Existential Active Integrity Constraints

**Marco Calautti, Luciano Caroprese, Sergio Greco,
Cristian Molinaro[1], Irina Trubitsyna, Ester Zumpano**

DIMES Department, Università della Calabria, Italy
{lastname}@dimes.unical.it

Abstract

Active integrity constraints (AICs) are a useful formalism to express integrity constraints and policies to restore consistency in databases violating them. However, AICs do not allow users to express different kinds of constraints commonly arising in practice, such as foreign keys.

In this paper, we propose *existential active integrity constraints* (EAICs), a powerful extension of AICs that allows us to express a wide range of constraints used in databases and ontological systems. We investigate different properties of EAICs. Specifically, we show that there exists a "representative" set of founded updates, called *universal*, which suffices for query answering. As such a set might contain an infinite number of founded updates, each of infinite size, we study syntactic restrictions ensuring finiteness, as well as the existence of a single universal founded update.

Keywords: Knowledge representation and management, Knowledge bases, Repair, Certain query answering.

1. Introduction

Although integrity constraints have long been used to maintain data consistency, nowadays there are many applications that have to deal with *inconsistent* data, that is, data violating integrity constraints (Arioua & Bonifati (2018); Geerts et al. (2014, 2013)). Thus, the problem of reasoning in the presence of inconsistent information has received much attention in the last decades.

The *consistent query answering* (CQA) framework is a principled approach to answer queries over inconsistent databases (Arenas et al. (1999)). It relies on the notions of *repair* and *consistent query answer*. Intuitively, a repair for a possibly inconsistent database is a consistent database that "minimally" differs from the original one. In general, there may be multiple repairs for an inconsistent database. The *consistent* (or

[1]Corresponding author. Address: Via P. Bucci 42C, 87036 Rende (CS), Italy. Phone: +39 0984 494727

certain) *answers* to a query over an inconsistent database are the query answers that can be obtained from every repair. The following example illustrates these notions.

Example 1. Consider the database schema consisting of two relations $\texttt{emp}(\texttt{Name}, \texttt{Dept})$ and $\texttt{dept}(\texttt{Name})$, where the former stores information on employees and departments they work for, and the latter stores all departments.

Suppose a referential integrity constraint is defined, stating that every department occurring in the emp relation must appear in the dept relation too. This constraint can be expressed as follows:

$$\forall \texttt{E}\, \forall \texttt{D}\, [\, \texttt{emp}(\texttt{E}, \texttt{D}) \wedge \neg\, \texttt{dept}(\texttt{D}) \Rightarrow]$$

Consider now the database D consisting of the following three facts: $\texttt{emp}(\texttt{john}, \texttt{cs})$, $\texttt{emp}(\texttt{john}, \texttt{math})$, and $\texttt{dept}(\texttt{math})$. Clearly, D is inconsistent, as the cs department appearing in the emp relation does not appear in the dept relation. A repair can be obtained by applying a minimal (under set-inclusion) set of update operations to the original database. We consider only fact insertions and deletions as admissible update operations.[2] Therefore, there exist two repairs: D_1, obtained by inserting the fact $\texttt{dept}(\texttt{cs})$ into D, and D_2, obtained by deleting the fact $\texttt{emp}(\texttt{john}, \texttt{cs})$ from D. The only consistent answer to the query asking for all departments' names is math. □

Although inconsistent databases can be repaired in different ways, in many applications it is natural and desirable to express that only a restricted set of update operations can be performed to restore consistency, which cannot be done with classical integrity constraints. *Active integrity constraints* (AICs) have been introduced to overcome such a limitation (Caroprese et al. (2009)). The basic idea is illustrated in the following example.

Example 2. Consider again the scenario of Example 1 and suppose that, when the integrity constraint is violated, we want to restore consistency only by adding missing departments (and thus avoid deleting facts of the emp relation). This behavior can be expressed by means of the following active integrity constraint:

$$\forall \texttt{E}\, \forall \texttt{D}\, [\, \texttt{emp}(\texttt{E}, \texttt{D}) \wedge \neg\, \texttt{dept}(\texttt{D}) \;\Rightarrow\; +\texttt{dept}(\texttt{D})\,]$$

The same constraint of Example 1 is defined on the left-hand side of $\Rightarrow$, while on the right-hand side the only admissible update operation is specified. Thus, only the insertion of $\texttt{dept}(\texttt{cs})$ can be performed to restore consistency of D, and D_1 is the only acceptable repair. As defined in the following, the insertion of $\texttt{dept}(\texttt{cs})$ is a "founded" update, because it is supported by the AIC above. In contrast, the deletion of $\texttt{emp}(\texttt{john}, \texttt{cs})$ is not a founded update, as it is not an admissible operation according to the AIC above. □

[2]Other minimality criteria and update operations, such as updating values within facts (Greco et al. (2018); Wijsen (2005); Greco & Molinaro (2008, 2012); Flesca et al. (2010)), have been considered in the literature. In this paper, we consider minimality under set-inclusion and insert/delete updates, which indeed are the most common minimality criteria and repair primitives considered in the literature.

Active integrity constraints allow users to express integrity constraints along with admissible update operations. One limitation of AICs is that they do not allow existential quantification, and thus do not allow users to formulate classical constraints such as foreign keys and more general inclusion dependencies, which require existentially quantified variables to be expressed (Cali et al. (2012)). The following example illustrates this aspect.

Example 3. Consider the database schema consisting of two relations $\texttt{emp}(\texttt{NameE}, \texttt{Dept})$ and $\texttt{dept}(\texttt{NameD}, \texttt{City})$, where the former stores information about employees and the departments they work for, while the latter stores information about all departments and the cities they are located in. Suppose that we want to define an inclusion dependency saying that every department in the `emp` relation must appear in the `dept` relation. Such a constraint cannot be expressed by means of AICs. What we need are existentially quantified variables. The aforementioned constraint can in fact be expressed as follows:

$$\forall \texttt{E}\, \forall \texttt{D}\, [\, \texttt{emp}(\texttt{E}, \texttt{D}) \wedge \nexists \texttt{C}\, \texttt{dept}(\texttt{D}, \texttt{C}) \Rightarrow]$$

However, no policy is stated as to how to resolve inconsistency when the constraint is violated. □

We can lift the idea of AICs (that is, to specify which update operations should be applied when a constraint is violated) to integrity constraints like the one in the previous example. This leads us to *existential active integrity constraints* (EAICs), introduced in this paper, which generalize AICs enabling users to express a wider class of integrity constraints commonly arising in practice (e.g., inclusion dependencies), as well as policies to restore consistency.

Example 4. The following EAIC:

$$\forall \texttt{E}\, \forall \texttt{D}\, [\, \texttt{emp}(\texttt{E}, \texttt{D}) \ \wedge \nexists \texttt{C}\, \texttt{dept}(\texttt{D}, \texttt{C}) \ \Rightarrow \exists \texttt{Z}\, {+}\texttt{dept}(\texttt{D}, \texttt{Z}) \,]$$

defines the same constraint of Example 3, but it also states that inconsistency must be resolved by adding missing departments to relation `dept`.

Importantly, EAICs lead to value invention because of existentially quantified variables, which is not the case for AICs, and this poses different new issues—for instance, for a database containing only the fact $\texttt{emp}(\texttt{john}, \texttt{cs})$, a city for the `cs` department needs to be invented. □

Contributions. This paper introduces *existential active integrity constraints* (EAICs), an extension of AICs allowing existential variables. EAICs allow users to express both data dependencies and policies to fix violations, and generalize several classes of constraints, including tuple-generating dependencies and formalisms for ontological reasoning. We introduce the notion of a *universal set of founded updates*, a restricted subset of founded updates that are sufficient to compute consistent query answers. We also identify an expressive subclass having a "deterministic" behavior, that is, to answer (positive) queries it suffices to consider only one founded update.

Organization. The rest of the paper is organized as follows. Preliminaries on (active) integrity constraints are reported in Section 2. In Section 3, we present the syntax and semantics of existential active integrity constraints, and introduce the important notion of a *universal set of founded updates*. In Section 4, we study restrictions guaranteeing finiteness of universal sets of founded updates, whereas in Section 5, we study the role of disjunction and the class of *normal* (i.e., disjunction-free) EAICs. In Section 6, we define a restricted class of EAICs having a "deterministic" behavior (i.e., for consistent query answering, it suffices to look at a single founded update). Related work is discussed in Section 7. Finally, we draw conclusions in Section 8.

2. Preliminaries

We assume the existence of the following (pairwise disjoint) enumerable sets: *predicates* $\mathcal{P}$, *variables* $\mathcal{V}$, and *constants* $\mathcal{C}$. Each predicate is associated with an *arity*, which is a non-negative integer. A *term* is either a constant or a variable.

An *atom* is of the form $p(t_1, \ldots, t_n)$, where p is a predicate of arity n and the t_i's are terms—when the atom's predicate is relevant, we call such an atom a p-atom. We write an atom containing only constants also as $p(\overline{c})$, where $\overline{c}$ is understood to be a sequence of constants, and write $p(\overline{X})$ to refer to an atom whose terms are the variables $\overline{X}$.

A *literal* is either an atom A (*positive* literal) or its negation $\neg A$ (*negative* literal). Logical formulae are built using literals and logical connectives—the syntax of the logical formulae we are interested in will be precisely defined later.

A term/atom/literal/formula is *ground* if it is variable-free. A formula φ' is a *ground instance* of a formula φ if φ' can be obtained from φ by substituting every variable in φ with a constant. We use $ground(\varphi)$ to denote the set of all ground instances of φ, and for a set of formulae Φ, we define $ground(\Phi) = \cup_{\varphi \in \Phi} ground(\varphi)$.

2.1. Integrity Constraints

We assume the standard concepts of the relational data model. A database is a collection of relations. Each relation is a finite set of tuples of constants and has a finite set of attributes. Each tuple $\overline{c}$ of constants in a relation p can be viewed as a ground atom $p(\overline{c})$, called *fact*; then a database can be viewed as a finite set of facts.

We consider queries expressed by means of safe nonrecursive Datalog programs, which is equivalent to relational algebra (RA) and safe relational calculus (RC). A *conjunctive* query consists of a Datalog rule of the form $Q(\overline{Z}) :- \varphi(\overline{X})$, where φ is a conjunction of atoms over the variables in $\overline{X}$, and $\overline{Z} \subseteq \overline{X}$. Such a query is equivalent to a RA query using only the following RA operators: Cartesian product, projection, and *positive* selection, that is, selection conditions are restricted to be conjunctions of equalities. By *positive queries* we mean safe RC queries and nonrecursive Datalog queries without negation, which are equivalent to unions of conjunctive RA queries. Although the choice of the language is not relevant for our purposes, from now on we assume that queries are formulated using safe, nonrecursive Datalog without negation. The result of evaluating a query Q on a database D will be denoted as $Q(D)$.

An (*universally quantified* or *full*) *integrity constraint* (IC) is of the form:

$$\forall \overline{X} \left[\bigwedge_{i=1}^{m} b_i(\overline{X}_i) \wedge \bigwedge_{i=m+1}^{n} \neg\, b_i(\overline{X}_i) \Rightarrow \right] \tag{1}$$

where $n \geq m \geq 0$, for every $1 \leq i \leq n$, the $b_i(\overline{X}_i)$'s are atoms, and the above conjunction is *safe*, that is, variables occurring in negative literals also occur in positive literals.

A database D is *consistent* w.r.t. a set of integrity constraints Σ (we also say that D *satisfies* Σ) if $D \models \Sigma$ in the standard model-theoretic sense. Otherwise, D is *inconsistent* w.r.t. Σ.

An *update atom* is of the form $+a(\overline{X})$ or $-a(\overline{X})$, where $a(\overline{X})$ is an atom. A variable-free update atom is said to be *ground*. Intuitively, a ground update atom $+a(\overline{c})$ (resp. $-a(\overline{c})$) states that $a(\overline{c})$ will be inserted into (resp. deleted from) the database. We use the notation $\pm a(\overline{c})$ to refer to a generic ground update atom, which may be either $+a(\overline{c})$ or $-a(\overline{c})$. Given a set $\mathcal{U}$ of ground update atoms, we define $\mathcal{U}^+ = \{a(\overline{c}) \mid +a(\overline{c}) \in \mathcal{U}\}$ and $\mathcal{U}^- = \{a(\overline{c}) \mid -a(\overline{c}) \in \mathcal{U}\}$. We say that $\mathcal{U}$ is *coherent* if it does not contain two update atoms $+a(\overline{c})$ and $-a(\overline{c})$ (i.e., if $\mathcal{U}^+ \cap \mathcal{U}^- = \emptyset$). Given a database D and a coherent set of update atoms $\mathcal{U}$, we use $\mathcal{U}(D)$ to denote the database obtained by applying $\mathcal{U}$ to D, that is, the database $(D \cup \mathcal{U}^+) \setminus \mathcal{U}^-$. When $\mathcal{U}$ is a singleton $\{\pm A\}$, with a slight abuse of notation, we also write $\pm A$ in place of $\{\pm A\}$.

