ from U to **dom**. At several places, it is implicitly understood that such a valuation θ is extended to be the identity on constants and on variables not in U. If $V \subseteq U$, then $\theta[V]$ denotes the restriction of θ to V. If θ is a valuation over a set U of variables, x is a variable (possibly $x \notin U$), and a is a constant, then $\theta_{[x \mapsto a]}$ is the valuation over $U \cup \{x\}$ such that $\theta_{[x \mapsto a]}(x) = a$ and for every variable y such that $y \neq x$, $\theta_{[x \mapsto a]}(y) = \theta(y)$.

Atoms and key-equal facts Each *relation name* R of arity n, $n \geq 1$, has a unique *primary key* which is a set $\{1, 2, \ldots, k\}$ where $1 \leq k \leq n$. We say that R has *signature* $[n, k]$ if R has arity n and primary key $\{1, 2, \ldots, k\}$. Elements of the primary key are called *primary-key positions*, while $k+1$, $k+2$, $\ldots$, n are *non-primary-key positions*. For all positive integers n, k such that $1 \leq k \leq n$, we assume denumerably many relation names with signature $[n, k]$. Every relation name has a unique *mode*, which is a value in $\{\mathsf{c}, \mathsf{i}\}$. Informally, relation names of mode c will be used for consistent relations, while relations that may be inconsistent will have a relation name of mode i. We often write R^{c} to make clear that R is a relation name of mode c. Relation names of mode c will be a convenient tool in the theoretical development, but they also constitute a useful modeling primitive that can be put at the disposal of end-users with domain knowledge [16].

If R is a relation name with signature $[n, k]$, then we call $R(s_1, \ldots, s_n)$ an *R-atom* (or simply atom), where each s_i is either a constant or a variable ($1 \leq i \leq n$). Such an atom is commonly written as $R(\underline{\vec{x}}, \vec{y})$ where the primary-key value $\vec{x} = s_1, \ldots, s_k$ is underlined and $\vec{y} = s_{k+1}, \ldots, s_n$. An *$R$-fact* (or simply fact) is an R-atom in which no variable occurs. Two facts $R_1(\underline{\vec{a}_1}, \vec{b}_1), R_2(\underline{\vec{a}_2}, \vec{b}_2)$ are *key-equal*, denoted $R_1(\underline{\vec{a}_1}, \vec{b}_1) \sim R_2(\underline{\vec{a}_2}, \vec{b}_2)$, if $R_1 = R_2$ and $\vec{a}_1 = \vec{a}_2$.

We will use letters F, G, H for atoms. For an atom $F = R(\underline{\vec{x}}, \vec{y})$, we denote by $\mathsf{key}(F)$ the set of variables that occur in $\vec{x}$, and by $\mathsf{vars}(F)$ the set of variables that occur in F, that is, $\mathsf{key}(F) = \mathsf{vars}(\vec{x})$ and $\mathsf{vars}(F) = \mathsf{vars}(\vec{x}) \cup \mathsf{vars}(\vec{y})$. We sometimes blur the distinction between relation names and atoms. For example, if F is an atom, then the term F-fact refers to a fact with the same relation name as F.

Databases, blocks, and repairs A *database schema* is a finite set of relation names. All constructs that follow are defined relative to a fixed database schema. A *database* is a finite set $\mathbf{db}$ of facts using only the relation names of the schema such that for every relation name R of mode c, no two distinct R-facts of $\mathbf{db}$ are key-equal.

A *relation* of $\mathbf{db}$ is a maximal set of facts in $\mathbf{db}$ that all share the same relation name. A *block* of $\mathbf{db}$ is a maximal set of key-equal facts of $\mathbf{db}$. A block of R-facts is also called an R-block. If A is a fact of $\mathbf{db}$, then $\mathsf{block}(A, \mathbf{db})$ denotes the block of $\mathbf{db}$ that contains A. If $A = R(\underline{\vec{a}}, \vec{b})$, then $\mathsf{block}(A, \mathbf{db})$ is also denoted by $R(\underline{\vec{a}}, \vec{*})$. A database $\mathbf{db}$ is *consistent* if no two distinct facts of $\mathbf{db}$ are key-equal (i.e., if no block of $\mathbf{db}$ contains more than one fact). A *repair* of $\mathbf{db}$ is a maximal (with respect to set inclusion) consistent subset of $\mathbf{db}$. We write $\mathsf{rset}(\mathbf{db})$ for the set of repairs of $\mathbf{db}$.

Boolean conjunctive queries A *Boolean query* is a mapping q that associates a Boolean (true or false) to each database, such that q is closed under isomorphism [24]. We write $\mathbf{db} \models q$ to denote that q associates true to $\mathbf{db}$, in which case $\mathbf{db}$ is said to *satisfy* q. A Boolean query q can be viewed as a decision problem that takes a database as input and asks whether $\mathbf{db}$ satisfies q. In this paper, the complexity class $\mathbf{FO}$ stands for the set of Boolean queries that can be defined in first-order logic with equality and constant symbols (which are interpreted as themselves), but without other built-in predicates or function symbols.

A *Boolean conjunctive query* is a finite set $q = \{R_1(\underline{\vec{x}_1}, \vec{y}_1), \ldots, R_n(\underline{\vec{x}_n}, \vec{y}_n)\}$ of atoms, without equality or built-in predicates. We denote by $\mathsf{vars}(q)$ the set of variables that occur in q. The set q represents the first-order sentence

$$\exists u_1 \cdots \exists u_k \left(R_1(\underline{\vec{x}_1}, \vec{y}_1) \wedge \cdots \wedge R_n(\underline{\vec{x}_n}, \vec{y}_n) \right),$$

where $\{u_1, \ldots, u_k\} = \mathsf{vars}(q)$. This query q is satisfied by a database $\mathbf{db}$ if there exists a valuation θ over $\mathsf{vars}(q)$ such that for each $i \in \{1, \ldots, n\}$, $R_i(\underline{\vec{a}}, \vec{b}) \in \mathbf{db}$ with $\vec{a} = \theta(\vec{x}_i)$ and $\vec{b} = \theta(\vec{y}_i)$.

We say that a Boolean conjunctive query q has a *self-join* if some relation name occurs more than once in q. If q has no self-join, then it is called *self-join-free*. We write sjfBCQ for the class of self-join-free Boolean conjunctive queries. If q is a query in sjfBCQ with an R-atom, then, by an abuse of notation, we sometimes write R to mean the R-atom of q.

Let θ be a valuation over some set X of variables. For every Boolean conjunctive query q, we write $\theta(q)$ for the query obtained from q by replacing all occurrences of each $x \in X \cap \mathsf{vars}(q)$ with $\theta(x)$; variables in $\mathsf{vars}(q) \setminus X$ remain unaffected (i.e., θ is understood to be the identity on variables not in X).

Atoms of mode c The *mode* of an atom is the mode of its relation name (a value in $\{\mathsf{c}, \mathsf{i}\}$). If q is a query in sjfBCQ, then q^{cons} is the set of all atoms of q that are of mode c.

Functional dependencies Let q be a Boolean conjunctive query. A *functional dependency for* q is an expression $X \to Y$ where $X, Y \subseteq \mathsf{vars}(q)$. Let $\mathcal{V}$ be a finite set of valuations over $\mathsf{vars}(q)$. We say that $\mathcal{V}$ *satisfies* $X \to Y$ if for all $\theta, \mu \in \mathcal{V}$, if $\theta[X] = \mu[X]$, then $\theta[Y] = \mu[Y]$. Let Σ be a set of functional dependencies for q. We write $\Sigma \models X \to Y$ if for every set $\mathcal{V}$ of valuations over $\mathsf{vars}(q)$, if $\mathcal{V}$ satisfies each functional dependency in Σ, then $\mathcal{V}$ satisfies $X \to Y$. Note that the foregoing conforms with standard dependency theory if variables are viewed as attributes, and valuations as tuples. As with standard functional dependencies, every set of functional dependencies for q is logically equivalent to a set of functional dependencies for q with singleton right-hand sides.

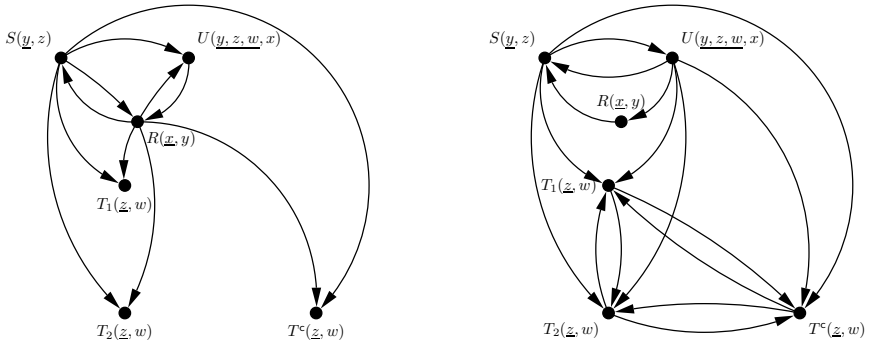
Consistent query answering Let q be a query in sjfBCQ. We define $\mathsf{CERTAINTY}(q)$ as the decision problem that takes as input a database $\mathbf{db}$, and asks whether every repair of $\mathbf{db}$ satisfies q.

The genre of a fact Let q be a query in sjfBCQ. For every fact A whose relation name occurs in q, we denote by $\mathsf{genre}_q(A)$ the (unique) atom of q that has the same relation name as A. From here on, if $\mathbf{db}$ is a database that is given as an input to $\mathsf{CERTAINTY}(q)$, we will assume that each relation name of each fact in $\mathbf{db}$ also occurs in q. Therefore, for every $A \in \mathbf{db}$, $\mathsf{genre}_q(A)$ is well defined. Of course, this assumption is harmless.

Attack graph Let q be a query in sjfBCQ. We define $\mathcal{K}(q)$ as the following set of functional dependencies: $\mathcal{K}(q) := \{\mathsf{key}(F) \to \mathsf{vars}(F) \mid F \in q\}$. For every atom $F \in q$, we define $F^{+,q}$ as the set of all variables $x \in \mathsf{vars}(q)$ satisfying $\mathcal{K}(q \setminus \{F\}) \cup \mathcal{K}(q^{\mathsf{cons}}) \models \mathsf{key}(F) \to x$. Informally, the term $\mathcal{K}(q^{\mathsf{cons}})$ is the set of all functional dependencies that arise in atoms of mode c. Clearly, if F has mode c, then $\mathcal{K}(q \setminus \{F\}) \cup \mathcal{K}(q^{\mathsf{cons}}) = \mathcal{K}(q)$; and if F has mode i, then $\mathcal{K}(q \setminus \{F\}) \cup \mathcal{K}(q^{\mathsf{cons}}) = \mathcal{K}(q \setminus \{F\})$. The *attack graph* of q is a directed graph whose vertices are the atoms of q. There is a directed edge from F to G ($F \neq G$), denoted $F \overset{q}{\rightsquigarrow} G$, if there exists a sequence

$$F_0 \overset{x_1}{\frown} F_1 \overset{x_2}{\frown} F_2 \cdots \overset{x_\ell}{\frown} F_\ell \tag{1}$$

such that $F_0 = F$, $F_\ell = G$, and for each $i \in \{1, \ldots, \ell\}$, F_i is an atom of q and x_i is a variable satisfying $x_i \in (\mathsf{vars}(F_{i-1}) \cap \mathsf{vars}(F_i)) \setminus F^{+,q}$. The sequence (1) is also called a *witness* for $F \overset{q}{\rightsquigarrow} G$. An edge $F \overset{q}{\rightsquigarrow} G$ is also called an *attack from* F *to* G; we also say that F *attacks* G. Informally, an attack from an atom $R(\vec{x}, \vec{y})$ to an atom $S(\vec{u}, \vec{w})$ indicates that, given a valuation over $\mathsf{vars}(\vec{x})$, the values for $\vec{u}$ that make the query true depend on the values chosen for $\vec{y}$.



Attack Graph. M-Graph.

Figure 1 Attack graph *(left)* and M-graph *(right)* of the same query $q_1 = \{R(\underline{x}, y), S(\underline{y}, z), U(\underline{y, z, w}, x), T_1(\underline{z}, w), T_2(\underline{z}, w), T^{\mathsf{c}}(\underline{z}, w)\}$. It can be verified that all attacks are weak and that the query is saturated. The attack graph has an initial strong component containing three atoms (R, S, and U). As predicted by Lemma 11, the subgraph of the M-graph induced by $\{R, S, U\}$ is cyclic.

An attack on a variable $x \in \mathsf{vars}(q)$ is defined as follows: $F \overset{q}{\rightsquigarrow} x$ if $F \overset{q \cup \{N(\underline{x})\}}{\rightsquigarrow} N(\underline{x})$ where N is a fresh relation name of signature $[1, 1]$. Informally, x is attacked in q if $N(\underline{x})$ has an incoming attack in the attack graph of $q \cup \{N(\underline{x})\}$.

▶ **Example 3.** Let $q_1 = \{R(\underline{x}, y), S(\underline{y}, z), U(\underline{y, z, w}, x), T_1(\underline{z}, w), T_2(\underline{z}, w), T^{\mathsf{c}}(\underline{z}, w)\}$. Using relation names for atoms, we have $R^{+,q_1} = \{x\}$. A witness for $R \overset{q_1}{\rightsquigarrow} U$ is $R \overset{y}{\frown} U$. The attack graph of q_1 is shown in Fig. 1.

An attack $F \overset{q}{\rightsquigarrow} G$ is *weak* if $\mathcal{K}(q) \models \mathsf{key}(F) \to \mathsf{key}(G)$; otherwise it is *strong*. A cycle in the attack graph is *strong* if at least one attack in the cycle is strong. It has been proved [20, Lemma 3.6] that if the attack graph contains a strong cycle, then it contains a strong cycle of length 2. The main result in [20] can now be stated.

▶ **Theorem 4** ([20])**.** *For every query q in* sjfBCQ*,*

- *if the attack graph of q is acyclic, then* CERTAINTY(q) *is in* **FO***;*
- *if the attack graph of q is cyclic but contains no strong cycle, then* CERTAINTY(q) *is* **L***-hard and in* **P***; and*
- *if the attack graph of q contains a strong cycle, then* CERTAINTY(q) *is* **coNP***-complete.*

Furthermore, it can be decided in quadratic time in the size of q which of these three cases applies.

Sequential proof Let q be a query in sjfBCQ. Let $Z \to w$ be a functional dependency for q with a singleton right-hand side (where set delimiters { and } are omitted). A *sequential proof for* $Z \to w$ is a (possibly empty) sequence $F_1, F_1, \ldots, F_\ell$ of atoms in q such that for every $i \in \{1, \ldots, \ell\}$, $\mathsf{key}(F_i) \subseteq Z \cup \left(\bigcup_{j=1}^{i-1} \mathsf{vars}(F_j)\right)$ and $w \in Z \cup (\bigcup_{j=1}^{\ell} \mathsf{vars}(F_j))$. Clearly, if $w \in \mathsf{vars}(F_k)$ for some $k < \ell$, then such a sequential proof can be shortened by omitting the atoms $F_{k+1}, \ldots, F_\ell$. Sequential proofs mimic the computation of a closure of a set of attributes with respect to a set of functional dependencies; see, e.g., [1, p. 165].

Notions from graph theory We adopt some terminology from [4]. A directed graph is *strongly connected* if there is a directed path from any vertex to any other. The maximal strongly connected subgraphs of a graph are vertex-disjoint and are called its *strong components*. If S_1 and S_2 are strong components such that an edge leads from a vertex in S_1 to a vertex in S_2, then S_1 is a *predecessor* of S_2 and S_2 is a *successor* of S_1. A strong component is called *initial* if it has no predecessor. For a directed graph, we define the length of a directed path as the number of edges it contains. A directed path or cycle without repeated vertices is called *elementary*. If G is a graph, then $V(G)$ denotes the vertex set of G, and $E(G)$ denotes the edge set of G.

Symmetric stratified Datalog We assume that the reader is familiar with the syntax and semantics of stratified Datalog. A stratified Datalog program is *linear* if in the body of each rule there is at most one occurrence of an IDB predicate of the same stratum (but there may be arbitrarily many occurrences of IDB predicates from lower strata). Assume that some stratum of a linear stratified Datalog program contains a recursive rule

$$L_0 \leftarrow L_1, L_2, \ldots, L_m, \neg L_{m+1}, \ldots, \neg L_n$$

such that L_1 is an IDB predicate of the same stratum. Then, since the program is linear, each predicate among $L_2, \ldots, L_n$ is either an EDB predicate or an IDB predicate of a lower stratum. Such a rule has a *symmetric rule*:

$$L_1 \leftarrow L_0, L_2, \ldots, L_m, \neg L_{m+1}, \ldots, \neg L_n.$$

A stratified Datalog program is *symmetric* if it is linear and the symmetric of any recursive rule is also a rule of the program.

It is known (see, for example, [15, Proposition 3.3.72]) that linear stratified Datalog is equivalent to Transitive Closure Logic. The data complexity of linear stratified Datalog is in **NL** (and is complete for **NL**). A symmetric Datalog program can be evaluated in logarithmic space [10] and cannot express directed reachability [11].

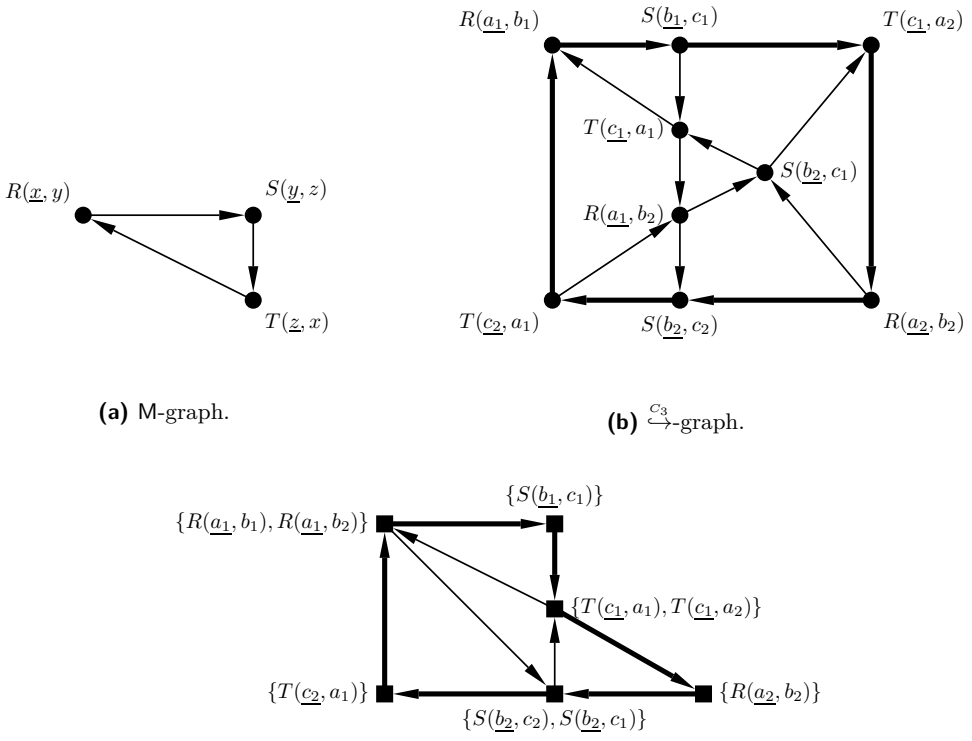
We will assume that given a (extensional or intentional) predicate P of some arity 2ℓ, we can express the following query (let $\vec{x} = \langle x_1, \ldots, x_\ell \rangle$, $\vec{y} = \langle y_1, \ldots, y_\ell \rangle$, and $\vec{z} = \langle z_1, \ldots, z_\ell \rangle$):

$$\{\vec{x}, \vec{y} \mid P(\vec{x}, \vec{y}) \wedge \forall z_1 \cdots \forall z_\ell \, (P(\vec{x}, \vec{z}) \rightarrow \vec{y} \leq_\ell \vec{z})\}, \tag{2}$$

where $\leq_\ell$ is a total order on $\mathbf{dom}^\ell$. Informally, the above query groups by the ℓ leftmost positions, and, within each group, takes the smallest (with respect to $\leq_\ell$) value for the remaining positions. Such a query will be useful in Section 8.3, where P encodes an equivalence relation on a finite subset of $\mathbf{dom}^\ell$, and the query (2) allows us to deterministically choose a representative in each equivalence class. The order $\leq_\ell$ can be first-order defined as the lexicographical order on $\mathbf{dom}^\ell$ induced by the linear order on $\mathbf{dom}$. For example, for $\ell = 2$, the lexicographical order is defined as $(y_1, y_2) \leq_2 (z_1, z_2)$ if $y_1 < z_1 \vee ((y_1 = z_1) \wedge (y_2 \leq z_2))$. Nevertheless, our results do not depend on how the order $\leq_\ell$ is defined. Moreover, all queries in our study will be order-invariant in the sense defined in [17]. The order is only needed in the proof of Lemma 25 to pick, in a deterministic way, an identifier from a set of candidate identifiers. In Datalog, we use the following convenient syntax for (2):

$$\mathsf{Answer}(\vec{x}, \min(\vec{y})) \leftarrow P(\vec{x}, \vec{y}).$$

Such a rule will always be non-recursive. Most significantly, if we extend a logspace fragment of stratified Datalog with queries of the form (2), the extended fragment will also be in



(a) M-graph.

(b) $\overset{C_3}{\hookrightarrow}$-graph.

(c) Block-quotient graph.

Figure 2 Examples of three different graphs used in this paper: M-graph, $\hookrightarrow$-graph, block-quotient graph.

logspace. Therefore, assuming queries of the form (2) is harmless for our complexity-theoretic purposes. We use *SymStratDatalog* for symmetric stratified Datalog, and *SymStratDatalog*$^{\min}$ for symmetric stratified Datalog that allows queries of the form (2). The need for the min operator will occur in the proof of Lemma 25.

4 The Main Theorem and an Informal Guide of its Proof

In this paper, we prove the following main result.

▶ **Theorem 5** (Main Theorem). *For every query q in* sjfBCQ*,*

- *if the attack graph of q contains a strong cycle, then* CERTAINTY(q) *is* **coNP***-complete; and*
- *if the attack graph of q contains no strong cycle, then* CERTAINTY(q) *is expressible in* *SymStratDatalog*$^{\min}$ *(and is thus in* **L***).*

The above result is stronger than Theorem 2, since it provides an effective criterion for the dichotomy between **coNP**-completeness and expressibility in symmetric stratified Datalog.

Before we delve into the proof in the next sections, we start with a guided tour that introduces our approach in an informal way. The focus of this paper is a deterministic logspace algorithm for CERTAINTY(q) whenever CERTAINTY(q) is in **P** but not in **FO**

(assuming $\mathbf{P} \neq \mathbf{coNP}$). In what follows, by a logspace algorithm, we will always mean a deterministic logspace algorithm. An exemplar query is $C_3 := \{R(\underline{x}, y), S(\underline{y}, z), T(\underline{z}, x)\}$, which can be thought of as a cycle of length 3. For the purpose of this example, let q be a query in sjfBCQ that includes C_3 as a subquery (i.e., $C_3 \subseteq q$).

An important novel notion in this paper is the M-graph of a query (see Section 6). The M-graph of C_3 is shown in Fig. 2a. Informally, a directed edge from an atom F to an atom G, denoted $F \xrightarrow{\mathsf{M}} G$, means that every variable that occurs in the primary key of G occurs also in F. In Fig. 2a, we have $T(\underline{z}, x) \xrightarrow{\mathsf{M}} R(\underline{x}, y)$, because R's primary key (i.e., x) occurs in the T-atom; there is no edge from $R(\underline{x}, y)$ to $T(\underline{z}, x)$ because z does not occur in the R-atom. Intuitively, one can think of edges in the M-graph as foreign-to-primary key joins. In what follows, we focus on cycles in the M-graph, called M-cycles. As we will see later on, such M-cycles will occur whenever the attack graph of a query is cyclic but contains no strong attack cycles.

Figure 2b shows an *instantiation* of the M-graph, called $\overset{C_3}{\hookrightarrow}$-graph (see Definitions 12 and 18), whose vertices are obtained by replacing variables with constants in $R(\underline{x}, y)$, $S(\underline{y}, z)$, or $T(\underline{z}, x)$. We write $A \overset{C_3}{\hookrightarrow} B$ to denote an edge from fact A to fact B. Each triangle in the $\overset{C_3}{\hookrightarrow}$-graph of Fig. 2b instantiates the query C_3; for example, the inner triangle is equal to $\theta(C_3)$ where θ is the valuation such that $\theta(xyz) = a_1b_2c_1$. We call such a triangle a 1-embedding (see Definition 18). Significantly, some edges are not part of any triangle. For example, the edge $S(\underline{b_1}, c_1) \overset{C_3}{\hookrightarrow} T(\underline{c_1}, a_2)$ is not in a triangle, but is present because the primary key of $T(\underline{c_1}, a_2)$ occurs in $S(\underline{b_1}, c_1)$.

Let **db** be a database that is input to CERTAINTY(q) such that **db** contains (but is not limited to) all facts of Fig. 2b. Since C_3 is a subquery of q, **db** will typically contain other facts with relation names in $q \setminus C_3$. Furthermore, **db** can contain R-facts, S-facts, and T-facts not shown in Fig. 2b. Then, **db** has at least $2^3 = 8$ repairs, because Fig. 2b shows two R-facts with primary key a_1, two S-facts with primary key b_2, and two T-facts with primary key c_1. Consider now the outermost elementary cycle (in thick lines) of length 6, i.e., the cycle using the vertices in $\mathbf{r} := \{R(\underline{a_1}, b_1), S(\underline{b_1}, c_1), T(\underline{c_1}, a_2), R(\underline{a_2}, b_2), S(\underline{b_2}, c_2), T(\underline{c_2}, a_1)\}$, which will be called a 2-embedding in Definition 18 (or an n-embedding with $n = 2$). One can verify that **r** does not contain distinct key-equal facts and does not satisfy C_3 (because the subgraph induced by **r** has no triangle). Let **o** be the database that contains **r** as well as all facts of **db** that are key-equal to some fact in **r**. A crucial observation is that if $\mathbf{db} \setminus \mathbf{o}$ has a repair that falsifies q, then so has **db** (the converse is trivially true). Indeed, if **s** is a repair of $\mathbf{db} \setminus \mathbf{o}$ that falsifies q, then $\mathbf{s} \cup \mathbf{r}$ is a repair of **db** that falsifies q. Intuitively, we can add **r** to **s** without creating a triangle in the $\overset{C_3}{\hookrightarrow}$-graph (i.e., without making C_3 true, and thus without making q true), because the facts in **r** form a cycle on their own and contain no outgoing $\overset{C_3}{\hookrightarrow}$-edges to facts in **s**. In Section 7, the set **o** will be called a *garbage set:* its facts can be thrown away without changing the answer to CERTAINTY(q). Note that the $\overset{C_3}{\hookrightarrow}$-graph of Fig. 2b contains other elementary cycles of length 6, which, however, contain distinct key-equal facts: for example, the cycle with vertices $R(\underline{a_1}, b_1)$, $S(\underline{b_1}, c_1)$, $T(\underline{c_1}, a_1)$, $R(\underline{a_1}, b_2)$, $S(\underline{b_2}, c_2)$, $T(\underline{c_2}, a_1)$ contains both $R(\underline{a_1}, b_1)$ and $R(\underline{a_1}, b_2)$.

Garbage sets thus arise from cycles in the $\overset{C_3}{\hookrightarrow}$-graph that *(i)* do not contain distinct key-equal facts, and *(ii)* are not triangles satisfying C_3. To find such cycles, we construct the quotient graph of the $\overset{C_3}{\hookrightarrow}$-graph with respect to the equivalence relation "is key-equal to." Since the equivalence classes with respect to "is key-equal to" are the *blocks* of the database, we call this graph the *block-quotient graph* (Definition 23). The block-quotient graph for our example is shown in Fig. 2c. The vertices are database blocks; there is an edge from

block $\mathbf{b}_1$ to $\mathbf{b}_2$ if the $\overset{C_3}{\hookrightarrow}$-graph contains an edge from some fact in $\mathbf{b}_1$ to some fact in $\mathbf{b}_2$. The block-quotient graph contains exactly one elementary directed cycle of length 6 (thick lines); this cycle obviously corresponds to the outermost cycle of length 6 in the $\overset{C_3}{\hookrightarrow}$-graph. A core result (Lemma 22) of this article is a logspace algorithm for finding elementary cycles in the block-quotient graph whose lengths are strict multiples of the length of the underlying M-cycle. In our example, since the M-cycle of C_3 has length 3, we are looking for cycles in the block-quotient graph of lengths $6, 9, 12, \ldots$ Note here that, since the $\overset{C_3}{\hookrightarrow}$-graph is tripartite, the length of any cycle in it must be a multiple of 3. Our algorithm can be encoded in symmetric stratified Datalog. This core algorithm is then extended to compute garbage sets (Lemma 24) for M-cycles.

In our example, C_3 is a subquery of q. In general, M-cycles will be subqueries of larger queries. The facts that belong to the garbage set for an M-cycle can be removed, but the other facts must be maintained for computations on the remaining part of the query, and are stored in a new schema that replaces the relations in the M-cycle with a single relation (see Section 8.3). In our example, this new relation has attributes for x, y, and z, and stores all triangles that are outside the garbage set for C_3.

We can now sketch our approach for dealing with queries q such that CERTAINTY(q) is in $\mathbf{P} \setminus \mathbf{FO}$. Lemma 11 tells us that such a query q will have an M-cycle involving two or more atoms of mode i. The garbage set of this M-cycle is then computed, and the facts not in the garbage set will be stored in a single new relation of mode i that replaces the M-cycle. In this way, CERTAINTY(q) is reduced to a new problem CERTAINTY(q'), where q' contains less atoms of mode i than q. Lemma 25 shows that this new problem will be in $\mathbf{P}$, and that our reduction can be expressed in symmetric stratified Datalog. We can repeat this reduction until we arrive at a query q'' such that CERTAINTY(q'') is in $\mathbf{FO}$.

To conclude this guided tour, we point out the role of atoms of mode c in the computation of the M-graph, which was not illustrated by our running example. In the M-graph of Fig. 1 *(right)*, we have $S(\underline{y}, z) \xrightarrow{\mathsf{M}} U(\underline{y, z, w}, x)$, even though w does not occur in the S-atom. The explanation is that the query also contains the consistent relation $T^{\mathsf{c}}(\underline{z}, w)$, which maps each z-value to a unique w-value. So even though w does not occur as such in $S(\underline{y}, z)$, it is nevertheless uniquely determined by z. It is thus important to identify all relations of mode c, which is the topic of the next section.

5 Saturated Queries

In this section, we show that we can safely extend a query q with new consistent relations. To achieve this, we need to identify a particular type of functional dependencies for q, which are called *internal*. Internal functional dependencies are used to define *saturated* queries.

▶ **Definition 6.** *Let q be a query in* sjfBCQ. *Let $Z \to w$ be a functional dependency for q. We say that $Z \to w$ is* internal to q *if the following two conditions are satisfied:*

1. *there exists a sequential proof for $\mathcal{K}(q) \models Z \to w$ such that no atom in the sequential proof attacks a variable in $Z \cup \{w\}$; and*
2. *for some $F \in q$, $Z \subseteq \mathsf{vars}(F)$.*

We say that q is saturated *if for every functional dependency σ that is internal to q, we have $\mathcal{K}(q^{\mathsf{cons}}) \models \sigma$.*

▶ **Example 7.** Assume $q = \{S_1(\underline{z}, u), S_2(\underline{u}, w), R_1(\underline{z}, u'), R_2(\underline{u'}, w), T_1(\underline{u}, v), T_2(\underline{v}, w)\}$. By using relation names as a shorthand for atoms, we have that $\langle S_1, S_2 \rangle$ is a sequential proof for $\mathcal{K}(q) \models z \to w$ in which neither S_1 nor S_2 attacks z or w, Indeed, S_1 attacks neither z nor w because $z, w \in {S_1}^{+,q}$. S_2 attacks no variable because $\mathsf{vars}(S_2) \subseteq {S_2}^{+,q}$. It follows that the functional dependency $z \to w$ is internal to q.

The next key lemma shows that we can assume without loss of generality that every internal functional dependency $Z \to w$ is satisfied, i.e., that every Z-value is mapped to a unique w-value. Therefore, whenever $Z \to w$ is internal, we can safely extend q with a new consistent relation $N^{\mathsf{c}}(\underline{Z}, w)$ that materializes the mapping from Z-values to w-values. Continuing the above example, we would extend q by adding a fresh atom $N^{\mathsf{c}}(\underline{z}, w)$.