Given a database D and a set of integrity constraints Σ, an *update* for $\langle D, \Sigma \rangle$ is a coherent set of ground update atoms $\mathcal{R}$ such that *(i)* $\mathcal{R}(D) \models \Sigma$ and *(ii)* there is no proper subset $\mathcal{R}'$ of $\mathcal{R}$ such that $\mathcal{R}'(D) \models \Sigma$. The set of all possible updates for $\langle D, \Sigma \rangle$ is denoted as $\mathbf{R}(D, \Sigma)$. Every database obtained by applying an update $\mathcal{R}$ to D is called a *repair* for $\langle D, \Sigma \rangle$. Thus, repairs are consistent databases derived from the original one by means of a minimal set of update operations—recall that we consider fact insertions and deletions as the only primitives to restore consistency.

The *certain* (or *consistent) answers* to a query Q on a database D w.r.t. a set of integrity constraints Σ are defined as:

$$\textsc{certain}(Q, D, \Sigma) = \bigcap_{\mathcal{R} \in \mathbf{R}(D,\Sigma)} Q(\mathcal{R}(D)).$$

2.2. Active Integrity Constraints

In this section we recall the syntax and semantics of *active integrity constraints* (AICs) (Caroprese et al. (2009)). We consider formulae of the form $\varphi \Rightarrow \psi$, where φ is a conjunction of literals and ψ is a disjunction of update atoms. For any formula $\varphi \Rightarrow \psi$, $body(\varphi \Rightarrow \psi) = \varphi$ denotes the body of the implication, whereas $head(\varphi \Rightarrow \psi) = \psi$ denotes the head. With a slight abuse of notation, we sometimes use $body(\sigma)$ (resp. $head(\sigma)$) to denote the *set* of body literals (resp. head update atoms).

The *complementary literal* of an update atom $+a(\overline{X})$ (resp. $-a(\overline{X})$) is $CompLit(+a(\overline{X})) = \neg\, a(\overline{X})$ (resp. $CompLit(-a(\overline{X})) = a(\overline{X})$). For any set $\mathcal{U}$ of update atoms, $CompLit(\mathcal{U}) = \{CompLit(\pm a(\overline{X})) \mid \pm a(\overline{X}) \in \mathcal{U}\}$

Definition 1. An *active integrity constraint* (AIC) σ is of the form:

$$\forall \overline{X} \left[\bigwedge_{i=1}^{m} b_i(\overline{X}_i) \wedge \bigwedge_{i=m+1}^{n} \neg\, b_i(\overline{X}_i) \Rightarrow \bigvee_{i=1}^{q} -a_i(\overline{X}_i) \vee \bigvee_{i=q+1}^{p} +a_i(\overline{X}_i) \right] \tag{2}$$

where *(i)* $n, p > 0$, *(ii)* the $b_i(\overline{X}_i)$'s are atoms, *(iii)* the $-a_i(\overline{X}_i)$'s and $+a_i(\overline{X}_i)$'s are update atoms, *(iv)* variables occurring in negated literals also occur in positive standard literals, and *(v)* $CompLit(head(\sigma)) \subseteq body(\sigma)$. □

For an AIC σ, $body^{+}(\sigma)$ and $body^{-}(\sigma)$ denote the set of positive and negated atoms in $body(\sigma)$, respectively. An AIC specifies both an integrity constraint (in the body) and the actions to be performed (in the head) if the integrity constraint is violated. We use $St(\sigma)$ to denote the integrity constraint of the form (1) derived from σ by removing all the head update atoms. For a set of active integrity constraints Σ, $St(\Sigma)$ denotes the corresponding set of integrity constraints, that is $St(\Sigma) = \{St(\sigma) \mid \sigma \in \Sigma\}$. Furthermore, for any set of AICs Σ and set of ground update atoms $\mathcal{U}$, $\Sigma[\mathcal{U}]$ denotes the set of AICs derived from $ground(\Sigma)$ by deleting head update atoms not occurring in $\mathcal{U}$ and AICs such that all head update atoms have been deleted.

Below we define the semantics of AICs. For every database D and set of AICs Σ, a set $\mathcal{R}$ of ground update atoms is an *update* for $\langle D, \Sigma \rangle$ if it is an update for $\langle D, St(\Sigma) \rangle$. The set of updates for $\langle D, \Sigma \rangle$ is denoted as $\mathbf{R}(D, \Sigma)$.

Definition 2. *Given a database D and a set of AICs Σ:*

- *An update $\mathcal{R}$ for $\langle D, \Sigma \rangle$ is* founded *iff it is an update for $\langle D, \Sigma[\mathcal{R}] \rangle$.*
- *A repair $\mathcal{R}(D)$ is* founded *iff $\mathcal{R}$ is a founded update.* □

The idea underlying the above definition is that the actions of an update must be determined only by the AICs allowing those actions. Observe that the founded semantics guarantees that, given a founded repair $\mathcal{R}$, for each update atom $\pm A \in \mathcal{R}$ there must be an AIC $\sigma \in gorund(\Sigma)$ such that $\pm A \in head(\sigma)$ (otherwise $\mathcal{R}$ is not minimal).

Example 5. Consider the database $D_5 = \{\mathtt{p}\}$ and the set of AICs Σ_5:

$$\begin{array}{ll} \sigma_1 : \mathtt{p} \wedge \mathtt{q} & \Rightarrow -\mathtt{p} \\ \sigma_2 : \mathtt{p} \wedge \neg \mathtt{q} & \Rightarrow +\mathtt{q} \end{array}$$

For $\langle D_5, \Sigma_5 \rangle$ there is a unique update $\mathcal{R} = \{-\mathtt{p}\}$, which is not founded as update $-\mathtt{p}$ is not determined by the first AIC, but (indirectly) by the second one. Observe that $\Sigma_5[\mathcal{R}] = \{\sigma_1\}$ and $\langle D, \Sigma_5[\mathcal{R}] \rangle$ has only one update $\mathcal{R}' = \emptyset$. □

It is worth noting that although the definition here introduced is different from that used in Caroprese et al. (2009), here we use the same name as the former is just a refinement of the latter, and its purpose is to overcome the problem of cyclic support—see Caroprese & Truszczynski (2011). Theorem 8 in Appendix A shows that every

founded update according to Definition 2 is also founded according to the definition given in Caroprese et al. (2009).

The set of all founded updates for a database D and a set of AICs Σ is denoted as $\mathbf{FR}(D, \Sigma)$. Clearly, $\mathbf{FR}(D, \Sigma) \subseteq \mathbf{R}(D, \Sigma)$.

Example 6. Consider the scenario of Example 2 and let Σ_6 be the set containing only the AIC σ reported therein.
Then, $\mathbf{R}(D,\Sigma_6) = \{\mathcal{R}_1, \mathcal{R}_2\}$, where $\mathcal{R}_1 = \{-\texttt{emp}(\texttt{john}, \texttt{cs})\}$ and $\mathcal{R}_2 = \{+\texttt{dept}(\texttt{cs})\}$. Moreover $\Sigma_6[\mathcal{R}_1] = \emptyset$, whereas $\Sigma_6[\mathcal{R}_2] = ground(\Sigma_6)$. Consequently, $\mathcal{R}_1$ is not founded, whereas $\mathcal{R}_2$ is founded. Therefore, $\mathbf{FR}(D, \Sigma_6) = \{\mathcal{R}_2\}$. □

The *certain* answers to a query Q on a database D w.r.t. a set of AICs Σ are:

$$\text{CERTAIN}(Q, D, \Sigma) = \bigcap_{\mathcal{R} \in \mathbf{FR}(D,\Sigma)} Q(\mathcal{R}(D)).$$

Thus, in the presence of active integrity constraints, the certain answers to a query are defined by considering only founded repairs, rather than all of them.

3. Existential Active Integrity Constraints

As shown in the Introduction, AICs suffer from limited expressivity, due to the lack of existential quantification. As already anticipated, we are now going to propose an extension of AICs, which lets us define AICs with existentially quantified variables, allowing for more expressive integrity constraints, such as inclusion dependencies. In particular, EAICs additionally allow update atoms with existential variables in the head, as well as negative body literals of a particular form. For instance, the EAIC of Example 4, which we report again below for convenience,

$$\forall \texttt{E}\, \forall \texttt{D}\, [\, \texttt{emp}(\texttt{E}, \texttt{D}) \ \wedge \nexists \texttt{C}\ \texttt{dept}(\texttt{D}, \texttt{C}) \ \Rightarrow \exists \texttt{Z}\ +\texttt{dept}(\texttt{D}, \texttt{Z})\,]$$

has the existentially quantified update atom $\exists \texttt{Z}\ +\texttt{dept}(\texttt{D}, \texttt{Z})$ in the head, and the negative body literal $\nexists \texttt{C}\ \texttt{dept}(\texttt{D}, \texttt{C})$ in the body—neither of them is allowed by AICs. More in general, as stated in the following, EAICs additionally allow update atoms of the form $\exists \overline{Z} + a(\overline{X}, \overline{Z})$ in the head, and negative literals of the form $\nexists \overline{Z} a(\overline{X}, \overline{Z})$ in the body.

The set of *complementary literals* of an update atom is redefined as follows:

- $CompLit(-a(\overline{X})) = \{a(\overline{X})\}$;
- $CompLit(\exists \overline{Z} + a(\overline{X}, \overline{Z})) = \{\nexists \overline{Y} a(\overline{X}', \overline{Y}) \mid \exists\, subst.\, \vartheta\ s.t.\ a(\overline{X}', \vartheta(\overline{Y})) \ = \ a(\overline{X}, \overline{Z})\}$.

For any set $\mathcal{U}$ of update atoms, $CompLit(\mathcal{U}) = \cup_{\pm A \in \mathcal{U}} CompLit(\pm A)$

Syntax. The following definition introduces the syntax of existential active integrity constraints.

Definition 3. An *Existential Active Integrity Constraint* (EAIC) is of the form:

$$\forall \overline{X} \left[\bigwedge_{i=1}^{m} b_i(\overline{X}_i) \wedge \bigwedge_{i=m+1}^{n} \nexists \overline{Z}_i \, b_i(\overline{X}_i, \overline{Z}_i) \Rightarrow \bigvee_{i=1}^{q} -a_i(\overline{X}_i) \vee \bigvee_{i=q+1}^{p} \exists \overline{Z}_i \, +a_i(\overline{X}_i, \overline{Z}_i) \right] \quad (3)$$

where *(i)* $n, p > 0$, *(ii)* universally quantified variables occurring in negative body literals also occur in positive body literals, *(iii)* every existentially quantified variable occurs only in one update atom or negative body literal, and *(iv)* for each $\pm A \in head(\sigma)$, the condition $CompLit(\pm A) \cap body(\sigma) \neq \emptyset$ holds. □

Example 7. Consider the database schema consisting of two relations $\texttt{node}(\texttt{Id})$ and $\texttt{edge}(\texttt{Source}, \texttt{Dest}, \texttt{Weight})$ used to store nodes and weighted edges of a graph, respectively. For the following EAIC

$$\sigma: \ \texttt{node}(\texttt{X}_1) \wedge \texttt{node}(\texttt{X}_2) \ \wedge \nexists(\texttt{Y}_1,\texttt{Y}_2) \ \texttt{edge}(\texttt{X}_1, \texttt{Y}_1, \texttt{Y}_2) \Rightarrow \exists \texttt{Z}_1 + \texttt{edge}(\texttt{X}_1, \texttt{X}_2, \texttt{Z}_1),$$

we have $CompLit(\exists \texttt{Z}_1 + \texttt{edge}(\texttt{X}_1, \texttt{X}_2, \texttt{Z}_1)) \cap body(\sigma) = \{\nexists(\texttt{Y}_1,\texttt{Y}_2) \ \texttt{edge}(\texttt{X}_1, \texttt{Y}_1, \texttt{Y}_2)\} \neq \emptyset$. □

A negative body literal $\nexists \overline{Z}_i \, b_i(\overline{X}_i, \overline{Z}_i)$ s.t. $\overline{Z}_i$ is empty will be simply written as $\neg \, b_i(\overline{X}_i)$. For an EAIC σ, $St(\sigma)$ denotes the constraint, called *existential integrity constraint* (EIC), obtained by deleting all head atoms from σ. For any set of EAICs Σ, we define $St(\Sigma) = \{St(\sigma) \mid \sigma \in \Sigma\}$. For ease of presentation (and w.l.o.g.), from now on we assume that constants do not appear in EAICs.

Semantics. We use $pground(\varphi)$ to denote the set of all *partially ground instances* of a formula φ obtained by replacing universally quantified variables with constants in all possible ways. For a set of formulae Φ, $pground(\Phi) = \cup_{\varphi \in \Phi} \, pground(\varphi)$.

A database D *satisfies* a partially ground conjunction of literals φ (denoted $D \models \varphi$), if $\varphi^+ \subseteq D$ and there is no substitution ϑ replacing existentially quantified variables in φ with constants s.t. $\vartheta(\varphi^-) \cap D \neq \emptyset$.[3] Thus, for any partially ground EIC σ of the form $\varphi \Rightarrow$, as it expresses a denial constraint (i.e., an implication whose head is false), $D \models \sigma$ iff $D \not\models \varphi$, that is, the following condition holds: if $body^+(\sigma) \subseteq D$, then there is a substitution ϑ replacing existentially quantified variables with constants s.t. $\vartheta(body^-(\sigma)) \cap D \neq \emptyset$. Furthermore, D *satisfies* an EIC σ if D satisfies every partially ground instance in $pground(\sigma)$; D *satisfies* an EAIC (or partially ground instance thereof) σ if it satisfies $St(\sigma)$. Finally, D *satisfies* a set of EAICs (or EICs) Σ if D satisfies every $\sigma \in \Sigma$—we also say that D is *consistent* w.r.t. Σ. Updates and repairs for databases with EICs and EAICs can be defined analogously to the cases of ICs and AICs, respectively.