▶ **Lemma 8.** *For every query q in* sjfBCQ*, it is possible to compute a query q' in* sjfBCQ *with the following properties:*

1. *there exists a first-order reduction from* CERTAINTY(q) *to* CERTAINTY(q')*;*
2. *if the attack graph of q contains no strong cycle, then the attack graph of* CERTAINTY(q') *contains no strong cycle; and*
3. *q' is saturated.*

6 M-Graphs and $\hookrightarrow$-Graphs

In this section, we introduce the M-*graph* of a query q in sjfBCQ, which is a generalization of the notion of Markov-graph introduced in [20] (hence the use of the letter M). An important new result, Lemma 11, expresses a relationship between attack graphs and M-graphs. Finally, we define $\hookrightarrow$-graphs, which can be regarded as data-level instantiations of M-graphs.

▶ **Definition 9.** *Let q be a query in* sjfBCQ *(which need not be saturated). The* M-graph *of q is a directed graph whose vertices are the atoms of q. There is a directed edge from F to G ($F \neq G$), denoted $F \xrightarrow{\mathsf{M}} G$, if $\mathcal{K}(q^{\mathsf{cons}}) \models \mathsf{vars}(F) \to \mathsf{key}(G)$. A cycle in the* M-*graph is called an* M-cycle*.*

Note that if all relation names in q have mode i, then $F \xrightarrow{\mathsf{M}} G$ implies $\mathsf{key}(G) \subseteq \mathsf{vars}(F)$. M-Graphs are technically easier to deal with than the Markov-graphs [20] on which they are inspired. In fact, Markov-graphs were in [20] only defined for queries containing no atoms of mode i with a composite primary key. Therefore, atoms with composite primary keys had first to be massaged into the form required by Markov graphs. This drawback is resolved by the new notion of M-graph.

▶ **Example 10.** The notion of M-graph is illustrated by Fig. 1. We have $\mathcal{K}({q_1}^{\mathsf{cons}}) = \{z \to w\}$. Since $\mathcal{K}({q_1}^{\mathsf{cons}}) \models \mathsf{vars}(S) \to \mathsf{key}(U)$, the M-graph has a directed edge from S to U.

The following lemma tells us about how the existence of M-cycles goes hand in hand with weak attack cycles. This relationship is important because, on the one hand, our logspace algorithm for CERTAINTY(q), with $q \in$ sjfBCQ, is centered on the existence of M-cycles, and, on the other, it must apply whenever all cycles in q's attack graph are weak. The notion of saturated query is also needed here, because the lemma fails for queries that are not saturated. The lemma generalizes Lemma 7.13 in [20]. Note that the lemma only considers strong components of the attack graph that are initial, which will be sufficient for our purposes.

▶ **Lemma 11.** *Let q be a query in* sjfBCQ *such that q is saturated and the attack graph of q contains no strong cycle. Let $\mathcal{S}$ be an initial strong component in the attack graph of q with $|\mathcal{S}| \geq 2$. Then, the* M-*graph of q contains a cycle all of whose atoms belong to $\mathcal{S}$.*

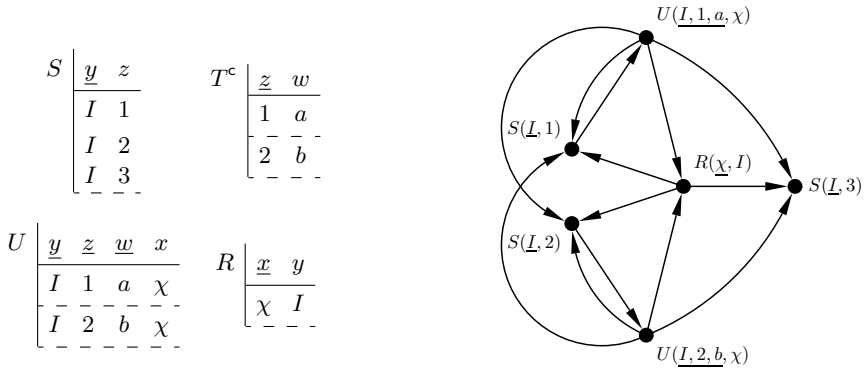


Figure 3 *Left:* Database that is input to CERTAINTY(q_1) for the query q_1 in Fig. 1. The relations for T_1 and T_2, which are identical to the relation for T^{c}, have been omitted. *Right:* The $\rightsquigarrow$-graph from which, for readability reasons, T_1-facts, T_2-facts, and T^{c}-facts have been omitted.

Given a query q, every database that instantiates the schema of q naturally gives rise to an instantiation of the $\xrightarrow{\mathsf{M}}$-edges in q's M-graph, in a way that is captured by the following definition.

▶ **Definition 12.** *The following notions are defined relative to a query q in* sjfBCQ *and a database* **db**. *The* $\rightsquigarrow$-graph *of* **db** *is a directed graph whose vertices are the atoms of* **db**. *There is a directed edge from A to B, denoted $A \rightsquigarrow B$, if there exists a valuation θ over* vars(q) *and an edge $F \xrightarrow{\mathsf{M}} G$ in the* M*-graph of q such that $\theta(q) \subseteq$ **db**, $A = \theta(F)$, and $B \sim \theta(G)$. A cycle in the $\rightsquigarrow$-graph is also called a $\rightsquigarrow$-cycle. In spoken language, the $\rightsquigarrow$-graph may be called the* instantiated M-graph.

The notion of $\rightsquigarrow$-graph is illustrated by Fig. 3. The following lemma states that if the $\rightsquigarrow$-graph of a database **db** has a directed edge from some fact A to some G-fact B, then A has outgoing edges to all the facts of block(B, **db**), and to no other G-facts.

▶ **Lemma 13.** *Let $q \in$ sjfBCQ and let* **db** *be a database. Let $A, B \in$ **db** and $F, G \in q$.*
1. *if $A \rightsquigarrow B$, then $A \rightsquigarrow B'$ for all $B' \in$ block(B, **db**);*
2. *if $A \rightsquigarrow B$ and $A \rightsquigarrow B'$ and $\mathsf{genre}_q(B) = \mathsf{genre}_q(B')$, then $B \sim B'$.*

7 Garbage Sets

Let **db** be a database that is an input to CERTAINTY(q) with $q \in$ sjfBCQ. In this section, we show that it is generally possible to downsize **db** by deleting blocks from it without changing the answer to CERTAINTY(q). That is, if the downsized database has a repair falsifying q, then so does the original database (the converse holds trivially true). Intuitively, the deleted blocks can be considered as "garbage" for the problem CERTAINTY(q).

▶ **Definition 14.** *The following definition is relative to a fixed query q in* sjfBCQ. *Let $q_0 \subseteq q$. Let* **db** *be a database. We say that a subset* **o** *of* **db** *is a* garbage set for q_0 in **db** *if the following conditions are satisfied:*
1. *for every $A \in$ **o**, we have that* $\mathsf{genre}_q(A) \in q_0$ *and* block(A, **db**) $\subseteq$ **o**; *and*
2. *there exists a repair* **r** *of* **o** *such that for every valuation θ over* vars(q), *if $\theta(q) \subseteq$ (**db** $\setminus$ **o**)$\cup$**r**, then $\theta(q_0) \cap$ **r** $= \emptyset$ (and thus $\theta(q_0) \cap$ **o** $= \emptyset$).*

The first condition in the above definition says that the relation names of facts in $\mathbf{o}$ must occur in q_0, and that every block of $\mathbf{db}$ is either included in or disjoint with $\mathbf{o}$. The second condition captures the crux of the definition and was illustrated in Section 4.

We now show a number of useful properties of garbage sets that are quite intuitive. In particular, by Lemma 15, there exists a unique maximum (with respect to $\subseteq$) garbage set for q_0 in $\mathbf{db}$, which will be called *the maximum garbage set for q_0 in $\mathbf{db}$*.

▶ **Lemma 15.** *Let q be a query in* sjfBCQ*, and let $q_0 \subseteq q$. Let $\mathbf{db}$ be a database. If $\mathbf{o}_1$ and $\mathbf{o}_2$ are garbage sets for q_0 in $\mathbf{db}$, then $\mathbf{o}_1 \cup \mathbf{o}_2$ is a garbage set for q_0 in $\mathbf{db}$.*

▶ **Lemma 16.** *Let q be a query in* sjfBCQ*, and let $q_0 \subseteq q$. Let $\mathbf{db}$ be a database. Let $\mathbf{o}$ be a garbage set for q_0 in $\mathbf{db}$. Then, every repair of $\mathbf{db}$ satisfies q if and only if every repair of $\mathbf{db} \setminus \mathbf{o}$ satisfies q (i.e., $\mathbf{db}$ and $\mathbf{db} \setminus \mathbf{o}$ agree on their answer to* CERTAINTY(q)*).*

▶ **Lemma 17.** *Let q be a query in* sjfBCQ*, and let $q_0 \subseteq q$. Let $\mathbf{db}$ be a database. Let $\mathbf{o}$ be a garbage set for q_0 in $\mathbf{db}$. Then, every garbage set for q_0 in $\mathbf{db} \setminus \mathbf{o}$ is empty if and only if $\mathbf{o}$ is the maximal garbage set for q_0 in $\mathbf{db}$.*

8 Garbage Sets for M-Cycles

In this section, we bring together notions of the two preceding sections. We focus on queries q in sjfBCQ whose M-graph has a cycle C. From here on, if C is an elementary cycle in the M-graph of some query q in sjfBCQ, then the subset of q that contains all (and only) the atoms of C, is also denoted by C.

Section 8.1 shows a procedural characterization of the maximal garbage set for C. Section 8.2 shows that the maximal garbage set for C can be computed in symmetric stratified Datalog. Finally, Section 8.3 shows a reduction, expressible in symmetric stratified Datalog, that replaces C with a single atom.

8.1 Characterizing Garbage Sets for M-Cycles

We define how a given M-cycle C of length k can be instantiated by cycles in the $\rightsquigarrow$-graph, called *embeddings*, whose lengths are multiples of k.

▶ **Definition 18.** *Let q be a query in* sjfBCQ*. Let $\mathbf{db}$ be a database. Let C be an elementary directed cycle in the* M*-graph of q. The cycle C naturally induces a subgraph of the $\rightsquigarrow$-graph, as follows: the vertex set of the subgraph contains all (and only) the facts A of $\mathbf{db}$ such that* $\mathsf{genre}_q(A)$ *is an atom in C; there is a directed edge from A to B, denoted $A \overset{C}{\rightsquigarrow} B$, if $A \rightsquigarrow B$ and the cycle C contains a directed edge from* $\mathsf{genre}_q(A)$ *to* $\mathsf{genre}_q(B)$*.*

Let k be the length of C. Obviously, the length of every $\overset{C}{\rightsquigarrow}$-cycle must be a multiple of k. Let n be a positive integer. An n-embedding of C in $\mathbf{db}$ *(or simply* embedding *if the value n is not important) is an elementary $\overset{C}{\rightsquigarrow}$-cycle of length nk containing no two distinct key-equal facts. A* 1*-embedding of C in $\mathbf{db}$ is said to be* relevant *if there exists a valuation θ over* $\mathsf{vars}(q)$ *such that $\theta(q) \subseteq \mathbf{db}$ and $\theta(q)$ contains every fact of the* 1*-embedding; otherwise the* 1*-embedding is said to be* irrelevant*.*

Let C and q be as in Definition 18, and let $\mathbf{db}$ be a database. There exists an intimate relationship between garbage sets for C in $\mathbf{db}$ and different sorts of embeddings.

- Let $A \in \mathbf{db}$ such that $\mathsf{genre}_q(A)$ belongs to C. If A belongs to some relevant 1-embedding of C in $\mathbf{db}$, then A will have an outgoing edge in the $\overset{C}{\rightsquigarrow}$-graph. If A does not belong to some relevant 1-embedding of C in $\mathbf{db}$, then A will have no outgoing edge in the $\overset{C}{\rightsquigarrow}$-graph, and $\mathsf{block}(A, \mathbf{db})$ is a garbage set for C in $\mathbf{db}$ by Definition 14 (choose $\mathbf{o} = \mathsf{block}(A, \mathbf{db})$ and $\mathbf{r} = \{A\}$).

- Every irrelevant 1-embedding of C in $\mathbf{db}$ gives rise to a garbage set. To illustrate this case, let $C = \{R(\underline{x}, y, z),\ S(\underline{y}, x, z)\}$. Assume that $R(\underline{a}, b, 1) \overset{C}{\hookrightarrow} S(\underline{b}, a, 2) \overset{C}{\hookrightarrow} R(\underline{a}, b, 1)$ is a 1-embedding of C in $\mathbf{db}$. This 1-embedding is irrelevant, because $1 \neq 2$. It can be easily seen that $R(\underline{a}, *, *) \cup S(\underline{b}, *, *)$ is a garbage set for q in $\mathbf{db}$.
- Every n-embedding of C in $\mathbf{db}$ with $n \geq 2$ gives rise to a garbage set. This was illustrated in Section 4 by means of the outermost cycle of length 6 in Fig. 2b, which is a 2-embedding of $\{R(\underline{x}, y),\ S(\underline{y}, z),\ T(\underline{z}, x)\}$.

These observations lead to the following lemma which provides a procedural characterization of the maximal garbage set for C in a given database.

▶ **Lemma 19.** *Let q be a query in* sjfBCQ*. Let $C = F_0 \xrightarrow{\mathsf{M}} F_1 \xrightarrow{\mathsf{M}} \cdots \xrightarrow{\mathsf{M}} F_{k-1} \xrightarrow{\mathsf{M}} F_0$ be an elementary cycle of length k ($k \geq 2$) in the* M*-graph of q. Let $\mathbf{db}$ be a database. Let $\mathbf{o}$ be a minimal (with respect to $\subseteq$) subset of $\mathbf{db}$ satisfying the following conditions:*

1. *the set $\mathbf{o}$ contains every fact A of $\mathbf{db}$ with $\mathsf{genre}_q(A) \in \{F_0, \ldots, F_{k-1}\}$ such that A has zero outdegree in the $\overset{C}{\hookrightarrow}$-graph;*
2. *the set $\mathbf{o}$ contains every fact that belongs to some irrelevant 1-embedding of C in $\mathbf{db}$;*
3. *the set $\mathbf{o}$ contains every fact that belongs to some n-embedding of C in $\mathbf{db}$ with $n \geq 2$;*
4. Recursive condition: *if $\mathbf{o}$ contains some fact of a relevant 1-embedding of C in $\mathbf{db}$, then $\mathbf{o}$ contains every fact of that 1-embedding; and*
5. Closure under "is key-equal to": *if $\mathbf{o}$ contains some fact A, then $\mathbf{o}$ includes $\mathsf{block}(A, \mathbf{db})$.*

Then, $\mathbf{o}$ is the maximal garbage set for C in $\mathbf{db}$.

▶ **Corollary 20.** *Let C be an elementary cycle in the* M*-graph of a query q in* sjfBCQ*. Let $\mathcal{S}$ be a strong component in the $\overset{C}{\hookrightarrow}$-graph of a database $\mathbf{db}$. If some fact of $\mathcal{S}$ belongs to the maximal garbage set for C in $\mathbf{db}$, then every fact of $\mathcal{S}$ belongs to the maximal garbage set for C in $\mathbf{db}$.*

8.2 Computing Garbage Sets for M-Cycles

In this section, we translate Lemma 19 into a Datalog program that computes, in deterministic logspace, the maximal garbage set for an M-cycle C. The main computational challenge lies in condition 3 of Lemma 19, which adds to the maximal garbage set all facts belonging to some n-embedding with $n \geq 2$, where the value of n is not upper bounded. Such n-embeddings can obviously be computed in nondeterministic logspace by using directed reachability in the $\overset{C}{\hookrightarrow}$-graph. This section shows a trick that allows doing the computation by using only *undirected* reachability, which, by the use of Reingold's algorithm [32], will lead to an algorithm that runs in deterministic logspace.

By Corollary 20, instead of searching for n-embeddings, $n \geq 2$, it suffices to search for strong components of the $\overset{C}{\hookrightarrow}$-graph containing such n-embeddings. These strong components can be recognized by a first-order reduction to the following problem, called $\mathsf{LONGCYCLE}(k)$, which is in logspace by Lemma 22.

▶ **Definition 21.** *A* k-circle-layered graph *[25] is a k-partite directed graph $G = (V, E)$ where edges only exist between adjacent partitions. More formally, the vertices of G can be partitioned into k groups such that $V = V_0 \cup V_1 \cup \cdots \cup V_{k-1}$ and $V_i \cap V_j = \emptyset$ if $i \neq j$. The only edges from a partition V_i go to the partition $V_{(i+1) \bmod k}$.*

For every positive integer k, $\mathsf{LONGCYCLE}(k)$ is the following problem.

Problem *$\mathsf{LONGCYCLE}(k)$*

Instance *A connected k-circle-layered graph $G = (V, E)$ such that every edge of E belongs to a directed cycle of length k.*

Question *Does G have an elementary directed cycle of length at least $2k$?*

▶ **Lemma 22.** *For every positive integer k, LONGCYCLE(k) is in* **L** *and can be expressed in SymStratDatalog.*

Proof (Sketch). Let $G = (V, E)$ be an instance of LONGCYCLE(k). A cycle of length k in G is called a *k-cycle*. Let $\widehat{G}$ be the undirected graph whose vertices are the k-cycles of G; there is an undirected edge between two vertices if their k-cycles have an element in common. The full proof in [22] shows that G has an elementary directed cycle of length $\geq 2k$ if and only if one of the following conditions is satisfied:

- for some n such that $2 \leq n \leq 2k-3$, G has an elementary directed cycle of length nk; or
- $\widehat{G}$ has a chordless undirected cycle (i.e., a cycle without cycle chord) of length $\geq 2k$.

The first condition can be tested in **FO**; the second condition can be reduced to an undirected connectivity problem, which is in logspace [32] and can be expressed in *SymStratDatalog*. ◀

To use Lemma 22, we take a detour via the quotient graph of the $\overset{C}{\hookrightarrow}$-graph relative to the equivalence relation "is key-equal to."

▶ **Definition 23.** *Let q be a query in* sjfBCQ*. Let* **db** *be a database. Let C be an elementary directed cycle of length $k \geq 2$ in the* M*-graph of q. The* block-quotient graph *is the quotient graph of the $\overset{C}{\hookrightarrow}$-graph of* **db** *with respect to the equivalence relation $\sim$.* [1]

The block-quotient graph of a database can obviously be constructed in **FO**. The strong components of the $\overset{C}{\hookrightarrow}$-graph that contain some n-embedding of C, $n \geq 2$, can then be recognized in logspace by executing the algorithm for LONGCYCLE(k) on the block-quotient graph.

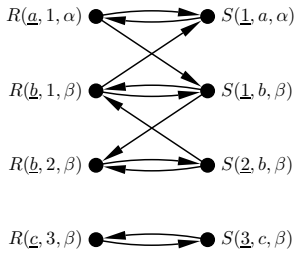
▶ **Lemma 24.** *Let q be a query in* sjfBCQ*. Let C be an elementary cycle of length k ($k \geq 2$) in the* M*-graph of q. There exists a program in SymStratDatalog that takes a database* **db** *as input and returns, as output, the maximal garbage set for C in* **db**.

8.3 Elimination of M-Cycles

Given a database **db**, the Datalog program of Lemma 24 allows us to compute the maximal garbage set **o** for C in **db**. The $\overset{C}{\hookrightarrow}$-graph of $\mathbf{db}' := \mathbf{db} \setminus \mathbf{o}$ will be a set of strong components, all initial, each of which is a collection of relevant 1-embeddings of C in $\mathbf{db}'$. The following Lemma 25 introduces a reduction that encodes this $\overset{C}{\hookrightarrow}$-graph by means of a fresh atom $T(\underline{u}, \vec{w})$, where $\mathsf{vars}(\vec{w}) = \mathsf{vars}(C)$ and u is a fresh variable. Whenever $\theta(q) \subseteq \mathbf{db}'$ for some valuation θ over $\mathsf{vars}(q)$, the reduction will add to the database a fact $T(\underline{cid}, \theta(\vec{w}))$ where *cid* is an identifier for the strong component (in the $\overset{C}{\hookrightarrow}$-graph) that contains $\theta(C)$. The construction is illustrated by Fig. 4. The following lemma captures this reduction and states that it *(i)* is expressible in $\mathit{SymStratDatalog}^{\min}$, and *(ii)* does not result in an increase of computational complexity.

▶ **Lemma 25.** *Let q be a query in* sjfBCQ*. Let $C = F_0 \xrightarrow{\mathsf{M}} F_1 \xrightarrow{\mathsf{M}} \cdots \xrightarrow{\mathsf{M}} F_{k-1} \xrightarrow{\mathsf{M}} F_0$ with $k \geq 2$ be an elementary cycle in the* M*-graph of q. Let u be a variable such that $u \notin \mathsf{vars}(q)$. Let T be an atom with a fresh relation name such that $\mathsf{key}(T) = \{u\}$ and $\mathsf{vars}(T) = \mathsf{vars}(C) \cup \{u\}$. Let p be a set containing, for every $i \in \{1, \ldots, k\}$, an atom N_i of mode* c *with a fresh relation name such that $\mathsf{key}(N_i) = \mathsf{key}(F_i)$ and $\mathsf{vars}(N_i) = \mathsf{key}(F_i) \cup \{u\}$. Then,*

[1] The quotient graph of a directed graph $G = (V, E)$ with respect to an equivalence relation $\equiv$ on V is a directed graph whose vertices are the equivalence classes of $\equiv$; there is a directed edge from class A to class B if E has a directed edge from some vertex in A to some vertex in B.



T

u	x	y	z
a	a	1	α
a	b	1	β
a	b	2	β
c	c	3	β

N_1^{c}

x	u
a	a
b	a
c	c

N_2^{c}

y	u
1	a
2	a
3	c

Figure 4 *Left:* Two strong components in the $\overset{C}{\hookrightarrow}$-graph of a database for an M-cycle $R(\underline{x}, y, z) \xrightarrow{\mathsf{M}} S(\underline{y}, x, z) \xrightarrow{\mathsf{M}} R(\underline{x}, y, z)$. The maximal garbage set is empty. *Right:* Encoding of the relevant 1-embeddings in each strong component. The u-values a and c are used to identify the strong components, and are chosen as the smallest x-values in each strong component.

1. *there exists a reduction from* CERTAINTY(q) *to* CERTAINTY$((q \setminus C) \cup \{T\} \cup p)$ *that is expressible in SymStratDatalog*$^{\min}$*; and*
2. *if the attack graph of q contains no strong cycle and some initial strong component of the attack graph contains every atom of $\{F_0, F_1, \ldots, F_{k-1}\}$, then the attack graph of $(q \setminus C) \cup \{T\} \cup p$ contains no strong cycle either.*

Proof (Crux). The crux in the proof of the first item is the deterministic choice of u-values for T-blocks. In Fig. 4, for example, the T-block encoding the top strong component uses $u = a$, and the T-block encoding the bottom strong component uses $u = c$. These u-values are the smallest x-values in the strong components, which can be obtained by the query (2) introduced in Section 3. In the example, we assumed $a = \min\{a, b\}$ and $c = \min\{c\}$. ◀

The proof of the main theorem, Theorem 5, is now fairly straightforward and is given in full detail in [22]. Informally, let q be a saturated query in sjfBCQ such that the attack graph of q has no strong attack cycles. If q contains an atom of mode i without incoming attacks, then this atom is rewritten in first-order logic, in the form defined by [34, Definition 8.3]; otherwise some M-cycle, which exists by Lemma 11, is eliminated in the way previously described in this section. In either case, the remaining smaller query will have a consistent *SymStratDatalog*$^{\min}$ rewriting.

9 Joins on Primary Keys

It is common that the join condition in a join of two tables expresses a foreign-to-primary key match, i.e., the columns (called the foreign key) of one table reference the primary key of another table. In our setting, we have primary keys but no foreign keys. Nevertheless, foreign keys can often be inferred from the query. For example, in the following query, the variable d in Movies references the primary key of Directors:

$$\{\mathsf{Movies}(\underline{m}, t, \text{'1963'}, d), \mathsf{Directors}(\underline{d}, \text{'Hitchcock'}, b)\}.$$

Given relation schemas Movies(M#, Title, Year, Director) and Directors(D#, Name, BirthYear), this query asks whether there exists a movie released in 1963 and directed by Hitchcock.

The *key-join property* that we define below captures this common type of join. Informally, a query has the key-join property if whenever two atoms have a variable in common, then their set of shared variables is either equal to the set of primary-key variables of one of the atoms, or contains all primary-key variables of both atoms.

▶ **Definition 26.** *We say that a query q in* sjfBCQ *has the* key-join property *if for all $F, G \in q$, either* $\mathsf{vars}(F) \cap \mathsf{vars}(G) \in \{\emptyset, \mathsf{key}(F), \mathsf{key}(G)\}$ *or* $\mathsf{vars}(F) \cap \mathsf{vars}(G) \supseteq \mathsf{key}(F) \cup \mathsf{key}(G)$.

Theorem 27 shows that if some query q in sjfBCQ has the key-join property, then CERTAINTY(q) falls on the logspace side of the dichotomy of Theorem 5.

▶ **Theorem 27.** *For every query q in* sjfBCQ *that has the key-join property,* CERTAINTY(q) *is expressible in SymStratDatalog*$^{\min}$ *(and is thus in* **L***).*

It is worth noting that many of the queries covered by Theorem 27 have an acyclic attack graph as well, and thus even have a consistent first-order rewriting.

10 Conclusion

The main result of this paper is a theorem stating that for every query q in sjfBCQ (i.e., the class of self-join-free Boolean conjunctive queries), CERTAINTY(q) is **coNP**-complete or expressible in *SymStratDatalog*$^{\min}$ (and thus in **L**). Since there exist queries $q \in$ sjfBCQ such that CERTAINTY(q) is **L**-complete, the logspace upper bound in Theorem 2 is tight. The theorem thus culminates a long line of research that started with the ICDT 2005 paper of Fuxman and Miller [13]. The outcome of this research is the following theorem.

▶ **Theorem 28.** *For every self-join-free Boolean conjunctive query q,*

- *if the attack graph of q is acyclic, then* CERTAINTY(q) *is in* **FO***;*
- *if the attack graph of q is cyclic but contains no strong cycle, then* CERTAINTY(q) *is* **L***-complete and expressible in SymStratDatalog*$^{\min}$*; and*
- *if the attack graph of q contains a strong cycle, then* CERTAINTY(q) *is* **coNP***-complete.*

An intriguing open problem is to extend these complexity results to Boolean conjunctive queries with self-joins and to UCQ. Progress in the latter problem may deepen our understanding of relationships between CQA and CSP, which were first discovered in [12].

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. URL: http://webdam.inria.fr/Alice/.

2 Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent Query Answers in Inconsistent Databases. In *ACM PODS*, pages 68–79, 1999. doi:10.1145/303976.303983.

3 Marcelo Arenas, Leopoldo E. Bertossi, Jan Chomicki, Xin He, Vijay Raghavan, and Jeremy P. Spinrad. Scalar aggregation in inconsistent databases. *Theor. Comput. Sci.*, 296(3):405–434, 2003. doi:10.1016/S0304-3975(02)00737-5.

4 Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A Linear-Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas. *Inf. Process. Lett.*, 8(3):121–123, 1979. doi:10.1016/0020-0190(79)90002-4.

5 Franz Baader, Ian Horrocks, Carsten Lutz, and Ulrike Sattler. *An Introduction to Description Logic*. Cambridge University Press, 2017. URL: http://www.cambridge.org/de/academic/subjects/computer-science/knowledge-management-databases-and-data-mining/introduction-description-logic?format=PB#17zVGeWD2TZUeu6s.97.

6 Pablo Barceló and Gaëlle Fontaine. On the data complexity of consistent query answering over graph databases. *J. Comput. Syst. Sci.*, 88:164–194, 2017. doi:10.1016/j.jcss.2017.03.015.

7 Leopoldo E. Bertossi. *Database Repairing and Consistent Query Answering.* Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011. doi:10.2200/S00379ED1V01Y201108DTM020.

8 Meghyn Bienvenu and Camille Bourgaux. Inconsistency-Tolerant Querying of Description Logic Knowledge Bases. In Jeff Z. Pan, Diego Calvanese, Thomas Eiter, Ian Horrocks, Michael Kifer, Fangzhen Lin, and Yuting Zhao, editors, *Reasoning Web: Logical Foundation of Knowledge Graph Construction and Query Answering - 12th International Summer School 2016, Aberdeen, UK, September 5-9, 2016, Tutorial Lectures*, volume 9885 of *Lecture Notes in Computer Science*, pages 156–202. Springer, 2016. doi:10.1007/978-3-319-49493-7_5.

9 Andrei A. Bulatov. Complexity of conservative constraint satisfaction problems. *ACM Trans. Comput. Log.*, 12(4):24:1–24:66, 2011. doi:10.1145/1970398.1970400.

10 László Egri, Benoit Larose, and Pascal Tesson. Symmetric Datalog and Constraint Satisfaction Problems in Logspace. In *LICS*, pages 193–202, 2007. doi:10.1109/LICS.2007.47.

11 László Egri, Benoit Larose, and Pascal Tesson. Directed st-Connectivity Is Not Expressible in Symmetric Datalog. In *ICALP*, pages 172–183, 2008. doi:10.1007/978-3-540-70583-3_15.

12 Gaëlle Fontaine. Why is it Hard to Obtain a Dichotomy for Consistent Query Answering? In *LICS*, pages 550–559, 2013. doi:10.1109/LICS.2013.62.

13 Ariel Fuxman and Renée J. Miller. First-Order Query Rewriting for Inconsistent Databases. In *ICDT*, pages 337–351, 2005. doi:10.1007/978-3-540-30570-5_23.

14 Ariel Fuxman and Renée J. Miller. First-order query rewriting for inconsistent databases. *J. Comput. Syst. Sci.*, 73(4):610–635, 2007. doi:10.1016/j.jcss.2006.10.013.

15 Erich Grädel, Phokion G. Kolaitis, Leonid Libkin, Maarten Marx, Joel Spencer, Moshe Y. Vardi, Yde Venema, and Scott Weinstein. *Finite Model Theory and Its Applications.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2007. doi:10.1007/3-540-68804-8.

16 Sergio Greco, Fabian Pijcke, and Jef Wijsen. Certain Query Answering in Partially Consistent Databases. *PVLDB*, 7(5):353–364, 2014. URL: http://www.vldb.org/pvldb/vol7/p353-greco.pdf, doi:10.14778/2732269.2732272.

17 Martin Grohe and Thomas Schwentick. Locality of order-invariant first-order formulas. *ACM Trans. Comput. Log.*, 1(1):112–130, 2000. doi:10.1145/343369.343386.

18 Phokion G. Kolaitis, Enela Pema, and Wang-Chiew Tan. Efficient Querying of Inconsistent Databases with Binary Integer Programming. *PVLDB*, 6(6):397–408, 2013. URL: http://www.vldb.org/pvldb/vol6/p397-tan.pdf, doi:10.14778/2536336.2536341.

19 Paraschos Koutris and Jef Wijsen. The Data Complexity of Consistent Query Answering for Self-Join-Free Conjunctive Queries Under Primary Key Constraints. In *PODS*, pages 17–29, 2015. doi:10.1145/2745754.2745769.

20 Paraschos Koutris and Jef Wijsen. Consistent Query Answering for Self-Join-Free Conjunctive Queries Under Primary Key Constraints. *ACM Trans. Database Syst.*, 42(2):9:1–9:45, 2017. doi:10.1145/3068334.

21 Paraschos Koutris and Jef Wijsen. Consistent Query Answering for Primary Keys and Conjunctive Queries with Negated Atoms. In *PODS*, pages 209–224, 2018. doi:10.1145/3196959.3196982.

22 Paraschos Koutris and Jef Wijsen. Consistent Query Answering for Primary Keys in Logspace. *CoRR*, abs/1810.03386, 2018. arXiv:1810.03386.

23 Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. Inconsistency-tolerant query answering in ontology-based data access. *J. Web Sem.*, 33:3–29, 2015. doi:10.1016/j.websem.2015.04.002.

24 Leonid Libkin. *Elements of Finite Model Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. doi:10.1007/978-3-662-07003-1.

25 Andrea Lincoln, Virginia Vassilevska Williams, and R. Ryan Williams. Tight Hardness for Shortest Cycles and Paths in Sparse Graphs. In *ACM-SIAM SODA*, pages 1236–1252, 2018. doi:10.1137/1.9781611975031.80.

26 Carsten Lutz and Frank Wolter. On the Relationship between Consistent Query Answering and Constraint Satisfaction Problems. In *ICDT*, pages 363–379, 2015. doi:10.4230/LIPIcs.ICDT.2015.363.

27 Mónica Caniupán Marileo and Leopoldo E. Bertossi. The consistency extractor system: Answer set programs for consistent query answering in databases. *Data Knowl. Eng.*, 69(6):545–572, 2010. doi:10.1016/j.datak.2010.01.005.

28 Dany Maslowski and Jef Wijsen. A dichotomy in the complexity of counting database repairs. *J. Comput. Syst. Sci.*, 79(6):958–983, 2013. doi:10.1016/j.jcss.2013.01.011.