Example 8. Given the database schema consisting of two relations $\texttt{node}(\texttt{Id})$ and $\texttt{edge}(\texttt{Source}, \texttt{Dest})$ used to store nodes and edges of a graph, respectively, consider the following EAIC:

$$\texttt{node}(\texttt{X}) \ \wedge \nexists \texttt{Y} \ \texttt{edge}(\texttt{X}, \texttt{Y}) \Rightarrow -\texttt{node}(\texttt{X}) \vee \exists \texttt{Z} \ + \texttt{edge}(\texttt{X}, \texttt{Z})$$

[3] Here φ^+ and φ^- denote the sets of positive and negated atoms in φ, respectively.

Intuitively, the EAIC says that every node must have an outgoing edge, and when this is not the case either the node is deleted or an outgoing edge is added.

The database $D = \{\texttt{node(a)}\}$ is clearly inconsistent. Since the domain $\mathcal{C}$ is infinite, the EAIC above suggests an infinite number of ways to repair the database, namely, by means of update atoms of the form $\{+\texttt{edge(a, c)}\}$ with $\texttt{c} \in \mathcal{C}$. Notice that $\{-\texttt{node(a)}\}$ is another possible way of restoring consistency. □

For any set of EAICs Σ and set of ground update atoms $\mathcal{U}$, $\Sigma[\mathcal{U}]$ denotes the set of partially ground EAICs derived from $pground(\Sigma)$ by first deleting every head update atom $\pm A$ for which there does not exist a substitution ϑ such that $\vartheta(\pm A) \in \mathcal{U}$, and then deleting every EAICs where all head update atoms have been deleted.

The definitions of founded update and founded repair are the same as those defined for AICs, that is, for any database D and set of EAICs Σ: (i) an update $\mathcal{R}$ is *founded* iff it is an update for $\langle D, \Sigma[\mathcal{R}]\rangle$, and (ii) a repair $\mathcal{R}(D)$ is *founded* iff $\mathcal{R}$ is a founded.

3.1. Nulls and Certain Query Answers

The introduction of existentially quantified variables increases the expressivity of active integrity constraints. The price to pay is that, differently from the AIC setting, decidability of query answering over knowledge bases with EAICs is no more guaranteed, in general. Example 8 showed that EAICs can admit an infinite number of updates, whereas the following one shows that EAICs can admit updates of infinite size (i.e., containing an infinite number of update atoms).

Example 9. Consider again the database schema of Example 8 and the set of EAICs Σ_9 consisting of the following EAIC:

$$\texttt{edge(X, Y)} \wedge \nexists \texttt{V}\ \texttt{edge(Y, V)} \Rightarrow \exists \texttt{Z}\ +\texttt{edge(Y, Z)}$$

Intuitively, the EAIC says that every node having an incoming edge must also have an outgoing edge, and when this does not hold the missing outgoing edge must be added.

The database $D = \{\texttt{edge}(\texttt{c}_1, \texttt{c}_2)\}$ is inconsistent. Assuming an infinite domain of constants $\mathcal{C} = \{\texttt{c}_1, \texttt{c}_2, \texttt{c}_3, \texttt{c}_4, \dots\}$, the set $\{+\texttt{edge}(\texttt{c}_2, \texttt{c}_3), +\texttt{edge}(\texttt{c}_3, \texttt{c}_4), \dots\}$ is an update with an infinite number of update atoms. □

To restrict the number of repairs to be considered for query evaluation, we next introduce the concepts of labeled null and universal repairs (see the next Subsection 3.2).

A labeled null can be used as a placeholder for any constant from $\mathcal{C}$. Thus, in addition to the set of constants $\mathcal{C}$, we assume the existence of an infinite enumerable set of labeled nulls $\mathcal{N}$ of the form $\perp_i$, where $i \in \mathbb{N}$ is a natural number. Below we report some auxiliary notation and terminology to deal with labeled nulls—some of them are slight generalizations of previously introduced notions to account for labeled nulls too, besides constants.

For any set of atoms D with values in $\mathcal{C} \cup \mathcal{N} \cup \mathcal{V}$, we use $\mathcal{C}(D)$ (resp. $\mathcal{N}(D)$, $\mathcal{V}(D)$) to denote the set of constants (resp. nulls, variables) occurring in D.

For every two sets of atoms D_1 and D_2 over S, a homomorphism h from D_1 to D_2, denoted $h : D_1 \to D_2$, is a mapping from $\mathcal{C}(D_1) \cup \mathcal{N}(D_1) \cup \mathcal{V}(D_1)$ to $\mathcal{C}(D_2) \cup \mathcal{N}(D_2) \cup \mathcal{V}(D_2)$ such that:

(i) $h(c) = c$, for every $c \in \mathcal{C}(D_1)$;

(ii) $h(\bot_i) \in \mathcal{C}(D_2) \cup \mathcal{N}(D_2)$, for every $\bot_i \in \mathcal{N}(D_1)$;

(iii) for every fact $R_i(\bar{t})$ of D_1, we have that $R_i(h(\bar{t}))$ is a fact of D_2 (where, if $\bar{t} = (a_1, ..., a_n)$, then $h(\bar{t}) = (h(a_1), ..., h(a_n))$).

A homomorphism that is the identity on $\mathcal{C} \cup \mathcal{N}$ (i.e., it maps variables only) is also called a *substitution*, whereas a substitution whose image is $\mathcal{C} \cup \mathcal{N}$ (resp. $\mathcal{C}$) is called a *matcher* (resp. *constant matcher*). The concepts of homomorphism can be extended to (sets of) update atoms.

The new definitions of ground (update) atom, update and repair are given in the following. A *ground atom* A is of the form $p(t_1, \ldots, t_n)$, where p is an n-ary predicate and $t_1, \ldots, t_n \in \mathcal{C} \cup \mathcal{N}$; we write it also as $p(\bar{t})$, where $\bar{t}$ is understood to be a sequence of constants and labeled nulls. Intuitively, A represents all atoms $B = p(c_1, \ldots, c_n)$, with $c_1, \ldots, c_n \in \mathcal{C}$, such that there exists a homomorphism from A to B. A *ground update atom* is of the form $+p(\bar{t})$ or $-p(\bar{t})$, where $p(\bar{t})$ is a ground atom. We use $\pm p(\bar{t})$ to refer to a generic ground update atom.

Certain answers. The semantics of a database D with labeled nulls is usually given in terms of the set $\text{POSS}(D)$ of its *possible worlds*, that is, all databases that can be obtained from D by replacing all nulls with constants. The *certain answers* to a query Q over D are defined as follows:

$$\text{CERTAIN}(Q, D) = \bigcap_{W \in \text{POSS}(D)} Q(W)$$

The definitions of partially ground constraints remains the same. A database D with labeled nulls *satisfies* a partially ground EIC σ if the following condition holds: for every homomorphism h from $body^+(\sigma)$ to D that maps nulls to constants, there is a constant matcher ϑ s.t. $\vartheta(body^-(\sigma)) \cap h(D) \neq \emptyset$. The definitions of satisfaction of (sets of) EICs and EAICs remain the same.

Moreover, the introduction of nulls implies that the definitions of coherent update atoms and update need to be revised.

Definition 4. A set of ground update atoms $\mathcal{U}$ is *coherent* if there are no two update atoms $+a(t_1), -a(t_2) \in \mathcal{U}$ and a homomorphism h such that $a(h(t_1)) = a(h(t_2))$.

Given two coherent sets of ground update atoms $\mathcal{U}_i$ and $\mathcal{U}_j$, we say that $\mathcal{U}_i$ is *more general than* $\mathcal{U}_j$, denoted $\mathcal{U}_i \sqsupseteq \mathcal{U}_j$, if there exists a homomorphism h from $\mathcal{U}_i$ to $\mathcal{U}_j$.

We also say that $\mathcal{U}_i$ and $\mathcal{U}_j$ are *(homomorphically) equivalent* (and write $\mathcal{U}_i \equiv \mathcal{U}_j$) if $\mathcal{U}_i \sqsupseteq \mathcal{U}_j$ and $\mathcal{U}_j \sqsupseteq \mathcal{U}_i$. □

For instance, $\{+\texttt{edge}(\bot_1, \bot_2)\} \sqsupseteq \{+\texttt{edge}(\texttt{a}, \bot_2)\} \sqsupseteq \{+\texttt{edge}(\texttt{a}, \texttt{a})\}$, and the sets $\{+\texttt{edge}(\texttt{a}, \bot_1)\}$, $\{+\texttt{edge}(\bot_2, \texttt{a})\}$, $\{-\texttt{node}(\texttt{a})\}$ are pairwise incomparable.

The introduction of nulls makes the minimality criterion of updates for the case of AICs and databases with constants only, not suitable anymore. Thus, the next definition extends the notion of update to EAICs.

Definition 5 (Update). Given a database D and a set of EICs Σ, an *update* for $\langle D, \Sigma \rangle$ is a coherent set of ground update atoms $\mathcal{R}$ such that *(i)* $\mathcal{R}(D) \models \Sigma$, and *(ii)* for every coherent set of ground update atoms $\mathcal{R}'$ such that $\mathcal{R}'(D) \models \Sigma$ if $\mathcal{R}' \sqsupseteq \mathcal{R}$, then also $\mathcal{R} \sqsupseteq \mathcal{R}'$ holds. □

Observe that the previous definition coincides with the one provided in Section 2 when applied to AICs, as update atoms contain only constants and $\mathcal{R} \sqsupseteq \mathcal{R}'$ is equivalent to $\mathcal{R} \subseteq \mathcal{R}'$.

Once we have revised the definition of coherent set of ground update atoms, update, founded update atom, founded update and founded repair are defined analogously to the case of AICs. Thus, we continue to use the same notation even for EAICs.

In the presence of a set of EAICs Σ, the definition of certain answers has to take into account all possible founded repairs for $\langle D, \Sigma \rangle$ and, for each founded repair, all its possible worlds. Thus,

$$\text{CERTAIN}(Q, D, \Sigma) = \bigcap_{\mathcal{R} \in \mathbf{FR}(D,\Sigma) \wedge M \in \text{POSS}(\mathcal{R}(D))} Q(M)$$

The following proposition states that certain query answering is undecidable in the presence of EAICs.

Proposition 1. *Deciding whether a tuple belongs to* $\text{CERTAIN}(Q, D, \Sigma)$ *is undecidable.*

Proof. It can be proved by reduction of the analogous certain answer problem defined for databases with TGDs. In such a case the certain answer problem is defined as $\text{CERTAIN}(Q, D, \Sigma) = \bigcap\{S \mid S \; is \; a \; solution \; for \; \langle D, \Pi \rangle\}$, where Π is a set of TGDs. A solution for $\langle D, \Pi \rangle$ is a set of tuples such that $D \subseteq S$ and $S \models \Sigma$. The problem of deciding whether a tuple belongs to $\text{CERTAIN}(Q, D, \Pi)$, where Π is a set of TGDs, is undecidable. Thus, the undecidability of the problem follows from the fact that, as we will show in Section 6.2, any set of TGDs Π can be mapped to a set of EAICs $eaic(\Pi)$ so that (universal) solutions for $\langle D, \Pi \rangle$ are equivalent to (universal) founded repairs for $\langle D, eaic(\Pi) \rangle$ (see Theorem 7 in Section 6.2) and both the problems of stating whether $\langle D, \Pi \rangle$ admits a (universal) solution and of stating whether a tuple is a certain answer for a query Q over a database D with dependencies Π, are, in the general case, undecidable. □

The previous result does not preclude, however, the existence of interesting classes of EAICs for which the problem of computing certain query answers is decidable, and even tractable. It is worth noting that it might be the case that there are no founded updates for a database and set of EAICs. This may already happen in the presence of AICs, as shown in Example 5.

In the next subsection we will introduce a property that allows us to select a minimal set of founded updates to compute certain answers.

3.2. Universal Set of Founded Updates

Although the introduction of nulls enlarges the number of (founded) updates, only a subset of these need to be considered in computing certain answers. This idea is captured by the notion of "universal set of founded updates", introduced in the following definition.

Definition 6 (Universal Set of Updates). Let D be a database and Σ a set of EAICs. A *universal set of updates* S is a minimal (w.r.t. $\subseteq$) set of S-updates for $\langle D, \Sigma \rangle$ s.t. for every update $\mathcal{R}_j$ for $\langle D, \Sigma \rangle$ there is an S-update $\mathcal{R}_i \in S$ s.t. $\mathcal{R}_i \sqsupseteq \mathcal{R}_j$. □

Roughly speaking, a universal set of founded updates is a set of founded updates that is representative of all founded updates. It is worth noting that the minimality property guarantees that a universal set of founded updates contains only founded incomparable updates.