29 Dany Maslowski and Jef Wijsen. Counting Database Repairs that Satisfy Conjunctive Queries with Self-Joins. In *ICDT*, pages 155–164, 2014. doi:10.5441/002/icdt.2014.18.

30 Fabian Pijcke. *Theoretical and Practical Methods for Consistent Query Answering in the Relational Data Model.* PhD thesis, University of Mons, 2018.

31 Piotr Przymus, Aleksandra Boniewicz, Marta Burzanska, and Krzysztof Stencel. Recursive Query Facilities in Relational Databases: A Survey. In *FGIT*, pages 89–99, 2010. doi:10.1007/978-3-642-17622-7_10.

32 Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4):17:1–17:24, 2008. doi:10.1145/1391289.1391291.

33 Jef Wijsen. On the first-order expressibility of computing certain answers to conjunctive queries over uncertain databases. In *PODS*, pages 179–190, 2010. doi:10.1145/1807085.1807111.

34 Jef Wijsen. Certain conjunctive query answering in first-order logic. *ACM Trans. Database Syst.*, 37(2):9:1–9:35, 2012. doi:10.1145/2188349.2188351.

35 Jef Wijsen. A Survey of the Data Complexity of Consistent Query Answering under Key Constraints. In *FoIKS*, pages 62–78, 2014. doi:10.1007/978-3-319-04939-7_2.

Learning Definable Hypotheses on Trees

Emilie Grienenberger
ENS Paris-Saclay, 61 Avenue du Président Wilson, 94230 Cachan, France
emilie.grienenberger@ens-cachan.fr

Martin Ritzert
RWTH Aachen University, Templergraben 55, 52062 Aachen, Germany
ritzert@informatik.rwth-aachen.de

Abstract

We study the problem of learning properties of nodes in tree structures. Those properties are specified by logical formulas, such as formulas from first-order or monadic second-order logic. We think of the tree as a database encoding a large dataset and therefore aim for learning algorithms which depend at most sublinearly on the size of the tree. We present a learning algorithm for quantifier-free formulas where the running time only depends polynomially on the number of training examples, but not on the size of the background structure. By a previous result on strings we know that for general first-order or monadic second-order (MSO) formulas a sublinear running time cannot be achieved. However, we show that by building an index on the tree in a linear time preprocessing phase, we can achieve a learning algorithm for MSO formulas with a logarithmic learning phase.

2012 ACM Subject Classification Theory of computation → Logic

Keywords and phrases monadic second-order logic, trees, query learning

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.24

Funding *Martin Ritzert*: This work is supported by the German research council (DFG) Research Training Group 2236 UnRAVeL.

1 Introduction

In this paper we study the algorithmic complexity of learning properties of nodes in directed labeled trees using a declarative framework introduced by Grohe and Turán [18]. Let T be such a tree with nodes $V(T)$. We call T the *background tree* of our learning problem. The tree T encodes the background knowledge of the learning problem and thus provides the information on which the classification of the nodes $u \in V(T)$ can be based. In our setting a (boolean) *classifier* is a function $H\colon V(T) \to \{+,-\}$ that estimates whether a given node admits a certain property. A learning algorithm gets a *training set* $S \subseteq V(T) \times \{+,-\}$, that is a set of pairs (u,c) of positive and negative *examples*, and the background tree T as input and returns a classifier $H_S\colon V(T) \to \{+,-\}$ as its *hypothesis*. We say that learning was successful if the hypothesis H_S is *consistent* with S which means that for every $(u,c) \in S$ we have that $H_S(u) = c$. Achieving consistency with S can be seen as the extreme case of minimizing the *training error*, i.e. the number of $(u,c) \in S$ such that $H_S(u) \neq c$. Minimizing the training error, also called *empirical risk minimization*, results in provably good generalization behavior in the PAC learning model (see [6]). For those generalization results, we use that logical formulas on trees admit bounded VC-dimension (shown by Grohe and Turán [18]) and that any consistent learner can be turned into a PAC learner by an appropriate training set S (see [6]). We give more details on the connection to PAC learning in Section 2.3.

An example of a simple property a node can admit is having an ancestor with label b. This property can be expressed by the logical formula $\varphi(x) = \exists y\,(y < x) \wedge R_b(y)$. We aim to learn properties which can be defined by logical formulas with parameters based on positive and negative examples over tree-structured data such as web pages, XML databases and JSON files.

▶ **Example 1.** Given a large website such as a news portal. The document object model (DOM) of a website is the tree of elements which form the website. Many websites contain a number of ads and some of them are not trivially detectable. A learning algorithm could then estimate the property of a position in the DOM to be part of some ad.

The output of a learning algorithm would then be a formula that distinguishes nodes belonging to the content of the web page from those belonging to ads. Those formulas could then be used as a basis for new simpler or better filter rules.

In Example 1, the user could select parts of a web page, which he sees as advertisement and then let the learning algorithm produce a classifier which is consistent with his choice. In this paper we will not go into detail of how we get our training set but instead only talk about finding consistent hypotheses for a given training set.

We consider learning algorithms that return classifiers based on logical formulas, especially quantifier-free formulas and formulas from monadic second-order logic. In our logical framework, a classifier consists of a formula $\varphi(x\,;\bar{y})$ and an instantiation $\bar{v}$ of the free variables $\bar{y}$. This formula φ has two types of free variables; we refer to x as the *instance variable* and to $\bar{y} = (y_1, \ldots, y_\ell)$ as *parameter variables*, where $\ell \in \mathbb{N}$. The background structure T is the (fixed) background knowledge that encodes the context of a node which is to be classified. The parameters of φ, which can be seen as constants the formula is allowed to use for the classification, are taken from $V(T)$. In Example 1, a classifier would consist of a logical formula and a number of positions in the DOM of the web page. The formula $\varphi(x\,;\bar{y})$ together with an ℓ-tuple $\bar{v} \in V(T)^\ell$ of parameters then defines a binary classifier $[\![\varphi(x\,;\bar{v})]\!]^T \colon V(T) \to \{+,-\}$ over the tree T as follows. An instance $u \in V(T)$ is classified as positive if $T \vDash \varphi(u,\bar{v})$ such that we have $[\![\varphi(x\,;\bar{v})]\!]^T(u) = +$. Correspondingly we have $[\![\varphi(x\,;\bar{v})]\!]^T(u') = -$ for any $u' \in V(T)$ with $T \nvDash \varphi(u',\bar{v})$. We call such a classifier where φ is an MSO formula an *MSO definable hypothesis*.

We assume the background tree T to be very large, which means large enough that just reading it sequentially takes long, while the logical formula returned by the learning algorithm is assumed to be small for every real-world query. This implies two strategies. First, we use a *data complexity* view for the analysis considering the tree T and the training set S as data and the hypothesis class parameterized by ℓ (and an additional parameter q introduced later) as a constant, such that the complexity results are only given in terms of T and S. Essentially this means that the influence of the formula is considered to be constant. Second, we are interested in finding algorithms which run in sublinear time in the size of T. As such sublinear algorithms are unable to read the whole background tree T, we model the exploration of T using oracles. Those oracles allow the learning algorithm to explore the tree by following edges, starting from the training examples in S. This is formally defined in Section 2.

In general there are no consistent sublinear learning algorithms for first-order and monadic second-order formulas over trees. In [16] the authors have shown that for learning first-order formulas over words, linear time is necessary. The same counterexample can also be used for trees, showing that linear time is again necessary. For structures of bounded degree, there exists a sublinear learning algorithm for first-order formulas (see [17]). This result is not applicable in our setting, as the ancestor relation $\leq$ in the signature of our trees induces unbounded degree (the degree of the root is $|V(T)| - 1$ as it is an ancestor of every other node).

1.1 Our Results

In our formal setting, we consider learnability on trees for monadic second-order logic and the quantifier-free fragment of first-order logic. We show that in contrast to the corresponding case on strings (see [16]), even the relatively simple task of learning quantifier-free formulas on trees needs at least linear time. This is due to the need to synthesize appropriate parameters, a task which can involve searching for the largest common ancestor of two nodes. We show that this is really the core of the problem by giving a sublinear learning algorithm for quantifier-free formulas in Section 3 where we exploit an additional oracle providing access to the largest common ancestor of two nodes.

As we know that there is no sublinear learning algorithm for MSO formulas on trees, we investigate whether the necessary linear computation depends on the training examples. This hardness still holds if the learning algorithm is allowed to return more parameters, as long as the complete training set can not be encoded in those parameters. It turns out that it is possible to build an auxiliary structure in linear time which can then be used for sublinear learning. We present an algorithm which builds such an index structure without knowledge of the training set in linear time and then uses logarithmic time to output a consistent hypothesis. The algorithm builds on the results on strings (see [16]) as well as known techniques for evaluating MSO formulas under updates (see [5]), combining them in a non-trivial way to a learning algorithm for MSO formulas. The linear indexing phase in the learning algorithm cannot be avoided since already for first-order formulas on words it is necessary to invest at least linear time in $\mathcal{O}(|T|)$ to find a consistent hypothesis. The following theorem is the main result of this paper.

▶ **Theorem 2.** *There is a consistent MSO learning algorithm on trees which uses linear indexing time $\mathcal{O}(|T|)$ and logarithmic learning time $\mathcal{O}(|S| \log |T|)$.*

As an application of our indexing algorithm we describe an online learning algorithm that computes an index in $\mathcal{O}(|T|)$ and then, for a sequence of examples, updates its MSO-definable hypothesis in time $\mathcal{O}(\log^2(T))$ per example. That is, in this setting the examples arrive one-by-one and we are able to maintain a consistent hypothesis in polylogarithmic time in the background structure.

1.2 Related work

The field of inductive logic programming (see for example [10, 22, 24, 25, 26]) is very close to our framework. In both cases the aim is to infer logical formulas from positive and negative examples such that the logical formula is consistent with the training examples. The main difference to our setting is that in the ILP framework the background knowledge is also encoded in logic (a so called background theory), while we use a structure to encode background knowledge. In our setting, facts such as gender and age of a person or the issuing institute of a credit card are represented using nodes for person and credit card, as well as unary relations to describe their attributes. Naturally facts which involve multiple entities can be represented by edges. Our framework is able to represent such facts as long as the union of all binary relations still describes a tree or forest while in the ILP setting there is no such restriction. The other important difference is that ILP focuses on first-order logic (and there especially Horn-formulas), while in this paper we work with monadic second-order logic (MSO) which is strictly more expressive than first-order logic. There is a number of other logical frameworks for machine learning, mainly originating from the field of formal verification and databases. Examples are given by [1, 8, 23, 14, 20, 36].

Another related field, which is based on the query by example strategy, is to learn XPATH queries as in [32]. There, unary relations defined by an XPATH expression are learned for arbitrary training sets. The main difference to our setting is that we use MSO formulas, which are in general more expressive than XPATH statements and then restrict the maximal complexity of our formulas.

The field of automata learning and learning of regular languages is also to some degree similar to our setting, especially since we are also applying automata based techniques. There are numerous negative results such as [2, 15, 28, 21, 4]. Of the positive results in that area [3, 30, 27, 13], most of them use an active framework where a teacher iteratively gives counterexamples until the hypothesis is correct. In our framework a consistent hypothesis for a training set is sought and that training set is known from the beginning. Even though our classification problem can be encoded as a learning problem for regular tree languages, their results seem technically unrelated to ours.

2 Preliminaries

In this paper we work with logical formulas from the quantifier-free fragment of first-order logic and formulas from monadic second-order logic. Quantifier-free formulas only consist of boolean combinations of atomic properties, while monadic second-order logic (MSO) extends first-order logic (FO) by quantification over sets of nodes. As an example, take the MSO formula $\exists X \forall z\, X z$ which is always satisfied as there is always a set X containing all elements of the structure. It is known that a set of trees can be recognized by a deterministic bottom-up tree automaton (DTA) $\mathcal{A}$ if and only if it can be characterized by an MSO sentence Φ and both $\mathcal{A}$ and Φ can be computed from each other. For a more detailed description of MSO and tree automata we refer to [33].

In this paper we consider labeled trees as background structures. The most prominent examples for trees in a database context are the tree-structured data exchange formats XML and JSON. Formally a labeled tree $T = (V(T), E_1, E_2, R_1, \dots, R_r, \leq)$ is a structure with vertex set $V(T)$ and binary edge relations E_1 and E_2 encoding the first and second child of a node in a binary tree, or in case of unranked trees, the first child and the next sibling of each node. The unary relations $R_1, \dots R_r$ define the label of each node and for every $a, b \in V(T)$ we have $a \leq b$ if a is an ancestor of b. In this paper we use formulas over the alphabet $\sigma = \{E_1, E_2, \leq, R_1, \dots, R_r\}$ which means that in a formula we can access the tree structure using E_1 and E_2 as well as the labels of each node using $R_1, \dots R_r$. A fragment of an XML document (focusing on persons) could look like the following.

```
<person name="A. Turing" birthday="1912-06-23">
  <interest>computer science</interest>
  <interest>marathon running</interest>
  ...
</person>
```

A formula has access to all tags occurring in the XML document. In the above XML fragment, the labels are given by the unary relations `‘person’,‘name’,‘birthday’,‘interest’`. Adding additional content-based labels such as `‘computer scientist’` or `‘runner’` to the set of unary relations allows to write formulas that depend on the content of nodes. Without such content-based labels a formula could only use structural properties and define sets such as all persons with at least three interests and two friends.

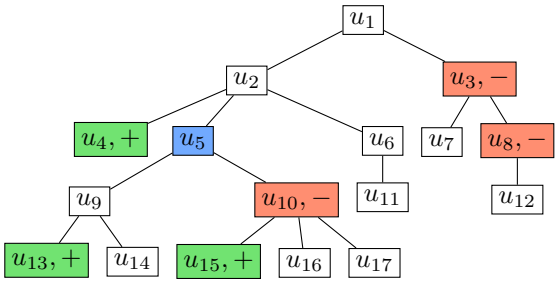


Figure 1 A tree with positive (green) and negative (red) example nodes.

2.1 Learning Model

We consider the model of *supervised learning* where the input to a *learning algorithm* is a *training set* (or training sequence) $S \subseteq V(T) \times \{+, -\}$. Each *example* $(u, c) \in S$ consists of a node u and its classification c. We assume that S is non-contradicting, that is if $(u, c) \in S$, then $(u, \neg c) \notin S$. For the tree given in Figure 1 we define the training set:

$$S = \{(u_3, -), (u_4, +), (u_8, -), (u_{10}, -), (u_{13}, +), (u_{15}, +)\}$$

As already defined in the introduction, a definable hypothesis $[\![\varphi(x\,;\bar{v})]\!]^T$ assigns $+$ to every position $u \in V(T)$ with $T \vDash \varphi(u, \bar{v})$ and $-$ otherwise. Let $\varphi(x\,;y) = \exists z(E(x,z) \wedge E(z,y)) \wedge x \neq y$ accepting all positions with a distance of 2 from the position of y (T contains no self-loops). In the example from Figure 1 we have $[\![\varphi(x\,;u_5)]\!]^T(u) = +$ if and only if $u \in \{u_1, u_4, u_6, u_{13}, u_{14}, u_{15}, u_{16}, u_{17}\}$. This hypothesis is *consistent* with S as it accepts all positive and none of the negative examples from S. The formula $\psi(x) = \exists\, z E_1(z, x)$ defines another consistent hypothesis with $[\![\psi(x)]\!]^T(u) = +$ if and only if $u \in \{u_2, u_4, u_7, u_9, u_{11}, u_{12}, u_{13}, u_{15}\}$.

The quantifier rank $\mathrm{qr}(\varphi)$ of a formula φ is the maximal nesting depth of quantifiers in φ. As every set $S^+ \subseteq V(T)$ is definable by a (long enough) MSO formula φ, we restrict our study to sets which are definable by formulas $\varphi(x\,;y_1,\dots y_\ell)$ with $\mathrm{qr}(\varphi) \leq q$ where q and ℓ are considered to be part of the problem. Restricting q and ℓ reduces the risk of overfitting as such restricted formulas can only memorize a bounded number of positions and thus, on larger training sets, have to exploit more general patterns in the data.

Our framework naturally admits two different learning problems. In *model learning* we assume that there is a consistent classifier $[\![\varphi(x\,;\bar{v})]\!]^T$ with $\bar{v} \in V(T)^\ell$ and $\mathrm{qr}(\varphi) \leq q$, but only q and ℓ are given to the learning algorithm. This reflects the assumption that there is a simple, as expressed by the choice of q and ℓ, but unknown pattern behind the classification of the training examples from S. In *parameter learning* the formula φ of a consistent classifier $[\![\varphi(x\,;\bar{v})]\!]^T$ is fixed. The learning algorithm is not allowed to modify φ and has to find a consistent parameter setting $\bar{v} \in V(T)^\ell$. This variant reflects the case where we have a general idea about how the solution looks like, but are missing the details. Counterintuitively, parameter learning is the harder problem: the restriction to a specific formula φ might impose (unnecessary) restrictions on the parameters. An edifying example is the parameter learning problem on a background structure T with a singleton unary relation R, using the formula $\vartheta(x\,;y) = R(y)$ and a training set $S = \{(u, +)\}$ for an arbitrary $u \in V(T)$. In this example, for any fixed traversal strategy of the learning algorithm on $V(T)$, the single possible parameter can be placed in the position which is evaluated last. For the associated model learning problem, a possible solution would be to return the formula $\psi(x) = \texttt{true}$ without parameters, which defines a hypothesis consistent with S. In the example given

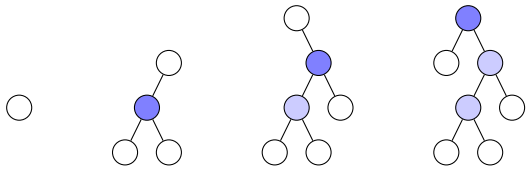


Figure 2 From left to right: Exploration of a tree using neighborhood queries starting from an example node. The dark blue node is the one on which the neighborhood query has been performed last.

in Figure 1, a learning algorithm for the model learning problem would be free to choose between any consistent hypothesis, such as the example formulas $\varphi(x\,;u_5)$ and $\psi(x)$ given above. In the parameter learning problem with the formula $\varphi(x\,;y)$, the (only) consistent output is the assignment $y = u_5$.

In practice, whenever we want to evaluate a definable hypothesis $[\![\varphi(x\,;\bar{v})]\!]^T$, we have to solve an instance of the model checking problem for the formula φ over the structure T. For the case of a fixed MSO formula $\varphi(x\,;\bar{y})$ on a tree T, there is an evaluation strategy in $\mathcal{O}(|T|)$ using tree automata, see for example [33], while a quantifier-free formula ϑ can be evaluated in time $\mathcal{O}(|\vartheta|)$.

2.2 Access Model

A sublinear learning algorithm is unable to read the whole background structure during its computation. We therefore model the access to the background structure by oracles.

Relation Oracles. For a k-ary relation R, the corresponding oracle returns on input of $\bar{u} \in V(T)^k$ whether $\bar{u} \in R$ holds in T.

Neighborhood Oracle. On input of a node $u \in V(T)$, returns the 1-neighborhood of u in T.

In a binary tree, the 1-neighborhood N of a node u consists of u, its parent and its child nodes. In the unranked case $N(u)$ consists of u as well as its left and right sibling, first child and parent. This access model is called *local access* as the tree can only be explored by following edges. Directly jumping to the closest node that is in a relation R or to the last child of an unranked node is not possible in this access model. In practice, those oracles can be implemented using random access on the background tree T. A learning algorithm using local access starts with all nodes occurring in the training set S and then explores the background tree T using the neighborhood and relation oracles. Figure 2 illustrates how the learning algorithm can explore the background tree using subsequent neighborhood queries on the topmost node. The neighborhood queries return the vertices, while the relation queries clarify directions and labels.

2.3 PAC learning

It is known that under certain simplicity restrictions, a consistent learner generalizes well to new and potentially unseen examples. For a *probably approximately correct (PAC)* learning algorithm we have that for every ε and δ there is a size $s \in \mathbb{N}$ of the training set S such that the error of the hypothesis under new examples is bounded by ε with a confidence level of $1-\delta$. This is made formal in Equation (1). Let $c^*\colon V(T) \to \{+,-\}$ be the function that assigns the correct classification to every node $u \in V(T)$. Let the training set $S \in 2^{V(T)\times\{+,-\}}$ be a set of t examples $(u, c^*(u))$ chosen independently and identically distributed (i.i.d.) according to

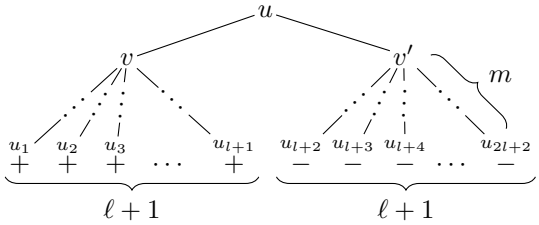


Figure 3 Sketch of the tree T_m parameterized by m and ℓ used in Lemma 4. Substituting the nodes v and v' by balanced binary trees results in the actual tree T_m.

a fixed distribution D. Let $(u, c^*(u)) \sim D$ and $S \sim D$ denote the random choices according to D. Let $H_S \colon V(T) \to \{+, -\}$ be the hypothesis returned by the learning algorithm on input of the training set S. Then the PAC criterion is given by

$$\Pr_{S\sim D}\left(\Pr_{(u,c)\sim D}(H_S(u) \neq c^*(u)) \leq \varepsilon\right) \geq 1 - \delta \tag{1}$$

where the outer probability (the *confidence*) $\Pr_{S\sim D}$ is taken over the training set S for which the learning algorithm produces a hypothesis H_S. The inner probability is the expected error of the hypothesis H_S for an example chosen according to the distribution D. For more details on the PAC learning model, we refer to [34].

There is a very general result from learning theory that shows that for any hypothesis class with bounded Vapnik-Chervonenkis (VC) dimension a consistent learner can be turned into a PAC learning algorithm by providing a large enough training set (see [35] or [6]). For the case of MSO definable hypotheses on trees, the VC dimension is bounded (see [18]). Hence, there is a sufficient size s of S, depending polynomially on $\frac{1}{\varepsilon}$ and $\frac{1}{\delta}$, to satisfy the PAC criterion. Using the algorithms from our Theorems 5 and 6, each providing a consistent learning algorithm for learning formulas on trees, we have the following corollary.

▶ **Corollary 3.** *There are (efficient) PAC learning algorithms for of learning quantifier-free formulas and monadic second-order formulas over trees.*

The running time of those PAC learning algorithms follows directly from the corresponding theorems. A PAC learning algorithm is called *efficient* if the dependence of the running time on the number of training examples is polynomial which is the case in Theorem 5 and 6.

3 Quantifier-free formulas

The class of quantifier-free formulas $\Phi_\sigma[n]$ with n free variables over the signature σ is defined as the fragment of first-order logic without quantifiers. This means that every formula $\varphi(\bar{x}) \in \Phi_\sigma[n]$ can only compare the free variables using a boolean combination of atomic properties from σ. We exploit this limitation to obtain a learning algorithm for quantifier-free formulas which runs in constant time with respect to T under a slightly relaxed notion of local access. We start by showing that there is no sublinear learning algorithm using the notion of local access as defined in Section 2.

▶ **Lemma 4.** *For every $\ell \in \mathbb{N}$, there is no consistent learning algorithm using local access that, given a binary tree T and a training set, returns a consistent hypothesis $[\![\varphi(x\,; \bar{v})]\!]^T$ with $\varphi \in \Phi_\sigma[\ell + 1]$ in time $o(|T|)$.*

Proof sketch. Let L be such a sublinear learning algorithm. Consider the family of trees $(T_m)_{m\in\mathbb{N}}$ shown in Figure 3 with ℓ fixed to the number of parameters used in the formulas returned by L. We define the training set $S = \{(u_1, +), \dots (u_{\ell+1}, +), (u_{\ell+2}, -), \dots, (u_{2\ell+2}, -)\}$ containing $2\ell + 2$ examples. In Figure 3, the positions of the nodes u_i are indicated by their classifications $+$ or $-$. The size of T_m is linear in m for any fixed ℓ and therefore we can choose m such that the running time of L on T_m is smaller than m. This is possible since L runs in time $o(|T_m|)$.

The formula $\varphi(x;y) = y < x$ with parameter v as indicated in Figure 3 is consistent with S. Let $\psi(x\,;\bar{y}) \in \Phi_\sigma[\ell+1]$ be the formula and $v_1, \dots, v_\ell \in V(T_m)$ the parameters returned by L on input T_m and S. Each parameter v_i is an ancestor to exactly one example from S as the algorithm may only return nodes as parameters it has seen during the computation and every leaf is an example node. Every quantifier-free formula can only compare the free variables (x and $\bar{y}$) directly using relations from σ. Therefore, independent of ψ, every parameter has the same effect on every example but (possibly) the one below the parameter. Since there are $\ell + 1$ positive and negative examples, at least one of them will be misclassified which proves the lemma. Intuitively this holds as all paths locally look identical and the algorithm is unable to detect on which side of the tree an example lies. ◀

In the following we extend the notion of local access to cope with the globality of the ancestor relation.

3.1 Extended local access

Our next result states that synthesizing the parameters for quantifier-free formulas is really the core of the linear complexity. *Extended local access* extends local access by a *common ancestor oracle*:

Common Ancestor Oracle. On input of two nodes u and v, it returns the lowest node w such that $w \leq u$ and $w \leq v$ (their lowest common ancestor)

The consistent parameter v for the counterexample from Theorem 4 can easily be found using extended local access. This is generalized in the following theorem showing that for quantifier-free formulas the gap between linear time and sublinear learnability is due to the computation of common ancestors. Let $\ell \in \mathbb{N}$ be an integer.

▶ **Theorem 5.** *There is a learning algorithm that, on input of a tree T and a training set S, outputs a consistent hypothesis $[\![\varphi(x\,;\bar{v})]\!]^T$, where $\varphi \in \Phi_\sigma[1+\ell]$ and $\bar{v} \in V(T)^\ell$ for binary trees and $\varphi \in \Phi_\sigma[1+2\ell]$ ($\bar{v} \in V(T)^{2\ell}$) for unranked trees. This algorithm uses extended local access and runs in time $\mathcal{O}\left(|S|^2 + |S|^\ell |S|\right)$.*

Note that for unranked trees we consider the alphabet $\sigma = \{E_1, E_2, \leq, \preceq\}$. The additional relation $\preceq$ is a partial order, defined as the reflexive transitive closure of E_2. It strictly increases the expressive power of quantifier-free formulas and accounts for need of additional parameters.

Proof sketch. We only sketch the proof for binary trees here. The extension to unranked trees uses the same general idea, but the candidate set for the parameters is slightly different.

The proof uses the concepts of *sufficient sets* and *relative positions*. Given a signature σ. Then v_1 and v_2 share the same relative position if for every binary $R \in \sigma$ and every $x \in S$ we have that $R(x, v_1) \equiv R(x, v_2)$ and $R(v_1, x) \equiv R(v_2, x)$. A *sufficient set* $W \subseteq V(T)$ for a training set S and a set of formulas Φ (such as all quantifier-free formulas) is a set of nodes such that whenever there is a consistent hypothesis $[\![\varphi(x\,;\bar{v})]\!]^T$ with $\varphi \in \Phi$ and

$\bar{v} \in V(T)^\ell$, then there exists another consistent hypothesis $[\![\psi(x\,;\bar{v}')]\!]^T$ with $\psi \in \Phi$ and $\bar{v}' \in W^\ell$. This means that we can restrict ourself to search for consistent parameters from such a sufficient set. For the case of quantifier-free formulas, a set W is sufficient for S if it contains a representative for every possible relative position. Let W be defined by first closing S under lowest common ancestors and then taking the 2-neighborhood of that set. Then W is linear in $|S|$ since we consider binary trees and W is sufficient which can be shown by a simple case-distinction.

The learning algorithm then uses a brute-force process and tests every possible quantifier-free formula for every parameter setting $\bar{v} \in W^\ell$ for consistency. It is bound to find a consistent hypothesis since W is a sufficient set. The set W can be computed in time $\mathcal{O}(|S|^2)$, and each evaluation of a quantifier-free formula uses constant time, resulting in the total runtime of $\mathcal{O}\left(|S|^2 + |S|^\ell |S|\right)$. ◀

4 Monadic Second-Order Logic

In this section we consider learning of unary MSO formulas on trees. The ordering relation $\leq$ used in the previous chapter can be expressed in MSO, thus we do not include it in the signature here. That is we have a binary tree T and want to learn a unary relation $R \subseteq V(T)$ using an MSO formula $\varphi(x\,; y_1, \ldots, y_\ell)$ and ℓ parameters $v_1, \ldots, v_\ell \in V(T)$. We aim for simple concepts and therefore restrict the number of parameters ℓ and the quantifier rank q of φ. Let $\mathrm{MSO}[q, \ell+1]$ be the class of MSO formulas with $\ell+1$ free variables and quantifier rank up to q. Note that the class $\mathrm{MSO}[q, \ell+1]$ is finite up to equivalence for every fixed q and ℓ, such that we can iterate over all formulas from this set to find a consistent one.

We present an algorithm that separates the task of analyzing the background structure and building an *index* from the task of finding a consistent hypothesis based on that index. Formally, we have an indexing phase in which the algorithm has local access to the tree T, but not to the training set S, and produces an index $I(T)$. In the learning phase, the algorithm gets S and has local access to T and $I(T)$. Based on that input, it produces a formula $\varphi(x\,;\bar{y})$ and a parameter configuration $\bar{v}$ such that $[\![\varphi(x\,;\bar{v})]\!]^T$ is consistent with S. We call such an algorithm an *indexing algorithm*, the time needed to build the index *indexing time* and the time for the actual search of a consistent hypothesis based on that index *search time*. Note that such an algorithm is especially useful when performing multiple similar learning tasks on the same data as the index could be reused in that case. We show the following theorem which implies Theorem 2 stated in the introduction.

▶ **Theorem 6.** *Let $q, \ell \in \mathbb{N}$ be fixed. Given a tree T and a training set S that is consistent with an MSO formula $\vartheta(x\,;\bar{y}) \in \mathrm{MSO}[q, \ell+1]$, it is possible to find a consistent formula $\varphi(x\,;\bar{y}) \in \mathrm{MSO}[q, \ell+1]$ that is consistent with S in indexing time $\mathcal{O}(|T|)$ (without access to S) and search time $\mathcal{O}(|S| \log |T|)$.*

Technically, we only consider binary trees, however, the results also extend to unranked trees as there exist MSO interpretations between those classes increasing the quantifier-rank of the resulting formula by 1.

4.1 Decomposing trees into strings

In order to apply techniques based on monoid structures we introduce the notion of path decompositions. Given a tree $T = (V(T), E(T), \leq)$ and a partition $P_1, \ldots, P_p$ of $V(T)$. Let $T[P_i]$ be the induced substructure of P_i on T defined by restricting $V(T), E(T)$ and $\leq$ to

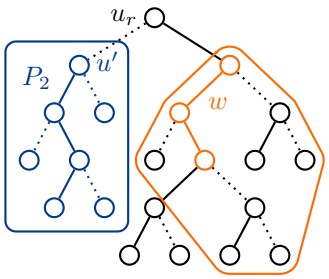


Figure 4 Heavy path decomposition of a tree, solid lines indicate the different heavy paths, the cut-off subtree of u_r is shown in blue and the dependent subtree of w is shown by the orange shape.

the elements from P_i. We consider the case that each induced substructure $T[P_i]$ is a string which implies that $\leq$ restricted to P_i is a (non-reflexive) total order. Let furthermore $T[u]_{\leq}$ be defined as $T[\{w \mid u \leq w\}]$, that is we restrict T to the induced subtree rooted at u.

One special case of decompositions into strings are *heavy path decompositions* introduced by Harel et al. [19]. We say that a node $u \in V(T)$ is at least as *heavy* as $v \in V(T)$ if $|\{w \mid u \leq w\}| \geq |\{w \mid v \leq w\}|$. That means that the heavier element is the one which is the root of the larger induced subtree. Using this definition, we can define the heavy path decomposition as follows.

The *heavy path* of T at $u \in V(T)$ is a path $(u_0, u_1, \ldots, u_\ell)$ such that $u_0 = u$, the node u_ℓ is a leaf and u_i is the leftmost child of u_{i-1} that is at least as heavy as all other children of u_{i-1} for every $i \leq \ell$. Let T be a binary tree and $P = (u_0, \ldots, u_\ell)$ a heavy path. Then the *cut-off subtree* at a node $u_i \in P$ with a child $u' \notin P$ is $T[u']_{\leq}$. The *dependent subtree* of a substring $w \subseteq P$ consists of w and the union of the cut-off subtrees for every $u \in w$. Note that the notions of dependent subtrees and cut-off subtrees refer to binary trees only. The heavy path decomposition $\mathrm{hp}(T) = \{P_1, \ldots, P_p\}$ of T is constructed by first computing P_1 as the heavy path of T at its root and then recursively computing the heavy paths of the cut-off subtrees for each $u \in P_1$. In Figure 4 the heavy path decomposition of a tree is shown. In this figure the blue box includes the cut-off subtree or the root u_r, the node u' is the cut-off child of u_r and the orange shape contains the dependent subtree of the string w. We use the following property of this decomposition in our algorithm.