Example 10. Consider the database and the EAIC of Example 8. The founded updates are $\mathcal{R}_0 = \{-\texttt{node}(\texttt{a})\}$, every $\mathcal{R}_i = \{+\texttt{edge}(\texttt{a}, \bot_\texttt{i})\}$, for some $i \in \mathbb{N}$, and every $\mathcal{R}_c = \{+\texttt{edge}(\texttt{a}, \texttt{c})\}$, for some $c \in \mathcal{C}$. The sets of the form $\{\mathcal{R}_0, \mathcal{R}_i\}$ are universal sets of founded updates, whereas the sets of the form $\{\mathcal{R}_0, \mathcal{R}_c\}$, for some $c \in \mathcal{C}$, are not. □

In the previous example we have an infinite number of universal sets of founded updates, but all of them are homomorphically equivalent. However, as previously pointed out, it might also happen that there is no founded update for a database D and set of EAICs Σ.

For any query Q, database D, and set of EAICs Σ, let S be a universal set of founded updates for $\langle D, \Sigma \rangle$, then the certain answers to Q over S are:

$$\textsc{certain}(Q, D, S) = \bigcap_{\mathcal{R} \in S \wedge M \in \textsc{poss}(\mathcal{R}(D))} Q(M)$$

Theorem 1. *Let D be a database, Σ a set of EAICs, and Q a query. Then, for any universal set of founded updates S for $\langle D, \Sigma \rangle$*

$$\textsc{certain}(Q, D, \Sigma) = \textsc{certain}(Q, D, S)$$

Proof. We show that *(i)* $\textsc{certain}(Q, D, \Sigma) \subseteq \textsc{certain}(Q, D, S)$ and *(ii)* $\textsc{certain}(Q, D, \Sigma) \supseteq \textsc{certain}(Q, D, S)$.

(i) To prove that $\textsc{certain}(Q, D, \Sigma) \subseteq \textsc{certain}(Q, D, S)$ we have to show that

$$\bigcap_{\mathcal{R} \in \mathbf{FR}(D, \Sigma) \wedge M \in \textsc{poss}(\mathcal{R}(D))} Q(M) \subseteq \bigcap_{\mathcal{R} \in S \wedge N \in \textsc{poss}(\mathcal{R}(D))} Q(N).$$

This is trivial as $S \subseteq \mathbf{FR}(D, \Sigma)$.

(ii) To prove that $\text{CERTAIN}(Q, D, S) \subseteq \text{CERTAIN}(Q, D, \Sigma)$ we have to show that

$$\bigcap_{\mathcal{R} \in S \wedge N \in \text{POSS}(\mathcal{R}(D))} Q(N) \subseteq \bigcap_{\mathcal{R} \in \mathbf{FR}(D,\Sigma) \wedge M \in \text{POSS}(\mathcal{R}(D))} Q(M).$$

For any $\mathcal{R} \in \mathbf{FR}(D, \Sigma)$ there is an $\mathcal{R}' \in S$ such that $\mathcal{R}' \sqsupseteq \mathcal{R}$. This means that that for every ground database $M \in \text{POSS}(\mathcal{R}(D))$, then $M \in \text{POSS}(\mathcal{R}'(D))$. Therefore $\text{CERTAIN}(Q, D, S) \subseteq \text{CERTAIN}(Q, D, \Sigma)$. □

Therefore, to answer queries it is possible to consider any universal set of founded updates. In the following, for each database D and set of EAICs Σ, $\mathbf{US}(D, \Sigma)$ denotes an arbitrary, but fixed, universal set of (founded) updates of $\langle D, \Sigma \rangle$. Moreover, whenever the universal set of updates contains only one update, we call such an element universal update. As a consequence, certain answers can be computed by only considering the repairs obtained by taking any universal set of founded updates; we call such a set of repairs *universal set of repairs*.

Corollary 1. *Given a query Q, a database D, and a set of EAICs Σ,*

$$\text{CERTAIN}(Q, D, \Sigma) = \bigcap_{\mathcal{R} \in \mathbf{US}(D,\Sigma) \wedge M \in \text{POSS}(\mathcal{R}(D))} Q(M)$$

Proof. Straightforward from Theorem 1. □

For databases (without EAICs) containing labeled nulls, certain answers to positive queries can be easily computed in polynomial time as follows (Imielinski & Lipski Jr. (1984)): first, the query is evaluated by treating nulls as standard constants, and then tuples containing nulls are discarded from the result of the query evaluation. This evaluation is sometimes called *naive* evaluation. We use $Q^{naive}(D)$ to denote the result of the naive evaluation of a query Q over a database (possibly with labeled nulls) D. As a consequence, we obtain the following corollary.

Corollary 2. *Given a positive query Q, a database D, and a set of EAICs Σ,*

$$\text{CERTAIN}(Q, D, \Sigma) = \bigcap_{\mathcal{R} \in \mathbf{US}(D,\Sigma)} Q^{naive}(\mathcal{R}(D))$$

Proof. It follows from Corollary 1 and the correctness of the naive evaluation for positive queries (Imielinski & Lipski Jr. (1984)). □

4. Finite Universal Set of Founded Updates

In the previous section, we have shown that there exists a "representative" set of founded updates, called *universal*, that can be considered for query answering. Since the problem of checking whether a tuple belongs to the certain answers is undecidable, one way of restoring decidability of certain answering is to isolate a fragment of EAICs for which both the cardinality of $\mathbf{US}(D, \Sigma)$ and the cardinality of each element therein

is finite. Another approach is to identify a fragment for which certain answering becomes decidable, regardless of the finiteness of $\mathbf{US}(D, \Sigma)$ or the elements therein. In this work, we follow the first approach and leave the second one for future work.

The set $\mathbf{US}(D, \Sigma)$ may be infinite, i.e., it may contain updates of infinite size or its cardinality may be infinite. The following two examples show these two cases.

Example 11. Consider again the scenario of Example 9 where the database is $D = \{\mathtt{node(a)}\}$ and the set of EAICs Σ_9 is defined as follows:

$$\mathtt{edge(X,Y)} \wedge \nexists \mathtt{V}\ \mathtt{edge(Y,V)} \Rightarrow \exists \mathtt{Z}\ +\mathtt{edge(Y,Z)}.$$

In this case, $\mathbf{US}(D, \Sigma_9)$ consists of a single founded update $\mathcal{R} = \{+\mathtt{edge}(\mathtt{c}_2, \bot_1), +\mathtt{edge}(\bot_1, \bot_2), \dots\}$, which has an infinite number of update atoms. □

Example 12. Consider the following set of EAICs Σ_{12}:

$$\begin{array}{lll} \sigma_1 : & \mathtt{node(X)} \wedge \nexists \mathtt{Y}\ \mathtt{edge(X,Y)} & \Rightarrow \exists \mathtt{Z}\ +\mathtt{edge(X,Z)} \\ \sigma_2 : & \mathtt{edge(X,X)} & \Rightarrow -\mathtt{edge(X,X)} \end{array}$$

Notice that the second EAIC forbids self-loops to be in the database. For the database $D = \{\mathtt{node(a)}\}$, $\mathbf{US}(D, \Sigma)$ contains infinitely many founded updates of the form $\{+\mathtt{edge(a,c)}\}$ with $\mathtt{c} \in \mathcal{C} - \{\mathtt{a}\}$. □

The problem with Example 9 is that Σ_9 is "recursive" and there is a cyclic propagation and generation of nulls. Regarding Example 12, assuming that $\mathcal{C}$ is infinite, we have an infinite number of universal founded repairs, which are not finitely representable (with the formalism used in this paper). More in detail, there is an infinite number of updates $\{+\mathtt{edge(a,c)}\}$, where $\mathtt{c}$ is any constant in $\mathcal{C}$ different from constant $\mathtt{a}$, that cannot be represented by a unique universal repair $\{+\mathtt{edge}(\mathtt{a}, \bot)\}$, as $\bot$ represents all constants.

To guarantee finiteness of $\mathbf{US}(D, \Sigma)$ we need to introduce some restrictions on the form of EAICs.

In the following, we first introduce a restriction, called *safety*, that ensures that each founded update is finite. Then, we introduce a (somehow orthogonal) restriction, called *weak monotonicity*, ensuring a finite number of universal founded updates. As shown in the following, if a set Σ of EAICs satisfies both criteria, then finiteness of $\mathbf{US}(D, \Sigma)$ and of its elements is guaranteed for every database D.

The first restriction concerns the propagation of values among arguments. This problem is similar to the well-known problem of guaranteeing termination of the chase procedure (Fagin et al. (2005a,b); Deutsch et al. (2008); Greco et al. (2012)). We next present a restriction inspired by the safety criterion used for TGDs to guarantee chase termination (Meier et al. (2009)). The safety criterion analyzes the structure of EAICs by constructing a directed graph and analyzing how nulls are created and propagated into "affected" positions (Cali et al. (2013)).

Given a set of EAICs Σ, for every atom $p(t_1, ..., t_n)$ (or update atom $\pm p(t_1, ..., t_n)$) occurring in Σ and for all $i \in [1, n]$, $p[i]$ is called a *position* of Σ. Moreover, Σ^+

denotes the set of EAICs obtained from Σ by deleting all negative body literals and all head delete update atoms. If the head of σ contains only delete atoms, then σ is deleted.

Let Σ be a set of EAICs. The set of *affected positions* $\mathit{aff}(\Sigma)$ of Σ is defined iteratively starting from the empty set as follows. Let $p[i]$ be a position occurring in the head of some $\sigma \in \Sigma^+$ and X be a variable occurring in this position. Then $p[i]$ is added to $\mathit{aff}(\Sigma)$ if:

- X is existentially quantified, or
- X is universally quantified and appears in body atoms of σ only in affected positions of Σ.

Intuitively, affected positions are positions where nulls may be introduced. The following example further clarifies this aspect.

Example 13. Consider the EAIC of Example 9:

$$\texttt{edge}(\texttt{X},\texttt{Y}) \wedge \nexists \texttt{V}\ \texttt{edge}(\texttt{Y},\texttt{V}) \Rightarrow \exists \texttt{Z}\ +\texttt{edge}(\texttt{Y},\texttt{Z}).$$

There is only one binary predicate edge and, therefore, two positions: edge[1] and edge[2]. Position edge[2] is affected, as there is an edge-atom in the head with an existential variable in the second argument. This means that, in computing a universal repair, we could add an edge-atom with a null in the second argument and, thus, edge[2] is an affected position.

Moreover, if we have an edge-atom with a null in the second argument, to satisfy the EAIC we could need to add another edge-atom, where the null occurring in the second argument of the positive body literal is "copied" into the first argument of the new atom. Thus, position edge[1] is affected as well. Intuitively, this means that to compute a universal repair we could need to add atoms having nulls in both positions. □

As shown in Examples 11 and 13, the EAIC of Example 9 could produce a universal repair with an infinite number of nulls and an infinite number of atoms. This is due to the fact there is a cyclic propagation and generation of nulls. More specifically, having an atom with a null in the second argument, we add a new atom where the null is copied into the first argument and a new null is put in the second argument. This situation can be represented by means of a labelled graph. The next definition introduces the concept of propagation graph which allows us to identify these cyclic conditions.

The *propagation graph* of Σ is a directed graph $G_\Sigma = (V, E)$, where $V = \mathit{aff}(\Sigma)$ is the set of nodes of G_Σ and E is the set of edges defined as follows. For every $\sigma \in \Sigma^+$ and for every variable X occurring in a body atom in position $p[i]$, if X occurs only in affected positions in $\mathit{body}(\sigma)$ then:

- for every occurrence of X in $\mathit{head}(\sigma)$ in position $q[j]$ there is an edge $p[i] \rightarrow q[j]$ in E;
- for every existentially quantified variable Z in the head of σ and for every position $q[j]$ where Z occurs, there is a labeled edge $p[i] \stackrel{*}{\rightarrow} q[j]$.

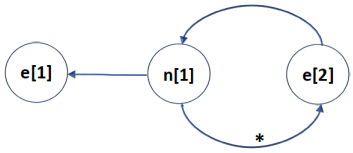

Figure 1: Propagation graph for Σ_{14}

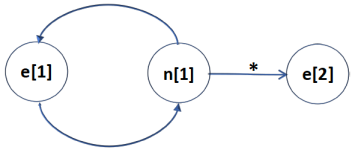

Figure 2: Propagation graph for Σ'_{14}

Definition 7 (Safe EAICs). A set of EAICs Σ is said to be *safe* if the propagation graph G_Σ has no cycles going through a labeled edge. □

Example 14. Consider the following set of EAICs Σ_{14}:

$$\sigma_1\colon\ \mathtt{node(X)} \wedge \nexists \mathtt{Y}\ \mathtt{edge(X,Y)} \ \Rightarrow \exists \mathtt{Z}\ +\mathtt{edge(X,Z)}$$
$$\sigma_2\colon\ \mathtt{edge(X,Y)} \wedge \neg\, \mathtt{node(Y)} \ \Rightarrow -\mathtt{edge(X,Y)} \vee +\mathtt{node(Y)}$$

The set of EAICs Σ^+_{14} is as follows:

$$\sigma'_1\colon\ \mathtt{node(X)} \ \Rightarrow \exists \mathtt{Z}\ +\mathtt{edge(X,Z)}$$
$$\sigma'_2\colon\ \mathtt{edge(X,Y)} \ \Rightarrow +\mathtt{node(Y)}$$

Position $\mathtt{edge}[2]$ is affected because variable $\mathtt{Z}$ is existentially quantified in that position in σ'_1. Moreover, position $\mathtt{node}[1]$ is affected as variable $\mathtt{Y}$ appears in the head and only in affected positions in the body of σ'_2. Consequently, from σ'_1 we derive that also position $\mathtt{edge}[1]$ is affected.

Thus, the vertices of the propagation graph are all positions of Σ_{14}. The propagation graph, shown in Figure 1, where predicate symbols node and edge have been replaced by their initials, contains the unlabeled edges $(\mathtt{n}[1], \mathtt{e}[1])$ and $(\mathtt{e}[2], \mathtt{n}[1])$, and the labeled edge $(\mathtt{n}[1], \mathtt{e}[2])$. This happens because (i) a null occurring in position $\mathtt{node}[1]$ may be copied to position $\mathtt{edge}[1]$ (edge $(\mathtt{n}[1], \mathtt{e}[1])$), (ii) a null occurring in position $\mathtt{edge}[2]$ may be copied to position $\mathtt{node}[1]$ (edge $(\mathtt{e}[2], \mathtt{n}[1])$), and (iii) a new null is introduced in position $\mathtt{edge}[2]$ which "depends" on the value in position $\mathtt{node}[1]$ (labeled edge from $\mathtt{n}[1]$ to $\mathtt{e}[2]$). As there is a cycle with a labeled edge, Σ_{14} is not safe—indeed, for the database $D = \{\mathtt{node(a)}\}$, there is a founded update with an infinite number of update atoms.

Observe that, by replacing $\mathtt{node(Y)}$ with $\mathtt{node(X)}$ in both the head and the body of σ_2, the new set of EAICs, denoted as Σ'_{14}, becomes safe as the propagation graph, shown in Figure 2, does not contain cycles with labeled edges. □

The second restriction regards the possibility to write EAICs with conflicting actions, and its aim is to guarantee a finite number of founded updates in $\mathbf{US}(D, \Sigma)$.

Definition 8 (Weakly monotonic EAICs). A set of EAICs Σ is *weakly monotonic* if there are no two update atoms $+A_1$ and $-A_2$ appearing in Σ and two substitutions ϑ_1 and ϑ_2 such that $\vartheta_1(A_1) = \vartheta_2(A_2)$. □

The weakly monotonic property guarantees that the sets of update atoms produced through the application of EAICs is always coherent, that is, there are no two EAICs which could add and delete two atoms having an homomorphism to the same ground atom with constants only. As an example, Σ_{14} is not weakly monotonic as the heads of the two EAICs contain "conflicting" updates.

As stated in the theorem below, by combining the safety and the weakly monotonicity restrictions, finiteness of $\mathbf{US}(D, \Sigma)$ is guaranteed, that is, $\mathbf{US}(D, \Sigma)$ contains a finite number of founded updates, and each of them is finite (i.e., it contains a finite number of update atoms). It is easy to see that both safety and weak monotonicity can be checked in polynomial time.

Theorem 2. *Let Σ be a set of safe, weakly monotonic EAICs. Then, for every database D, the set $\mathbf{US}(D, \Sigma)$ is finite, and each $\mathcal{R} \in \mathbf{US}(D, \Sigma)$ is finite.*

Proof. Nulls may be introduced only in arguments corresponding to positions such that there is path ending in such positions (in the propagation graph of Σ) and having a labeled edge. Since these positions do not occur in cycles with labeled edges, the number of labeled nulls that can be introduced is finite. As for the remaining positions, only constants (already occurring in the input database) can occur in arguments corresponding to those positions. Moreover, as EAICs are weakly monotonic, every null represents all constants in the domain, that is the possibly infinite number of constant atoms represented by a ground atom (possibly with nulls) is finitely representable.

To show that the size is polynomial it is sufficient to consider the skolemized version of Σ denoted by $sk(\Sigma)$, where existentially quantified variables occurring in the head of NEAICs and replaced by complex terms built using fresh function symbols and universally quantified variables, have as arguments only variables which can take values from constants occurring in the input database. As the number of function symbols and their arities depend only on Σ, for any database D, the size of the ground instantiation of $sk(\Sigma)$ obtained by replacing variables with constants in D, is polynomial in the size of the database domain $dom(D)$ and, therefore, in the size of the input database D. Moreover, since $|ground(sk(\Sigma))|$ is polynomial in $|D|$, the founded update atoms which can be inferred from $ground(sk(\Sigma))$ is also polynomial in the size of D. A founded update can be obtained by replacing ground complex terms with nulls. □

The next example shows a set of EAICs which are safe and weakly monotonic.

Example 15. Consider a database consisting of three relations with schemas $\mathtt{emp}(\mathtt{Name}, \mathtt{Dept})$, $\mathtt{dept}(\mathtt{Name}, \mathtt{Mgr})$ and $\mathtt{e\text{-}dept}(\mathtt{Name})$ storing information about employees (name and department where she/he works), departments (name and manager) and excellent departments[4] (name), and the below set of EAICs Σ:

$$\mathtt{emp}(\mathtt{E}, \mathtt{D}) \wedge \mathtt{e\text{-}dept}(\mathtt{D}) \wedge \nexists \mathtt{M}\, \mathtt{dept}(\mathtt{D}, \mathtt{M}) \Rightarrow \exists \mathtt{Z} + \mathtt{dept}(\mathtt{D}, \mathtt{Z})$$
$$\mathtt{dept}(\mathtt{D}, \mathtt{E}) \wedge \nexists \mathtt{Y}\, \mathtt{emp}(\mathtt{E}, \mathtt{Y}) \Rightarrow \exists \mathtt{Z} + \mathtt{emp}(\mathtt{E}, \mathtt{Z})$$

[4] In Italy some university departments have been classified as excellent for the quality of research.

stating that (i) for each employee E working in a department D which is also an excellent department there must be a related tuple in the department table and, (ii) for each department D with manager E there must be an employee with name E in relation emp. The set of EAICs is safe and weakly monotonic. Therefore, for any database instance D, $\mathbf{US}(D, \Sigma)$ is finite and consists only of finite updates. □

The problem of identifying classes of TGDs admitting finite universal solutions has been deeply investigated in recent years (see Fagin et al. (2005a)). More general (and complex) criteria guaranteeing finiteness of a universal set of founded updates could be defined and we reserve such a topic for further future investigation. Criteria recently studied guaranteeing chase termination for TGDs (Greco et al. (2015); Calautti et al. (2016)) may be the inspiration for more general conditions for finiteness of universal set of founded updates.

5. Normal EAICs

So far we have considered general (canonical) EAICs allowing body negation, head disjunction and head updates. We next consider *normal* EAICs (NEAICs), that is, EAICs whose head contains exactly one update atom, and compare the expressive power of EAICs and NEAICs. To this end we also introduce the definition of equivalence among sets of EAICs.

Definition 9. *Given two sets of EAICs Σ and Σ', we say that Σ and Σ' are* equivalent *if for every database D, $\mathbf{FR}(D, \Sigma) = \mathbf{FR}(D, \Sigma')$.* □

Clearly, for any pair of equivalent sets of EAICs Σ and Σ', database D and query Q, $\text{CERTAIN}(Q, D, \Sigma) = \text{CERTAIN}(Q, D, \Sigma')$.

Theorem 3. *The problem of deciding whether two sets of EAICs are equivalent is undecidable.*

Proof. The proof can be carried out by reduction from the *Query Equivalence Problem*: given two queries Q and Q' of the same arity, is it the case that Q and Q' are equivalent (i.e., does $Q(D) = Q'(D)$ hold on every database D)? For the sake of simplicity, we assume that the two queries are Boolean and defined by two sets of (safe, not recursive) Datalog rules, that is $Q = (g, P)$ (resp. $Q' = (g, P')$, where g is a predicate symbol of arity 0 (called query goal) and P (resp. P') is a set of Datalog rules (called program). Therefore $Q(D)$ is equivalent to checking whether g belongs to the minimal model M of $P \cup D$, which is equal to the least fixpoint of $P \cup D$. We show that checking whether $g \in M$ is equivalent to checking whether g belongs to the (unique) founded repair $\mathcal{R}$ of (Σ_P, D), where Σ_P is a set of AICs derived from P as follows. Let $\Sigma_P = \{body(X, Y) \wedge \neg a(X) \Rightarrow +a(X) \mid a(X) \leftarrow body(X, Y) \in P\}$, the minimal model M of $P \cup D$ coincides with the the (unique) founded repair $\mathcal{R}$ for $\Sigma_P \cup D$. This can be proved inductively showing that at each step i, $M_i = \mathcal{R}_i$, where M_i and $\mathcal{R}_i$ are, respectively, the fixpoints of $P \cup D$ and $\Sigma_P \cup D$ computed until step i:

- (base case - step 0). $M_0 = \mathcal{R}_0 = D$;
- (inductive case - step i). Assuming that $M_{i-1} = \mathcal{R}_{i-1}$, first assign $M_i = M_{i-1}$ and $\mathcal{R}_i = \mathcal{R}_{i-1}$ and then update M_i and $\mathcal{R}_i$ as follows. For each rule $a(X) \leftarrow body(X,Y)$ and for each (constant) matcher θ such that $M_i \models \theta(body(X))$ and $M_i \not\models \theta(a(X))$, add $\theta(a(X))$ to M_i. Moreover, this implies that there is an AIC $body(X,Y) \wedge \neg a(X) \Rightarrow +a(X)$ such that $\mathcal{R}_i \models \theta(body(X) \wedge \neg a(X))$ and, therefore, $\theta(a(X))$ is added to $\mathcal{R}_i$ as well. Clearly also the vice versa holds. Therefore $M_i = \mathcal{R}_i$.

Moreover, let $M = M_j$ such that $M_j = M_{j+1}$ and let $\mathcal{R} = \mathcal{R}_j$ such that $\mathcal{R}_j = \mathcal{R}_{j+1}$. We have that M is the minimal model of $P \cup D$ and $\mathcal{R}$ is the (unique) founded repair for (D, Σ). Clearly checking whether $g \in M$ is equivalent to checking whether there is a founded repair containing g. □

Given a set of EAICs Σ, $Normal(\Sigma)$ denotes the set of NEAICs derived from Σ by replacing every EAIC $B \Rightarrow A_1 \vee \cdots \vee A_n \in \Sigma$ with n NEAICs $B \Rightarrow A_i$, $i \in [1, n]$.

Theorem 4. *For any set of EAICs Σ, $Normal(\Sigma)$ and Σ are equivalent.*

Proof. Let $\Sigma' = Normal(\Sigma)$. First of all, observe that $\mathbf{R}(D, \Sigma) = \mathbf{R}(D, \Sigma')$ for every database D, as $St(\Sigma) = St(\Sigma')$. The fact that $\mathbf{FR}(D, \Sigma) = \mathbf{FR}(D, \Sigma')$ holds, follows from the fact that for any repair $\mathcal{R}$ for $\langle D, \Sigma \rangle$, $Normal(\Sigma[\mathcal{R}]) = (Normal(\Sigma))[\mathcal{R}]$. □

Thus, normal EAICs have the same expressivity of general EAICs and head disjunction is only used as a shorthand for several EAICs. We point out that by adding disjunction to standard TGDs we have an increment of the expressivity and complexity.

6. Deterministic EAICs

The framework studied so far is very general and highly expressive. This implies that the complexity of computing certain answers is, in the general case, high. Clearly, the complexity depends on the large number of updates and alternative repairs that must be considered. In practical cases, users would not necessarily need the full expressive power of EAICs, and rely on less expressive fragments. One of such fragments of EAICs is that for which the universal set of founded updates contains at most one update, that is, all repairs of a universal set are homomorphically equivalent. There are several reasons to study such a fragment: (i) to identify subsets of EAICs for which computing certain answers is tractable (polynomial time complexity), (ii) to effectively generate a repaired database and not only use repairs to compute certain answers, and (iii) to model ontological reasoning (see subsection 6.2).

Thus, in this section we study a subset of EAICs with a deterministic behavior, that is, such that $\mathbf{US}(D, \Sigma)$ has at most one founded update. As the problem of checking whether a set of EAICs is deterministic is undecidable (see next), we shall define a syntactic subclass of EAICs, called confluent, for which at most one universal repair

exists, for every input database. We will also show that this class is expressive enough to capture meaningful scenarios, and that every set of TGDs can be rewritten into an "equivalent" set of confluent NEAICs.

Let us start by introducing the formal definition of a deterministic set of EAICs.

Definition 10 (Deterministic EAICs). A set of EAICs Σ is *deterministic* if $|\mathbf{US}(D, \Sigma)| \leq 1$, for every database D. □

Therefore, for deterministic EAICs Σ, if $\langle D, \Sigma \rangle$ admits founded repairs, then there exists a founded repair $\mathcal{R}_i$ such that for any other founded repair $\mathcal{R}_j$, $\mathcal{R}_i \sqsupseteq \mathcal{R}_j$ holds. This also implies that in such a case positive queries can be answered by computing exactly one universal founded update (if it exists). The following example shows a set of deterministic EAICs which has no founded updates.