▶ **Lemma 7** ([5]). *Every path from the root of T to a node $u \in V(T)$ touches at most* $\log |T|$ *different heavy paths of the heavy path decomposition $hp(T)$.*

4.2 DFAs simulating DTAs

We now describe how we can use path decompositions such as $\mathrm{hp}(T)$ and deterministic finite automata (DFAs) to simulate a tree automaton $\mathfrak{A}$ on T as shown in [5].

Let T be a tree, $\mathrm{hp}(T) = \{P_1, \ldots, P_p\}$ its heavy path decomposition and $\mathfrak{A}$ a tree automaton with states Q. Let ρ be the run of $\mathfrak{A}$ on T and $Q' = Q \dot\cup \{q_0\}$. We define a DFA $\mathcal{A}$ and a function $f_\rho \colon V(T) \to \Sigma \times Q' \times \{\mathrm{L}, \mathrm{R}\}$ extending the labels of $V(T)$ by annotations about the state and direction of the cut-off subtree. Those extended labels from f_ρ allow us to run a DFA $\mathcal{A}$ on $\mathrm{hp}(T)$ which simulates the tree automaton $\mathfrak{A}$ on T.

▶ **Lemma 8.** *Let ρ be the run of the DTA $\mathfrak{A}$ on T and T' the tree we get from applying f_ρ to every node of T. Then we have for every $u \in P_i$ that $\mathcal{A}$ is in state q after reading $T'[P_i]$ up to position u, if and only if $\rho(u) = q$.*

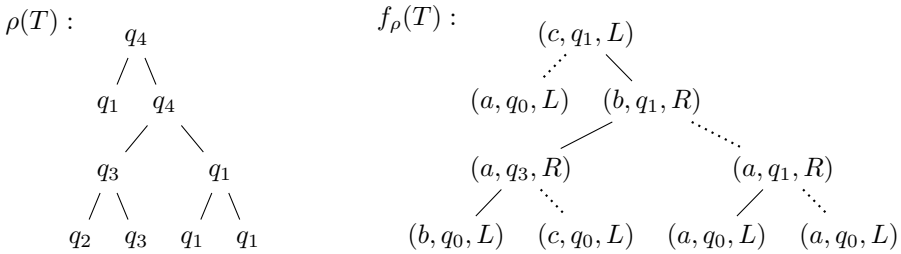


Figure 5 Applying the label transformation f_ρ to $\mathrm{hp}(T)$ for some tree T.

To match the evaluation order of bottom-up tree automata, the constructed DFA reads the paths $T'[P_i]$ in reversed order, that is starting from the leaves and running towards the root of T. The function f_ρ makes use of the run ρ of $\mathfrak{A}$ on T and is defined as follows. For a leaf u with label a in T we have $f_\rho(u) = (a, q_0, \mathrm{L})$. For an inner node $u \in P_i$ with label a and children $u' \in P_i$ and $u'' \notin P_i$ where u'' is a right child of u we have $f_\rho(u) = (a, \rho(u''), \mathrm{R})$ and correspondingly $(a, \rho(u''), \mathrm{L})$ if u'' was a left child of u. Intuitively f_ρ stores the state of the DTA $\mathfrak{A}$ on the cut-off subtree at u in the label of u. Figure 5 shows an example for f_ρ given a tree T and a run ρ of a DTA on T.

Let $\mathfrak{A} = (Q, \Sigma, \delta, F)$ be a deterministic bottom-up tree automaton and $\mathrm{hp}(T) = \{P_1, \dots, P_p\}$ the heavy path decomposition of T. We assume that the transition relation δ of $\mathfrak{A}$ is split into $\delta_0 \colon \Sigma \to Q$ for leaf nodes and $\delta_2 \colon \Sigma \times Q \times Q \to Q$ for inner nodes. We now define the DFA $\mathcal{A} = (Q', \Sigma \times Q' \times \{\mathrm{L}, \mathrm{R}\}, \delta', q_0, F)$ where $Q' = Q \dot\cup \{q_0\}$ with Q, Σ and F taken from $\mathfrak{A}$. The transition relation δ' of the DFA $\mathcal{A}$ is defined as follows:

$$\begin{aligned}
\delta'(q_0, (a, q_0, d)) &= \delta_0(a) && \text{for all } a \in \Sigma, d \in \{\mathrm{L}, \mathrm{R}\} \\
\delta'(p, (a, q, \mathrm{L})) &= \delta_2(a, q, p) && \text{for all } a \in \Sigma,\ p, q \in Q \\
\delta'(p, (a, q, \mathrm{R})) &= \delta_2(a, p, q) && \text{for all } a \in \Sigma,\ p, q \in Q
\end{aligned}$$

Lemma 8 holds as at every position $u \in P_i$ the DFA $\mathcal{A}$ has access to the same information as $\mathfrak{A}$ through the extended alphabet.

Let $<_{\mathrm{hp}}$ be the partial order showing dependence between heavy paths of $\mathrm{hp}(T)$. That is, we have $P_j <_{\mathrm{hp}} P_i$ if P_j is part of the dependent subtree of P_i. For every node $u \in P_i$ with cut-off child $u' \in P_j$, the state $\rho(u')$ only depends on P_j with $P_j <_{\mathrm{hp}} P_i$ such that we can recursively compute ρ using $\mathcal{A}$ on $\mathrm{hp}(T)$ together with Lemma 8. We say that $\mathcal{A}$ accepts a tree T if it accepts the string $T[P_1]$ where $P_1 \in \mathrm{hp}(T)$ contains the root of T.

4.3 Monoids and factorization trees

A monoid $\mathcal{M} = \{M, \cdot_M, 1_M\}$ consists of a set of elements M and an associative multiplication operator $\cdot_M$ with neutral element 1_M. We usually refer to the monoid $\mathcal{M}$ by its set of elements M and write $m_1 m_2$ or $m_1 \cdot m_2$ for $m_1 \cdot_M m_2$. A monoid morphism $h \colon M \to M'$ is a function that translates a monoid in a consistent way, meaning that we have $h(m_1 \cdot m_2) = h(m_1) \cdot h(m_2)$ and $h(1_M) = 1_{M'}$.

The free monoid $M_{\mathrm{free}} = \{\Sigma^*, \cdot_{\mathrm{free}}, \varepsilon\}$ consists of all finite strings over Σ with concatenation as multiplication and the empty word ε as neutral element. A language $\mathcal{L}$ over Σ is *finitely monoid recognizable* if there is a finite monoid M, a monoid morphism $h \colon M_{\mathrm{free}} \to M$ and a subset $F \subseteq M$ such that $w \in \mathcal{L}$ if and only if $h(w) \in F$. A language $\mathcal{L}$ is finitely monoid recognizable, if and only if it is regular [29]. The *transition monoid* M of a DFA $\mathcal{A}$ with

state space Q is defined as functions $m\colon Q \to Q$ with composition as multiplication, and the identity as neutral element. The corresponding monoid morphism h_M maps a string $w \in \Sigma^*$ to the function $m\colon Q \to Q$ modelling the effect of reading w on the states of $\mathcal{A}$. A monoid element m is *productive* if there are m', m'' with $m' \cdot m \cdot m'' \in F$. For further details on the equivalence between finite monoids and DFAs see for example [11, 7].

Every path $P_i \in \mathrm{hp}(T)$ induces a string $T[P_i]$ over Σ which we interpret as an element of the free monoid over the tree alphabet Σ. In the following we will not explicitly distinguish between the path P_i, the corresponding string $T[P_i]$ and the monoid element.

A *factorization tree* of a sequence of monoid elements $s = m_1, m_2, \ldots, m_n$ from a monoid M is a tree F_s where each node $u \in V(F_s)$ is labeled by an element m_u from M and the sequence of leaf labels is s. For each inner node $u \in V(F_s)$ labeled by m_u with children $u_1, \ldots, u_i$ which are labeled by $m_{u_1}, \ldots, m_{u_i}$ we have $m_u = m_{u_1} m_{u_2} \cdots m_{u_i}$. By choosing a balanced binary tree for F_s we get a factorization tree of height $\lceil \log |s| \rceil$. An alternative are *Simon factorization trees*, where each node u is either a leaf, a binary node or satisfies the property that all child nodes $u_1, \ldots, u_n$ of u share the same idempotent label m_u with $m_u = m_u \cdot m_u$.

▶ **Theorem 9** (Simon [31]). *For every sequence $s = [m_1, m_2, \ldots, m_n]$ of monoid elements from M, there is a Simon factorization tree of height at most $3|M|$. This factorization tree can be computed in time $n \cdot poly(|M|)$.*

In a set of *updates* $U = \{(i_1, m_1), \ldots, (i_s, m_s)\}$ with $i_j \in \mathbb{N}$ and $m_j \in M$ each tuple consists of an index and the new monoid element for this index. Applying U to a sequence of monoid elements s results a sequence s^U with $s^U[i_j] = m_j$ for $j \in \{1, \ldots, s\}$ and $s^U[j] = s[j]$ for all $j \notin \{i_1, \ldots, i_s\}$ where $s[i]$ denotes the ith element of the sequence s. We use the following lemma from [16] to apply such updates in Simon factorization trees.

▶ **Lemma 10** ([16]). *Given a Simon factorization tree F of height h over a sequence s of elements from M and a set U of updates. There is an algorithm returning a Simon factorization tree of height $2h + 3|M|$ for the updated sequence s^U in time $\mathcal{O}(|U|h + |U||M|)$.*

4.4 Constructing the necessary monoids and monoid morphisms

In the following we construct and analyze the monoid structure used in the learning algorithm in Section 4.5. Let let T be a tree over Σ and T_1 the same tree over $\Sigma_1 = \Sigma \times 2^{\{y_1, \ldots, y_\ell\}} \times \{?, N, P\}$ including information about the free variables of a formula $\psi(P, N, \bar{y})$ in the labels. Let $\Sigma_2 = \Sigma_1 \times Q' \times \{\mathrm{L}, \mathrm{R}\}$ be the alphabet from Section 4.2 which extends Σ_1 in order to work with string automata instead of tree automata. Let $f_\rho : V(T) \to \Sigma_2$ be the labeling function used in Lemma 8 and T_2 the tree that results from T by relabeling its nodes using f_ρ. Note that the concrete states Q', and therefore the whole construction, depend on the formula ψ that performs the consistency check.

▶ **Lemma 11.** *Let $\varphi(x\,;\bar{y})$ be an MSO formula and $\mathfrak{A}$ a tree automaton for φ with states Q. Let T be a tree over Σ with $hp(T) = \{P_1, \ldots, P_n\}$ and S a training set. Then there is*

- *an alphabet Σ_3,*
- *a construction transforming T to a tree T_3 over Σ_3 depending on $\mathfrak{A}$*
- *a monoid $\hat{\mathcal{M}}$ with a set $F \subseteq \hat{\mathcal{M}}$ of final elements*
- *and a monoid morphism $\hat{h} : \Sigma_3^* \to \hat{\mathcal{M}}$*

such that: $\hat{h}(T_3[P_1]) \in F$ if and only if there exists $\bar{v} \in V(T)^\ell$ such that $[\![\varphi(x\,;\bar{v})]\!]^T$ is consistent with S.

Proof. We start by defining the formula $\psi(P, N, \bar{y})$ which checks for a given training set S, split into positive (P) and negative examples (N), whether for some parameter vector $\bar{v} \in V(T)^\ell$ the hypothesis $[\![\varphi(x\,;\bar{v})]\!]^T$ is consistent with S:

$$\psi(P, N, \bar{y}) = \forall x(P(x) \to \varphi(x, \bar{y})) \wedge (N(x) \to \neg\varphi(x, \bar{y})).$$

The computation of the final monoid $\hat{\mathcal{M}}$ uses the following intermediate constructions.

1. DTA $\mathfrak{A}$ on $\Sigma_1 = \Sigma \times 2^{\{y_1,\dots,y_\ell\}} \times \{?, N, P\}$
2. DFA $\mathcal{A}$ on $\Sigma_2 = \Sigma \times 2^{\{y_1,\dots,y_\ell\}} \times \{?, N, P\} \times Q \times \{\mathrm{L}, \mathrm{R}\}$
3. Transition monoid $\mathcal{M}$ for $\mathcal{A}$ with final elements $F \subseteq \mathcal{M}$ and monoid morphism $h\colon \Sigma_2 \to \mathcal{M}$
4. Monoid morphism $h'\colon \Sigma_2' \to \mathcal{M}$ with $\Sigma_2' = \Sigma \times 2^{\{y_1,\dots,y_\ell\}} \times \{?, N, P\} \times \mathcal{M} \times \{L, R\}$
5. Monoid $\hat{\mathcal{M}} = 2^{\mathcal{M}}$ with elements $\hat{m} \subseteq \mathcal{M}$ for every $\hat{m} \in \hat{\mathcal{M}}$ and monoid morphism $\hat{h}\colon \Sigma_3 = \Sigma \times \{?, N, P\} \times \hat{\mathcal{M}} \times \{\mathrm{L}, \mathrm{R}\} \to \hat{M}$

The first step from ψ to the tree automaton $\mathfrak{A}$ is a standard construction (see e.g. [9]) that involves extending the tree alphabet Σ by unary relations for the free variables of ψ resulting in the alphabet $\Sigma_1 = \Sigma \times \{P, N, ?\} \times 2^{\{y_1,\dots,y_\ell\}}$. The second step in the construction consists of a translation from a DTA with states Q to a DFA with states $Q' = Q \dot\cup \{q_0\}$ which uses the construction from Lemma 8. The constructed DFA $\mathcal{A}$ is able to simulate a run of $\mathfrak{A}$ on the tree T_2 over the alphabet $\Sigma_2 = \Sigma_1 \times Q' \times \{\mathrm{L}, \mathrm{R}\}$ where $T_2 = T$ but the labels are given by f_ρ from Lemma 8. The monoid $\mathcal{M}$ computed in the third step is the transition monoid of $\mathcal{A}$ (for the construction see e.g. [33]), $h\colon \Sigma_2 \to \mathcal{M}$ is the corresponding monoid morphism and $F \subseteq \mathcal{M}$ is the set of accepting monoid elements. In the fourth step we construct the monoid morphism h' that works on $\Sigma_2' = \Sigma \times 2^{\{y_1,\dots,y_\ell\}} \times \{?, N, P\} \times \mathcal{M} \times \{\mathrm{L}, \mathrm{R}\}$; that is, we substitute the component containing for every node u the state q_u of $\mathcal{A}$ on the cut-off subtree by a monoid element $m_u \in \mathcal{M}$. Essentially this step works as we can extract the state q_u on the cut-off subtree at u from the monoid element m_u and then call the morphism h. With $\mathcal{M}$ and h' we can check whether the hypothesis $[\![\varphi(x\,;\bar{v})]\!]^T$ for some $\bar{v} \in V(T)^\ell$ is consistent with the training set S.

In the fifth and last step we construct the monoid $\hat{\mathcal{M}}$ and the monoid morphism $\hat{h}$ based on $\mathcal{M}$ and h' such that $\hat{h}$ can be used to check whether there is a consistent set $\bar{v} \in V(T)^\ell$ of parameters. We then use $\hat{h}$ and $\hat{\mathcal{M}}$ in the actual learning algorithm. Let $\hat{\mathcal{M}} = (2^{\mathcal{M}}, \cdot_{\hat{\mathcal{M}}}, \{1_{\mathcal{M}}\})$ be a structure with multiplication $\hat{m}_1 \cdot_{\hat{\mathcal{M}}} \hat{m}_2 = \{m_1 \cdot_{\mathcal{M}} m_2 \mid m_1 \in \hat{m}_1, m_2 \in \hat{m}_2\}$. $\hat{\mathcal{M}}$ is a monoid as it is closed under multiplication and $\{1_{\mathcal{M}}\} \in \hat{\mathcal{M}}$ is neutral for $\cdot_{\hat{\mathcal{M}}}$ because $1_{\mathcal{M}} \in \mathcal{M}$ is the neutral element of $\mathcal{M}$. Let $\Sigma_3 := \Sigma \times \{P, N, ?\} \times 2^{\mathcal{M}} \times \{L, R\}$. The final monoid morphism $\hat{h}\colon \Sigma_3^* \to \hat{\mathcal{M}}$ is given using the morphism h'. For $(a, \hat{m}, d) \in \Sigma_3$ with $a \in \Sigma \times \{P, N, ?\}$ and $d \in \{\mathrm{L}, \mathrm{R}\}$ we let

$$\hat{h}((a, \hat{m}, d)) = \left\{ h'((a, \bar{y}, m, d) \;\middle|\; \bar{y} \in 2^{\{y_1,\dots,y_\ell\}}, m \in \hat{m} \right\}.$$

For a given position $u \in V(T)$ labeled $(a, \hat{m}, d) \in \Sigma_3$, the definition of $\hat{h}$ can be read as calling h' for every possible distribution of parameters in the cut-off subtree indicated by $m \in \hat{m}$ and every choice of parameters $\bar{y}$ for that position. For a word $w \in \Sigma_3^*$ we split the word into its positions $w_1, \dots, w_n$ and compute the product $\hat{m} = \prod_{i=1}^n \hat{h}(w_i)$ of the monoid elements of those. This is possible since $\hat{h}$ is a monoid morphism from the free monoid to $\hat{\mathcal{M}}$. Note that the interpretation with the cut-off subtrees only makes sense for strings $w = T_3[P_i]$ defined by heavy paths or substrings of such a w.