Example 16. Consider the set of EAICs Σ_5 of Example 5, which is shown below:

$$\begin{aligned} \sigma_1 &: \mathtt{p} \wedge \mathtt{q} \quad \Rightarrow -\mathtt{p} \\ \sigma_2 &: \mathtt{p} \wedge \neg\mathtt{q} \quad \Rightarrow +\mathtt{q} \end{aligned}$$

Since for every database D, $|\mathbf{US}(D, \Sigma_5)| \leq 1$, Σ_5 is deterministic. Indeed, if D does not contain $\mathtt{p}$, then there is only one founded update (namely the empty one, as D is consistent) and only one founded repair, namely D. If D contains both $\mathtt{p}$ and $\mathtt{q}$, then there is only one founded update (namely $-\mathtt{p}$) and thus only one founded repair. On the other hand, if D contains $\mathtt{p}$ but does not contain $\mathtt{q}$, there is no founded update and, therefore, no founded repair.

Consider now the set of EAICs Σ_{16} obtained from Σ_5 by replacing σ_2 with the following EAIC:

$$\sigma_3 : \mathtt{p} \wedge \mathtt{q} \Rightarrow -\mathtt{q}$$

We have that Σ_{16} is not deterministic anymore as for $D' = \{\mathtt{p}, \mathtt{q}\}$, $\mathbf{US}(D', \Sigma_{16}) = \{\{-\mathtt{p}\}, \{-\mathtt{q}\}\}$. □

As discussed before, the deterministic property for EAICs is a desirable one, that would allow us to obtain good computational behavior. It is thus crucial to understand what is the complexity of checking whether a given set of EAICs is deterministic.

Theorem 5. *Checking whether a set of EAICs Σ is deterministic is undecidable.* □

Proof. The proof can be carried out by reduction from the problem of checking whether a Datalog program P has a total well founded model or, equivalently, whether it has a unique stable model, for each database D.

Let P be a Datalog program, we denote by $\Sigma_P = \{body(\overline{X}, \overline{Y}) \wedge \neg a(\overline{X}) \Rightarrow +a(\overline{X}) \mid a(\overline{X}) \leftarrow body(\overline{X}\,\overline{Y}) \in P\}$ be the set of AICs derived from P. We prove that for every database D and stable model M for $P \cup D$, M is a founded repair for $\langle D, \Sigma_P \rangle$ and $\mathcal{R} = \{+a(\overline{x}) \mid a(x) \in M \setminus D\}$ is a founded update for $\langle D, \Sigma_P \rangle$.

Let $P' = ground(P)$, M a set of ground atoms, and $P'' = \{a(\overline{x}) \leftarrow body(\overline{x}, \overline{y}) \in P' \mid a(x) \in M \setminus D\}$. First, observe that M is a stable model for $P' \cup D$ iff it is a stable

model for $P'' \cup D$. This follows from the definition of stable model. In fact, let P'_M (resp. P''_M) be the positive program derived from P' (resp. P'') by *(i)* deleting rules having a negative body literal $\neg b(\overline{z})$ with $b(\overline{z}) \in M$ and *(ii)* deleting the remaining negative body literals. Then, M is a minimal model for $P'_M \cup D$ iff it is a minimal model for $P''_M \cup D$.

Consider the ground instantiations $P' = ground(P)$ and $\Sigma'_P = ground(\Sigma_P)$. We have that for each rule $a(\overline{x}) \leftarrow body(\overline{x}, \overline{y}) \in P'$ with $a(\overline{x}) \in M \setminus D$ there exists a ground AIC $body(\overline{x}, \overline{y}) \wedge \neg a(\overline{x}) \Rightarrow +a(\overline{x})$ such that $M \not\models body(\overline{x}, \overline{y}) \wedge \neg a(\overline{x})$. As M is minimal (recall that stable models are minimal models), it is a repair for $\langle D, \Sigma_{P'} \rangle$ and, consequently, a repair for $\langle D, \Sigma_P \rangle$.

Moreover, $M \not\models body(\sigma')$ for every $\sigma' \in \Sigma_{P'}$ implies that $M \not\models body(\sigma'')$ for every $\sigma'' \in \Sigma_{P''}$. As $\Sigma_{P''} = \Sigma_{P'}[\mathcal{R}]$, M is a repair for both $\langle D, \Sigma_{P'} \rangle$ and $\langle D, \Sigma_{P'}[\mathcal{R}] \rangle$, i.e. it is a founded repair for $\langle D, \Sigma \rangle$ □

6.1. Confluent EAICs

In light of the previous theorem, we introduce sufficient conditions ensuring that a set of EAICs is deterministic. To this end we introduce further restrictions as shown in Figure 3. In particular, while weak monotonicity guarantees that founded updates are coherent, the monotonicity property we are going to introduce adds a further restriction guaranteeing that mutually recursive NEAICs perform only additions of atoms or only deletions of atoms. On the other hand, the new confluence property we are going to introduce guarantees that updates generated by means of a fixpoint computation are founded and universal. Finally, we define the inner class $eaic(TGD)$ as the the class of EAICs corresponding to TGDs, which is included in the classes of EAICs discussed above. The roadmap for the rest of the section is to study the fragments shown in Figure 3 not considered so far.

A set of EAICs (possibly with disjunction in the head) intuitively describes different alternative ways to construct a repair. Hence, a first restriction might be to disallow disjunction in the head. However, as previously shown, NEAICs (which are disjunction-free) have the same expressivity of general EAICs. Hence, this only restriction will not be enough in order to obtain decidability. Thus, we focus our attention on normal EAICs enjoying further restrictions. Before proceeding with the formal definition of our syntactic restriction, we introduce some auxiliary notations and definitions.

An NEAIC σ is *applicable* (or *fireable*) w.r.t. a database D, if there exist a matcher ϑ (mapping only universally quantified variables of σ) and an homomorphism $h : \mathcal{N} \rightarrow \mathcal{C}$ such that $h(D) \models \vartheta(body(\sigma))$. Here, $h(D)$ is a possible instantiation of D, whereas $\vartheta(body(\sigma))$ is the body of a partially ground version of σ.

The application of a fireable NEAIC σ to a database D with matcher ϑ gives a new database $D' = \vartheta'(head(\sigma))(D)$, where ϑ' extends ϑ by mapping existential variables in $head(\sigma)$ with fresh distinct nulls not occurring in D, and is denoted by $D \xrightarrow{\sigma,\vartheta} D'$.

Given two (not necessarily distinct) NEAICs $\sigma_i, \sigma_j \in \Sigma$, we say that:

- σ_i *fires* σ_j if there exists a database D and a matcher ϑ_i such that: (i) $D \xrightarrow{\sigma_i,\vartheta_i} D_i$, (ii) σ_j is not fireable in D, but is fireable in D_i.

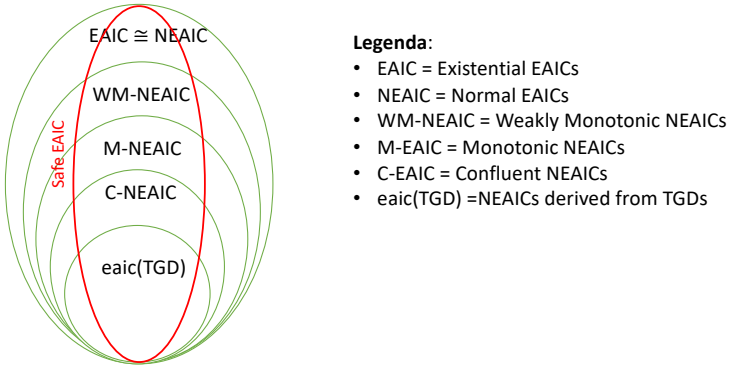

Figure 3: Restricted fragments of EAICs

- σ_i *blocks* σ_j if there exist two databases D_1 and D_2 and two matchers ϑ_i and ϑ_j such that: (i) $D_1 \xrightarrow{\sigma_i,\vartheta_i} D_1'$, (ii) $D_2 \xrightarrow{\sigma_j,\vartheta_j} D_2'$ and (iii) let $\pm A_k$ be the update atom such that $\pm A_k(D_k) = D_k'$ (with $k \in \{1,2\}$), then $\pm A_1(D_2) \not\models \vartheta_j(body(\sigma_2) \setminus CompLit(head(\sigma_2)))$.

Intuitively, the above definition states that σ_i fires σ_j if there are two partially ground instantiations of σ_i and σ_j, say σ_i' and σ_j', such that the firing of σ_i' modifies the database so that σ_j' becomes fireable. Analogously, we say that σ_i blocks σ_j if there are two partially ground instantiations of σ_i and σ_j, say σ_i' and σ_j', such that the firing of σ_i' performs an update which makes σ_j' (which was applicable before) not fireable because σ_i' made an alternative update making σ_j' satisfied. Here $\pm A_1(D_2) \not\models \vartheta(body(\sigma) \setminus CompLit(head(\sigma), \sigma))$ means that update $\pm A_1$ made false some literal in the body of σ_j' which is not made false by update $\pm A_2$, that is σ_2' has been made satisfied by the update performed by σ_1'.

Example 17. Consider the database $D = \{\texttt{node(a)}\}$ and the NEAICs of Example 12 shown below:

$$\begin{aligned} \sigma_1 &: \texttt{node(X)} \wedge \nexists \texttt{Y}\ \texttt{edge(X,Y)} && \Rightarrow \exists \texttt{Z}\ +\texttt{edge(X,Z)} \\ \sigma_2 &: \texttt{edge(X,X)} && \Rightarrow -\texttt{edge(X,X)} \end{aligned}$$

We have that σ_1 is applicable w.r.t. D by taking the substitution $\vartheta = \{\texttt{X/a}\}$ and an empty homomorphism h, whereas σ_2 is not applicable as D does not contain any edge-atom. The application of σ_1 to D with substitution ϑ produces $D' = \{\texttt{node(a)}, \texttt{edge(a,}\perp_1\texttt{)}\}$. As the NEAIC σ_2 was not fireable w.r.t. D, but it is fireable w.r.t. D' with substitution $\{\texttt{X/a}\}$ and homomorphism $h' = \{\perp_1 /a\}$, we have that σ_1 fires σ_2. Consider now the database $D_1 = \{\texttt{node(a)}, \texttt{edge(a,a)}\}$. NEAIC σ_2 is fireable w.r.t.

D_1, whereas σ_1 is not fireable. The application of σ_2 to D_1 produces a new database D_1' which makes σ_1 fireable. Thus σ_2 fires σ_1.

Regarding the set of NEAICs Σ_5 of Example 5, we have that σ_2 fires σ_1 (e.g., consider the database $D = \{p\}$) and σ_1 blocks σ_2 (e.g., consider the databases $D_1 = \{\mathtt{p}, \mathtt{q}\}$ and $D_2 = \{\mathtt{p}\}$). □

Definition 11 (Firing Graph). Given a set of NEAICs Σ, the *firing graph* of Σ is a labeled directed graph $\Gamma_\Sigma = (\Sigma, E)$ where the set of edges E is defined as follows:

$$E = \begin{array}{l} \{(\sigma_i, \sigma_j, sign(head(\sigma_i))) \mid \sigma_i, \sigma_j \in \Sigma \; and \; \sigma_i \text{ fires } \sigma_j\} \cup \\ \{(\sigma_i, \sigma_j, block) \mid \sigma_i, \sigma_j \in \Sigma \; and \; \sigma_i \text{ blocks } \sigma_j\}, \end{array}$$

with $sign(head(\sigma))$ being the type of update performed by σ, that is, $sign(+A) =$ "$+$", whereas $sign(-A) =$ "$-$". □

Thus, edges in the firing graph may have three different labels: "+", "–", and "block".

The next definition introduces the class of confluent NEAICs, which we will show to be deterministic.

Definition 12. A set of NEAICs Σ is said to be

1. *monotonic* if it is weakly monotonic and Γ_Σ does not contain any cycle with both insert and delete edges;
2. *confluent* if it is monotonic and Γ_Σ does not contain cycles with blocking edges (i.e., edges with label "block"). □

The monotonicity property guarantees that mutually recursive NEAICs perform only additions of atoms or only deletions of atoms. The confluence property guarantees that if a partially grounded EAIC σ_j has been applied producing an update $\pm A$, then the database will not be modified in the next steps so that σ_j is not applicable anymore making $\pm A$ unfounded.

Example 18. Considering the set of NEAICs Σ_5 of Example 5, we have that $\Gamma_{\Sigma_5} = \langle\{\sigma_1, \sigma_2\}, \{(\sigma_2, \sigma_1, +), (\sigma_1, \sigma_2, block)\}\rangle$. Since Γ_{Σ_5} is weakly monotonic and does not contain any cycle with two edges labeled respectively with "+" and "-", Σ_5 is monotonic. However, since the firing graph has a cycle with a blocking edge, the set of NEAICs is not confluent.

Regarding the set of NEAICs Σ_{12} of Example 12, we have that $\Gamma_{\Sigma_{12}} = \{\sigma_1, \sigma_2\}$, $\{(\sigma_1, \sigma_2, +), (\sigma_2, \sigma_1, -)\}\rangle$. Since the firing graph contains a cycle with both insert and delete edges, Σ_{12} is not monotonic as it neither is weakly monotonic nor satisfies Item 1 of Definition 12. Consequently Σ_{12} is not confluent. □

We now introduce a procedure to compute a universal set of founded updates for confluent NEAICs. For any database D and confluent set of NEAICs Σ, for each $i \geq 0$, we inductively define the database D_i as follows:

- $i = 0$: $D_i = D$;
- $i > 0$: D_i is defined as a database for which it holds $D_{i-1} \xrightarrow{\sigma,\vartheta} D_i$, for some arbitrary $\sigma \in \Sigma$ and matcher ϑ, such that there is no path in Γ_Σ from an $\sigma' \in \Sigma$ applicable in D_{i-1} to σ involving a blocking edge.

It is easy to see, from the definition of confluent NEAICs, that there always exists $n \geq 0$, such that $D_n = D_{n+1}$. We denote such a D_n as $fixpoint(D, \Sigma)$. Moreover, as we can have different alternative fixpoint sequences, we can have different final databases that, as shown in the next theorem, are all homomorphically equivalent and are all universal founded repairs. Observe that the fixpoint presented above is equivalent to partitioning Σ into strata so that the same stratum does not contain two NEAICs σ_i and σ_j such that there is a path with a blocking edge from σ_i to σ_j. and computing one stratum at a time following the topological order of Γ_Σ. Basically, it works similarly to the stratified fixpoint algorithm for Datalog with stratified negation.

Theorem 6. *For any database D and confluent set of NEAICs Σ, $fixpoint(D, \Sigma)$ is a universal founded repair.*

Proof. First we prove that, is such a case, D_n is a founded repair. This derives from the fact that at each step it is applied an NEAIC σ that is fireable with respect to the current database and a matcher ϑ. The application of σ with matcher ϑ gives a new instance $\vartheta(head(\sigma))(D)$ and the update performed is never contradicted in the next steps, that is, each positive literal in $\vartheta(body(\sigma))$ will not be deleted and for each negative body literal $\nexists \overline{Y} b(\overline{X}, \overline{Y})$ no atom $A \models \nexists \overline{Y}\, b(\vartheta(\overline{X}), \overline{Y})$ will be added. The update is never contradicted because there is no path in Γ_Σ from an $\sigma' \in \Sigma$ applicable in D_i to σ involving a blocking edge. This process continues while there is some NEAIC that is applicable with respect to the current database and a given matcher. At the end of the (possibly infinite) fixpoint sequence the final instance is consistent and, therefore, is a repair. Obviously, the repair is founded as it is derived from the application of NEAICs where each body literal will not be contradicted in the next steps.

In the following, given two instances I and J, we say that there exists a homomorphism $h : I \to J$ if there exists a homomorphism h s.t. $h(I) \subseteq J$.

To show that D_n is a universal founded repairs, that is for any founded update $\mathcal{R}$, $D_n \sqsupseteq \mathcal{R}(D)$, we first prove the following: let $I \xrightarrow{\sigma,\vartheta} J$ a repair step with $head(\sigma) = \exists \overline{Y}\ + a(\overline{X}, \overline{Y})$, and let K be an instance such that *(i)* $K \not\models body(\sigma)$ and *(ii)* there exists a homomorphism $h : I \to K$; then there exists a homomorphism $h' : J \to K$.

As $\vartheta : body(\sigma) \to I$ and $h : I \to K$ are homomorphisms, $h \circ \vartheta : body(\sigma) \to K$ is a homomorphism as well. Since K satisfies σ, there exists a homomorphism $h'' : body(\sigma) \to K$ such that h'' is an extension of $h \circ \vartheta$ mapping also head existential variables and $h''(X) = h(\vartheta(X))$ where X is universally quantified. For each existential variable Y occurring in the head, denote by $\perp_Y$ the fresh labeled null used to substitute variable Y at the current step (i.e. at step $I \xrightarrow{\sigma,\vartheta} J$) in the fixpoint procedure. Define h' on $\mathcal{N}(J)$ as follows: $h'(\perp_j) = h(\perp_j)$, if $\perp_j \in \mathcal{N}(I)$, and $h'(\perp_Y) = h''(\perp_Y)$ for each head existential variable Y at step i. We need to show that h' is a homomorphism

from J to K, which means that h' maps facts of J to corresponding facts of K. For facts of J that are also in I this is true because h is a homomorphism. Moreover J contains, in addition to any facts of I, a fact $a(\vartheta(\overline{X}), \mathcal{N}_{\overline{Y}})$, where $\mathcal{N}_{\overline{Y}}$ denotes a vector of labelled nulls, one for each variable in $\overline{Y}$. The image under h'' of this fact is, by definition of h'', the fact $a(h(\vartheta(\overline{X})), h''(\overline{Y}))$. Since $h''(\overline{X}) = h(\vartheta(\overline{X}))$, this is the same as $a(h''(\overline{x}), h''(\overline{Y}))$. But h'' homomorphically maps $a(\overline{X}, \overline{Y})$ into a fact of K. Thus, h' is a homomorphism.

Analogously we can prove the following: let $I \xrightarrow{\sigma,\vartheta} J$ with $head(\sigma) = -a(\overline{X})$, and let K be an instance such that: *(i)* $K \not\models body(\sigma)$ and *(ii)* there exists a homomorphism $h : I \to K \cup \{a(\overline{X})\}$; then there exists a homomorphism $h' : J \to K$.

For any repair $\mathcal{R}(D)$, by applying the above results at each fixpoint step, we obtain a homomorphism $h : D_n \to \mathcal{R}(D)$. Thus, D_n is a universal founded repairs. □

Thus, any confluent set admits a founded repair $\mathcal{R}$ which is universal, that is, for any other founded repair $\mathcal{R}'$, we have that $\mathcal{R} \sqsupseteq \mathcal{R}'$, and thus $|\mathbf{US}(D, \Sigma)| = 1$. It is worth noting that considering two applications of the fixpoint algorithm we can have two different repairing sequences $D = D_0 \xrightarrow{\sigma_1,\vartheta_1} D_1 \xrightarrow{\sigma_2,\vartheta_2} \cdots \xrightarrow{\sigma_n,\vartheta_n} D_n$ and $D = D'_0 \xrightarrow{\sigma'_1,\vartheta'_1} D'_1 \xrightarrow{\sigma'_2,\vartheta'_2} \cdots \xrightarrow{\sigma'_m,\vartheta'_m} D'_m$, but $D_n \equiv D'_m$. For computing certain answers to positive queries we can take any one of these repairs. The next example further clarifies this aspect.

Example 19. Consider the database $D = \{\mathtt{p(a)}, \mathtt{q(a)}\}$ and the confluent set of NEAICs Σ_{19}:

$$\sigma_1 : \mathtt{p(X)} \wedge \nexists \mathtt{e(X,Y)} \Rightarrow \exists \mathtt{Z} + \mathtt{e(X,Z)}$$
$$\sigma_2 : \mathtt{q(X)} \wedge \nexists \mathtt{e(X,X)} \Rightarrow \exists \mathtt{Z} + \mathtt{e(X,X)}$$

There are two possible fixpoint sequences $D = \{\mathtt{p(a)}, \mathtt{q(a)}\} \xrightarrow{\sigma_1,\{\mathtt{X}/\mathtt{a}\}} \{\mathtt{p(a)}, \mathtt{q(a)}, \mathtt{e(a,}\bot\mathtt{)}\} \xrightarrow{\sigma_2,\{\mathtt{X}/\mathtt{a}\}} \{\mathtt{p(a)}, \mathtt{q(a)}, \mathtt{e(a,}\bot\mathtt{)}, \mathtt{e(a,a)}\} = D_2$ and $D = \{\mathtt{p(a)}, \mathtt{q(a)}\} \xrightarrow{\sigma_2,\{\mathtt{X}/\mathtt{a}\}} \{\mathtt{p(a)}, \mathtt{q(a)}, \mathtt{e(a,a)}\} = D'_1$. Moreover, $D_2 \equiv D'_1$ and, for computing certain answers to positive queries, it makes no difference if we take D_2 or D'_1. □

The next proposition shows that for confluent AICs universal models coincide with grounded repairs introduced in Cruz-Filipe (2016a).

Proposition 2. *For any database D and set of AICs Σ, $fixpoint(D, \Sigma)$ is a grounded repair for $\langle D, \Sigma \rangle$.*

Proof. It is sufficient to show that our $fixpoint(D, \Sigma)$ coincides with the fixpoint of the operator $\mathcal{T}_\Sigma^D$ (denoted simply as $\mathcal{T}$ whenever D and Σ are understood) introduced by Cruz-Filipe (2014), applied iteratively to the input database D. Let us first recall how the operator $\mathcal{T}$ works. Initially (step 0), $\mathcal{U}_0 = \emptyset$. Then, at each step $i > 0$ the operator $\mathcal{T}$ selects nondeterministically a rule $\sigma \in ground(\Sigma)$ such that if $\mathcal{U}_{i-1}(D) \models body(\sigma)$ then $\mathcal{T}(\mathcal{U}_{i-1}) = \mathcal{U}_{i-1} \cup \{head(\sigma)\} = \mathcal{U}_i$, where $\mathcal{U}_{i-1} = \{\pm A_1, ..., \pm A_{i-1}\}$ is the set of updates performed at steps $j < i$. For confluent AICs the $\mathcal{T}$ operator is monotonic as it never generates conflicting updates.

To show the equivalence, assume first that $D_0 = D$ and $\mathcal{U}_0 = \emptyset$. This means that $D_0 = \mathcal{U}_0(D)$. Then, for each $i > 0$ consider the repair step $D_{i-1} \xrightarrow{\sigma_i, \vartheta_i} D_i$ performed by the fixpoint algorithm. Let $D_{i-1} = \mathcal{U}_{i-1}(D)$, then $\mathcal{U}_{i-1} \models body(\vartheta_i(\sigma))$, $D_i = head(\vartheta_i(\sigma_i))(D_{i-1})$ and $\mathcal{U}_i = \mathcal{U}_{i-1} \cup \{head(\vartheta_i(\sigma_i))\}$. The vice versa holds as well for every $i > 0$. □

Corollary 3. *For any database D and for any safe, confluent set of NEAICs Σ, a universal set of founded updates can be computed in polynomial time.*

Proof. It follows from Theorem 2 and the fact that every confluent set Σ is weakly monotonic. □

6.2. EAICs vs TGDs

The standard semantics for querying databases with data dependencies defined by TGDs and EGDs is based on (universal) solutions, that is, databases that contain the input one and can be homomorphically mapped to every other model of the input database together with the dependencies. The framework proposed in this paper is strictly more general than that based on TGDs, and the above holds not only because the semantics of EAICs allow to update the database through tuple insertion and deletion. Indeed, EAICs are more general than TGDs, even if we restrict EAICs to the confluent fragment allowing only insertions in the head. For instance, consider the constraint stating that employees not working for any department must be allocated in the IT department. This constraint can be expressed by the following EAIC:

$$\texttt{employee(X)}, \nexists \texttt{Y}\, \texttt{worksFor(X, Y)} \Rightarrow \texttt{worksFor(X, ``IT'')}$$

However, such a constraint cannot be expressed using TGDs and even normal TGDs as proposed in Alviano et al. (2017) (that is, TGDs with negated body literals and stable model semantics). Indeed, both TGDs and normal TGDs do not allow existentially quantified variables in the body of constraints. We will appropriately discuss the relationship with normal TGDs and stable model semantics.

We now establish a relationship between tuple-generating dependencies (TGDs)—see, e.g., Fagin et al. (2005a)—and confluent EAICs. Specifically, the latter (strictly) generalize the former. We start by stating how TGDs can be translated into NEAICs.

Definition 13. Given a TGD σ of the form

$$body(\overline{X}) \rightarrow \exists \overline{Y} a(\overline{X}', \overline{Y}),$$

where $\overline{X}' \subseteq \overline{X}$, we denote by $eaic(\sigma)$ the EAIC

$$body(\overline{X}) \wedge \nexists \overline{Y} a(\overline{X}', \overline{Y}) \Rightarrow \exists \overline{Z} + a(\overline{X}', \overline{Z}),$$

where $\overline{Z}$ is a list of new variables not occurring in σ. Given a set of TGDs Σ, we define $eaic(\Sigma) = \{eaic(\sigma) \mid \sigma \in \Sigma\}$. □

The next proposition states that the set of EAICs derived from a set of TGDs is confluent.

Proposition 3. *For every set of TGDs Σ, $eaic(\Sigma)$ is confluent.*

Proof. The set of EAICs $eaic(\Sigma)$ is trivially in normal form and monotonic as the head of every EAIC has only one update atom and such an update is an insert atom. We now show that Γ_Σ does not contain blocking edges. Assume that there are two NEAICs $\sigma_i, \sigma_j \in eaic(\Sigma)$ such that σ_i blocks σ_j. This means that there are $D_{i-1}, D_{j-1}, \vartheta_i, \vartheta_j$ such that $i)$ $D_{i-1} \xrightarrow{\sigma_i,\vartheta_i} D_i$ and that, let $+A_i$ be the update performed by rule σ_i with substitution ϑ_i (i.e. $\{A_i\} = D_i \setminus D_{i-1}$), $+A_i(D_{j-1}) \not\models \vartheta_j(body(\sigma_j) \setminus CompLit(head(\sigma_j)))$. Moreover, we know that $D_{j-1} \models body(\vartheta_j(\sigma_j))$. Since $\vartheta_j(body(\sigma_j) \setminus CompLit(head(\sigma_j)))$ contains only positive literals and derived from a TGD, and $D_{j-1} \models \vartheta_j(body(\sigma_j) \setminus CompLit(head(\sigma_j)))$, we have that $+A_i(D_{j-1}) = D_{j-1} \cup \{A_i\} \models \vartheta_j(body(\sigma_j) \setminus CompLit(head(\sigma_j)))$ as well. Thus, we have found a contradiction. Therefore, since Γ_Σ does not contain blocking edges, $eaic(\Sigma)$ is confluent. □

The following theorem states the "equivalence" between a set of TGDs Σ and the corresponding set of EAICs $eaic(\Sigma)$. Roughly speaking, for every universal model for a set of TGDs Σ and database D there is a homomorphically equivalent universal repair for the EAICs $\langle D, eaic(\Sigma)\rangle$.