Let w be the string from the heavy path P_1 containing the root of T. Since $\hat{h}$ tests all possible distributions of parameters in the dependent subtree, we know that if $\hat{h}(w) \cap F \neq \emptyset$ then there is a consistent parameter setting. Correspondingly there is no consistent parameter setting if $\hat{m}_r \cap F = \emptyset$. ◀

We now categorize the elements from $\mathcal{M}$ constructed in the third step of the proof of Lemma 11. Since $\mathcal{A}$ only accepts trees in which every parameter, indicated by a unary relation, is assigned exactly once we get that every productive monoid element $m \in \mathcal{M}$ contains the information which parameters have been read. This holds as otherwise there was an accepted tree where a single parameter has been assigned multiple times or not at all. Thus we can assign a set of parameters to every productive monoid element. We now formalize this idea.

▶ **Lemma 12.** *There is a function $p : \mathcal{M} \to 2^{\{y_1,\dots,y_\ell\}} \cup \{\bot\}$ that assigns a set $\bar{y} \in 2^{\{y_1,\dots,y_\ell\}}$ of parameters to each productive monoid element from $\mathcal{M}$ in a consistent way. That is, for a tree T and a substring w of a path in $hp(T)$ with $h'(w) = m$, exactly the parameters $p(m)$ occur in the dependent subtree of w.*

4.5 Algorithms

Using the monoids $\mathcal{M}$ and $\hat{\mathcal{M}}$ as well as the monoid morphism $\hat{h}$ from Lemma 11, we can compute a consistent parameter setting $\bar{v}$ for a given formula $\varphi(x\,;\bar{y})$ and a training set S. This proves Theorem 6. The presented algorithm is split in three parts, where the first part is done independent of S in the indexing phase of the algorithm. The remaining two parts, updating according to S and tracing the parameters, together make up the search phase of the indexing algorithm. The indexing part is linear in $|T|$, while the search phase takes time $\mathcal{O}((|S| + \ell) \log |T|)$ where the summands are for the updating and the tracing algorithm respectively. We assume that the underlying formula φ is fixed and therefore ignore the factors depending only on φ but have argued that those can be non-elementary due to exploding statespaces of the constructed automata.

The indexing algorithm

The indexing algorithm starts by computing the monoid $\hat{\mathcal{M}}$ from φ as described in Lemma 11 as well as the heavy path decomposition $\mathrm{hp}(T)$. It outputs the set of Simon factorization trees over $\hat{\mathcal{M}}$ for each P_i from $\mathrm{hp}(T)$ computed by the algorithm from [31]. Technically, the order of the computation of the factorization trees has to be consistent with the dependence relation $<_{\mathrm{hp}}$ as a label $a_u \in \Sigma_3$ of a node $u \in V(T)$ contains the monoid element $\hat{m} \in \hat{\mathcal{M}}$ of the cut-off subtree at u. As the computation of $\hat{\mathcal{M}}$ only depends on φ, the overall computation is dominated by the computation of the factorization trees in $|T| \cdot \mathrm{poly}(|\hat{\mathcal{M}}|)$.

The update algorithm

In the update part of the learning algorithm we add the information from S to the set of Simon factorization trees computed in the indexing part of the algorithm. The updates are performed bottom-up, that is we change the labels in the leafs each factorization tree belonging to positions from S and then propagate this information towards the root of T. The presented algorithm works in two stages: an outer stage that collects all updates for each heavy path and an inner stage that actually performs the update.

The outer stage orchestrates the update process by computing the set of updates U_{P_i} for every $P_i \in \mathrm{hp}(T)$. The set U_{P_i} consists of updates for every $u \in P_i$ occurring in S as well as (possible) updates from paths P_j with $P_j <_{\mathrm{hp}} P_i$ due to previous updates. Updates originating from S change the component $\{P, N, ?\}$ of Σ_3 while updates induced from changes in the cut-off subtree of $u \in P_i$ change the monoid element $\hat{m}$ in the label of u. By updating the factorization trees in an order consistent with $<_{\mathrm{hp}}$ every heavy path needs up be updated at

most once. The inner algorithm which actually performs the update is taken from Lemma 10. As every path from a node to the root touches at most logarithmically many heavy paths as stated in Lemma 7, the total runtime of the update part is in $\mathcal{O}(|S| \log |T|)$.

The tracing algorithm

The tracing algorithm gets the updated set of factorization trees $\mathcal{F} = F_{P_1}, \ldots, F_{P_n}$ and computes a set of parameters $\bar{v}$ such that $[\![\varphi(x\,;\bar{y})]\!]^T$ is consistent with S. Let P_1 contain the root of T. The algorithm works in a top down manner, that is it starts in the root of F_{P_1} and traces the parameters downwards using Lemma 12. We again divide the algorithm into an inner and an outer algorithm where the inner algorithm traces the parameters similar to [16] in a single factorization tree and the outer algorithm orchestrates the search within $\mathcal{F}$.

Let $\hat{m}_r \in \hat{\mathcal{M}}$ be the monoid element reached in the root of F_{P_1}. The tracing starts by choosing the (initial) local target $m_r \in \hat{m}_r \cap F$ for the root of F_{P_1}. Within each factorization tree F_{P_i} at a position u with local target m_u we select monoid elements m_1, m_2 from the children u_1, u_2 of u such that $m_1 m_2 = m_u$ using a brute-force test. The tracing continues for those u_i where $p(m_i) \neq \emptyset$ with m_i as local target at u_i until the leaves of F_{P_i} are reached and outputs pairs $(u, m_u) \in V(T) \times \mathcal{M}$ with $p(m_u) \neq \emptyset$. Then the outer part of the algorithm decides which of the parameters $p(m_u)$ are placed in P_i and which are further traced in cut-off subtrees with $P_j <_{\text{hp}} P_i$. The algorithm then selects a target monoid element $m \in \mathcal{M}$ for the root of F_{P_j} and calls the inner tracing algorithm until every parameter is placed in some heavy path (which happens eventually as T is finite). The choice of the target monoid element m for the root of F_{P_j} is done by testing at the node $u \in P_i$ every possibility of reachable monoid m in the cut-off subtree while fixing the remaining parameters $K = p(m_u) \setminus p(m)$ at u. This check is possible since the label of u contains a monoid element $\hat{m}_u$ that consists of all reachable monoid elements in the cut-off subtree of u.

The parameter configuration $\bar{v}$ found this way is consistent with S as it is computed in a way that $h'(T_2[P_1]) = m_r \in F$ which is accepted. For each parameter y_i, the inner algorithm uses constant time within each factorization tree and is called at most $\log |S|$ times by Lemma 7 stating that every path from a node u to the root u_r touches at most $\log |S|$ heavy paths. Together with the indexing and update algorithm this is means that we can find a consistent parameter setting $\bar{v}$ for φ in indexing time $\mathcal{O}(|T|)$ and search time $\mathcal{O}(\log |T| \cdot |S|)$.

5 Online learning of MSO formulas

For Theorem 6 we assumed that after an indexing phase the complete training set S is known and the learning task is to find a hypothesis consistent with S. We now lift this result to an online setting where we again have a linear indexing phase and then on input of new batch of examples S_i the algorithm updates its hypothesis H such that H_i is consistent with all examples $S = \bigcup_i S_i$ it has seen so far. This online setting allows examples to arrive over time which is natural for many tasks with human interaction. Note that the algorithm can also handle label updates of the nodes as both types of updates induce a label change in the tree T. Label changes are a common type of update in a database setting since updating the attributes of an already present entity can be modeled by an update of the entity's label.

An *online learning algorithm* is an algorithm that takes as input a background structure T, an index $I(T)$ and a sequence $S = (u_1, c_1), (u_2, c_2), \ldots$ of training examples and outputs for every $i \leq |S|$ a hypothesis $H_i = [\![\varphi(x\,;\bar{v})]\!]^T$ consistent with $S_i = \{(u_1, c_1), \ldots, (x_i, c_i)\}$.

▶ **Theorem 13.** *Let $q, \ell \in \mathbb{N}$. There is an indexing algorithm A that, given a tree T computes an index $A(T)$ and an online learning algorithm B that, given $T, I(T)$, and an $MSO[q, \ell+1]$-realizable sequence $S = (u_1, c_1), \dots$ for T, maintains a consistent hypothesis $H_i = [\![\varphi(x\,; \bar{v})]\!]^T$ for every $i \in \mathbb{N}$ such that A runs in time $\mathcal{O}(|T|)$ and B runs in time $\mathcal{O}(\log^2(|T|))$ per update.*

This can be achieved by substituting Simon factorization trees by the conceptually simpler binary factorization trees in the construction. This implies two main differences. First, we can update labels arbitrarily often without changing the structure of the factorization tree (which does not hold for Simon factorization trees due to the idempotent elements). Second, the height of each factorization tree is logarithmic in its length, i.e. at most logarithmic in T. Therefore it takes logarithmic time to update each path and since a single update may involve updating logarithmically many paths this results in a runtime of $\mathcal{O}(|S_i| \log^2(|T|))$ per update.

6 Conclusion

We considered the setting of learning quantifier-free and MSO formulas on trees. All learning algorithms provided in this paper search for consistent hypotheses, thus they can be turned into PAC learning algorithms by providing a large enough training set.

We assumed the background structures to be huge and therefore have been researching sublinear algorithms which access to the background structures through the local access oracles. The first result is that even for quantifier free formulas there is no sublinear learning algorithm. However, there is a sublinear learning algorithm when given access to the largest common ancestor of two nodes. Our main result is a learning algorithm for unary MSO formulas which uses a linear indexing phase to build up an auxiliary structure (the index) and admits a logarithmic learning time with local access to that index.

Further research questions might include lifting the result to higher dimensions where examples consist of pairs or tuples of nodes instead of single positions in the tree. Another direction of research could be to extend our results for tree-like structures. For structures of bounded tree-width the approach could use a similar structure as the one from [12]. A slightly different research question would be to look for approximate solutions where only a certain (relative) amount of examples needs to be consistent. Such approaches could also deal with faulty examples, which occur quite regularly in practice.

References

1 A. Abouzied, D. Angluin, C.H. Papadimitriou, J.M. Hellerstein, and A. Silberschatz. Learning and verifying quantified boolean queries by example. In R. Hull and W. Fan, editors, *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 49–60, 2013.

2 D. Angluin. On the Complexity of Minimum Inference of Regular Sets. *Information and Control*, 39(3):337–350, 1978.

3 D. Angluin. Learning Regular Sets from Queries and Counterexamples. *Information and Computation*, 75(2):87–106, 1987.

4 D. Angluin. Negative Results for Equivalence Queries. *Machine Learning*, 5:121–150, 1990.

5 A. Balmin, Y. Papakonstantinou, and V. Vianu. Incremental validation of XML documents. *ACM Trans. Database Syst.*, 29(4):710–751, 2004.

6 A. Blumer, A. Ehrenfeucht, D. Haussler, and M.K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM (JACM)*, 36:929–965, 1989.

7 M. Bojańczyk. Algorithms for regular languages that use algebra. *SIGMOD Record*, 41(2):5–14, 2012.

8 A. Bonifati, R. Ciucanu, and S. Staworko. Learning Join Queries from User Examples. *ACM Trans. Database Syst.*, 40(4):24:1–24:38, 2016.

9 J Richard Büchi. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6(1-6):66–92, 1960.

10 W.W. Cohen and C.D. Page. Polynomial Learnability and Inductive Logic Programming: Methods and Results. *New generation Computing*, 13:369–404, 1995.

11 T. Colcombet. Green's Relations and Their Use in Automata Theory. In *Language and Automata Theory and Applications - 5th International Conference, LATA 2011, Tarragona, Spain, May 26-31, 2011. Proceedings*, volume 6638 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 2011.

12 B. Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and computation*, 85(1):12–75, 1990.

13 F. Drewes and J. Högberg. Learning a regular tree language from a teacher. In *Developments in Language Theory*, pages 279–291. Springer, 2003.

14 P. Garg, D. Neider, P. Madhusudan, and D. Roth. Learning invariants using decision trees and implication counterexamples. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 499–512, 2016.

15 E.M. Gold. Complexity of Automaton Identification from Given Data. *Information and Control*, 37(3):302–320, 1978.

16 M. Grohe, C. Löding, and M. Ritzert. Learning MSO-definable hypotheses on strings. In *International Conference on Algorithmic Learning Theory, ALT 2017, 15-17 October 2017, Kyoto University, Kyoto, Japan*, pages 434–451, 2017.

17 M. Grohe and M. Ritzert. Learning first-order definable concepts over structures of small degree. In *Proceedings of the 32nd ACM-IEEE Symposium on Logic in Computer Science*, 2017.

18 M. Grohe and G. Turán. Learnability and definability in trees and similar structures. *Theory of Computing Systems*, 37(1):193–220, 2004.

19 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *siam Journal on Computing*, 13(2):338–355, 1984.

20 C. Jordan and L. Kaiser. Machine Learning with Guarantees using Descriptive Complexity and SMT Solvers. *ArXiv (CoRR)*, 1609.02664 [cs.LG], 2016. `arXiv:1609.02664`.

21 M.J. Kearns and L.G. Valiant. Cryptographic Limitations on Learning Boolean Formulae and Finite Automata. *Journal of the ACM*, 41(1):67–95, 1994.

22 J.-U. Kietz and S. Dzeroski. Inductive Logic Programming and Learnability. *SIGART Bulletin*, 5(1):22–32, 1994.

23 C. Löding, P. Madhusudan, and D. Neider. Abstract Learning Frameworks for Synthesis. In M. Chechik and J.-F. Raskin, editors, *Proceedings of the 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 9636 of *Lecture Notes in Computer Science*, pages 167–185. Springer Verlag, 2016.

24 S. Muggleton. Inductive logic programming. *New Generation Computing*, 8(4):295–318, 1991.

25 S.H. Muggleton, editor. *Inductive Logic Programming*. Academic Press, 1992.

26 S.H. Muggleton and L. De Raedt. Inductive Logic Programming: Theory and methods. *The Journal of Logic Programming*, 19-20:629–679, 1994.

27 J. Oncina and P. García. Identifying regular languages in polynomial time. In *Proceedings of the International Workshop on Structural and Syntactic Pattern Recognition*, volume 5 of *Machine Perception and Artificial Intelligence*, pages 99—-108. World Scientific, 1992.

28 L. Pitt and M.K. Warmuth. The Minimum Consistent DFA Problem Cannot be Approximated within any Polynomial. *Journal of the ACM*, 40(1):95–142, 1993.

29 M.O. Rabin and D.Scott. Finite Automata and Their Decision Problems. *IBM Journal of Research and Development*, 3:114–125, 1959.

30 R.L. Rivest and R.E. Schapire. Inference of Finite Automata Using Homing Sequences. In *Machine Learning: From Theory to Applications*, volume 661 of *Lecture Notes in Computer Science*, pages 51–73. Springer, 1993.

31 I. Simon. Factorization Forests of Finite Height. *Theoretical Computer Science*, 72(1):65–94, 1990.

32 Sławek Staworko and Piotr Wieczorek. Learning twig and path queries. In *Proceedings of the 15th International Conference on Database Theory*, pages 140–154. ACM, 2012.

33 W. Thomas. Languages, Automata, and Logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 389–456. Springer-Verlag, 1997.

34 L.G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

35 V. Vapnik and A. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16:264–280, 1971.

36 Y. Weiss and S. Cohen. Reverse Engineering SPJ-Queries from Examples. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 151–166. ACM, 2017.