Theorem 7. *Let Σ be a set of TGDs, D a database, and $M = \mathbf{US}(D, eaic(\Sigma))$. For every universal solution S of $\langle D, \Sigma\rangle$, S and M are homomorphically equivalent.*

Proof. Assume that S is a canonical universal solution, as for any other universal solution S' we have that $S \equiv S'$. This means that S is a fixpoint of the chase algorithm, that is, it can be obtained through a sequence of chase steps $D = D_0 \xrightarrow{\sigma_1,\vartheta_1} D_1 \xrightarrow{\sigma_2,\vartheta_2} \cdots \xrightarrow{\sigma_n,\vartheta_n} D_n \cdots$ and S be the fixpoint of this sequence. Moreover, since $eaic(\Sigma)$ is a set of confluent NEAICs, we can consider an $M = fixpoint(D, eaic(\Sigma))$. Thus, there is a repair sequence $D = D'_0 \xrightarrow{\sigma'_1,\vartheta'_1} D'_1 \xrightarrow{\sigma'_2,\vartheta'_2} \cdots \xrightarrow{\sigma'_n,\vartheta'_m} D'_m \cdots = M$. We can show that by taking $\sigma'_i = eaic(\sigma_i)$ and $\vartheta'_i = \vartheta_i$, we have that $D'_i \equiv D_i$ and, therefore, $n = m$ and $S \equiv M$.

- $i = 0$: $D'_0 = D_0$ by definition;
- $i > 0$: assuming $D'_{i-1} = D_{i-1}$ we can show that, by taking $\sigma'_i = eaic(\sigma_i)$ and $\vartheta'_i = \vartheta_i$, $D'_i \equiv D_i$ holds. Indeed the chase step $D_{i-1} \xrightarrow{\sigma_1,\vartheta_1} D_i$ implies that $D_{i-1} \models body(\vartheta_i(\sigma_i))$ and $D_{i-1} \not\models head(\vartheta_i(\sigma_i))$. However, $D'_{i-1} \models body(\vartheta'_i(\sigma'_i))$ and, let $head(\sigma_i) = \exists Z + a(X, Z)$, $D'_i = D'_{i-1} \cup \{a(\vartheta'_i(X), \bot_Z)\}$, where $\bot_Z$ is a vector of new nulls, one for each variable in Z. Moreover, we also have that $D_i = D_{i-1} \cup \{a(\vartheta'_i(X), \bot'_Z)\}$. Thus, $D'_i \equiv D_i$, i.e. D_i and D'_i differ only in the name of nulls. Thus, $D'_n \equiv D_n$, for every $n > 0$. □

Example 20. Consider the set Σ consisting of the TGD:

$$\texttt{employee}(\texttt{X}) \rightarrow \exists \texttt{Y}\ \texttt{worksFor}(\texttt{X}, \texttt{Y}).$$

The corresponding set of EAICs $eaic(\Sigma)$ contains the EAIC:

$$\texttt{employee}(\texttt{X}) \wedge \nexists \texttt{Y}\ \texttt{worksFor}(\texttt{X}, \texttt{Y}) \Rightarrow +\exists \texttt{Z}\ \texttt{worksFor}(\texttt{X}, \texttt{Z}).$$

Consider now the database $D = \{\texttt{employee}(\texttt{a})\}$. Notice that the pair $\langle D, \Sigma \rangle$ is said consistent in the ontology setting, while we say that $\langle D, eaic(\Sigma) \rangle$ is inconsistent. The universal solution of $\langle D, \Sigma \rangle$ is $S = \{\texttt{employee}(\texttt{a}), \texttt{worksFor}(\texttt{a}, \perp_{\texttt{i}})\}$, whereas $\mathbf{US}(D, eaic(\Sigma)) = \{+\texttt{worksFor}(\texttt{a}, \perp_{\texttt{j}})\}$, which applied to D yields the repair $M = \{\texttt{employee}(\texttt{a}), \texttt{worksFor}(\texttt{a}, \perp_{\texttt{j}})\}$. Consequently, M is homomorphically equivalent to S. □

It is worth noting that, since universal models of Σ are homomorphically equivalent to the repairs obtained from the universal set of founded updates of $eaic(\Sigma)$, equivalence w.r.t. query answering is preserved as well, as the query answers are the same in both settings (this is an immediate corollary of Theorem 7).

Moreover confluent EAICs strictly generalize TGDs. First, the former allow facts to be deleted, which is not the case for TGDs. Even if we focus on confluent EAICs containing only insert operations, they strictly generalize TGDs, as shown in the following example.

Example 21. Consider the database schema of Example 8 and the NEAIC of Example 7 reported below[5]:

$$\texttt{node}(\texttt{X}) \wedge \nexists \texttt{Y}\ \texttt{edge}(\texttt{X}, \texttt{Y}) \Rightarrow +\texttt{edge}(\texttt{X}, \texttt{X})$$

stating that for each node without an outgoing edge we must add a loop edge. Considering the database $D_{21} = \{\texttt{node}(\texttt{a})\}$, we have that $\mathbf{US}(D_{21}, \Sigma_{21})) = \{+\texttt{edge}(\texttt{a}, \texttt{a})\}$.

We point out that considering the stable model semantics proposed in Alviano et al. (2017) for TGDs extended with body negation, the semantics proposed in that work does not capture the meaning here proposed. Indeed, consider the corresponding set of TGDs extended with negation Σ_{21}:

$$\texttt{node}(\texttt{X}) \wedge \neg\texttt{edge1}(\texttt{X}) \rightarrow \texttt{edge}(\texttt{X}, \texttt{X})$$
$$\texttt{edge}(\texttt{X}, \texttt{Y}) \rightarrow \texttt{edge1}(\texttt{X})$$

where the EAIC above has been rewritten into two TGDs just to make range restricted variables in negative body literals. Then, $\Sigma_{21} \cup D_{21}$ has no stable model and, therefore, there is no way to make the database consistent. □

[5]This EAICs is safe, weakly monotonic and also confluent, as the firing graph does not contains edges.

7. Related Work

Automatic Maintenance of Integrity Constraints. The notion of automatic consistency maintenance in the presence of integrity constraints has been extensively considered in the context of database management systems. Many approaches proposed in the literature make use of ECA (event-condition-action) rules for checking and enforcing integrity constraints. In addition, current DBMS languages offer the possibility of defining triggers (special ECA rules) well suited to respond automatically, by performing actions, to events that are taking place inside (or even outside) the database. The effort of "adding" active rules into conventional database systems has raised considerable interest both in the scientific community and in the commercial world. As a consequence, a number of prototypes and systems have been developed (Widom & Ceri (1996)). However, the problem with active rules is the difficulty to understand the behaviour when a significant number of triggers act simultaneously (May & Ludascher (2002); Paton & Diaz (1999); Ceri et al. (2000)).

In Ceri & Widom (1990) and Ceri et al. (1994) a framework for database maintenance enforcing constraints by issuing actions to be performed to correct violations has been proposed, while the problem of maintaining integrity constraints in database systems has been considered in Medeiros & Andrade (1994).

A declarative framework for updating views over databases has been proposed in Caroprese et al. (2012). This work introduces the concepts of *constrained updates*. Constrained updates fulfill the view-update request, changing the database minimally and avoiding arbitrary commitments. A similar approach has been presented for abductive logic programming extended with integrity constraints in Caroprese et al. (2014). This work introduces the concept of *constrained explanation* for an observation as an explanation having no arbitrariness.

Consistent Query Answering. Querying inconsistent data and knowledge bases has been deeply investigated in the areas of databases and artificial intelligence (Arenas et al. (1999); Subrahmanian (1994)).

The notions of *repair* and *consistent query answer* have been introduced in Arenas et al. (1999). The same work has also proposed a method, based on query rewriting, to compute consistent query answers. The proposed technique is simple, but has a limited applicability (first-order queries without disjunction or quantification, and binary universal integrity constraints).

Several works have considered the use of logic programs to capture repairs as answer sets of logic programs with negation and disjunction (Arenas et al. (2003); Greco & Zumpano (2000); Greco et al. (2003); Furfaro et al. (2007)), so that consistent query answers can be found by resorting to answer set solvers (Gebser et al. (2012b,a); Leone et al. (2006); Greco et al. (2010)). These approaches are quite general, being able to handle arbitrary universal constraints and first-order queries.

For surveys on repairing and querying inconsistent databases we refer to Bertossi (2006); Chomicki (2007); Bertossi (2011).

Recently, Arioua & Bonifati (2018) has proposed an update-based repairing technique in the presence of both contradiction-detecting dependencies, a subset of denial

constraints capturing contradictions in the data, and tuple-generating dependencies. In such a framework repairs are suggested by the user and are computed through an interactive framework letting the user repair the knowledge base and meet her requirements.

Active integrity constraints. The formalism of AICs was first proposed in Flesca et al. (2004), although the formal declarative semantics has been defined in Caroprese et al. (2006, 2009). It is worth noting that Caroprese et al. (2009) introduced also a class of AICs, called *general active integrity constraints* (GAICs), allowing existentially quantified variables only in the body. However, as stated in Theorem 4 of Caroprese et al. (2009), GAICs do not add expressive power to AICs, as they can be rewritten into equivalent AICs.

Different problems around AICs have been investigated in the last years. Algorithms for computing all classes of repairs discussed in Caroprese & Truszczynski (2011) have been proposed in Cruz-Filipe et al. (2013)—see Cruz-Filipe et al. (2015) for an implementation over SQL databases. Cruz-Filipe et al. (2018) proposed a framework for applying AICs to more knowledge representation systems, other than databases, generalizing also the approach in Rantsoudis et al. (2017a) (see also Feuillade et al. (2019)). The computation of repairs is also related to the computation of grounded fixpoints, which have been shown to be the natural semantics in many knowledge representation formalisms (Bogaerts et al. (2015)). Furthermore, Cruz-Filipe (2014) proposed a notion of stratification of AICs, also discussed in Cruz-Filipe (2016a), that bears resemblance to the notion of an EAIC firing or blocking another. Alternative semantics, identifying more restricted classes of repairs, have been proposed in Caroprese & Truszczynski (2011). Recently, different algorithms for the computation of repairs have been proposed in Cruz-Filipe (2016b). A new class of semantics for AICs that are natural counterparts of existing semantics in various non-monotonic reasoning domains has been proposed in Bogaerts & Cruz-Filipe (2017, 2018). Existentially quantified variables in the body have been studied in Caroprese et al. (2009); the novelty of this paper is that we extend AICs by also considering head existentially quantified variables. A detailed comparison between the AICs framework and Revision Programming has been presented in Caroprese & Truszczynski (2011). Rantsoudis et al. (2017b) explored the idea underlying AICs in the description logic setting.

Existential rules. There has been a lot of interest in constraints expressed by means of tuple-generating dependencies (TGDs) (Fagin et al. (2005a)). As stated in Theorem 7, EAICs generalize TGDs—indeed, it suffices to consider the class of confluent EAICs to subsume the formalism of TGDs. In contrast to TGDs, EAICs allow negation in the body and deletion of facts. Indeed, the EAIC of Example 21 cannot be expressed even with the formalism considered in Alviano et al. (2017), which allows negation in the body of TGDs. In fact, Alviano et al. (2017) allows only a restricted form of negation, called *safe*, which imposes every variable in the body to appear in a positive literal.

Several Datalog-like languages with existential variables in the head have been recently proposed, e.g., Datalog$\pm$, *TriQ-Lite*, and *Wardalog* (Cali et al. (2012); Arenas et al. (2014b); Gottlob et al. (2014); Gottlob & Pieris (2015); Bellomarini et al. (2017)). Reasoning in the presence of inconsistent knowledge bases expressed by means of Datalog$\pm$ fragments has been addressed as well (Lukasiewicz et al. (2018)).

The main difference with respect to EAICs is that these languages do not sepa-

rate constraints from actions to be performed and some of them allow only stratified negation. We point out that EAICs allow us to express classical ontological axioms such as concept inclusion (e.g., $\texttt{emp} \sqsubseteq \texttt{person}$ can be expressed as $\texttt{emp}(\texttt{X}) \wedge \neg\texttt{person}(\texttt{X}) \Rightarrow +\texttt{person}(\texttt{X})$) and participation (e.g., $\texttt{emp} \sqsubseteq \exists\texttt{worksfor}$ can be expressed as $\texttt{emp}(\texttt{X}) \wedge \nexists \texttt{Y}\, \texttt{worksfor}(\texttt{X}, \texttt{Y}) \Rightarrow \exists \texttt{Z} + \texttt{worksfor}(\texttt{X}, \texttt{Z})$). Negative constraints can be expressed by means of EAICs with empty heads, whereas equality-generating dependencies can be expressed by simply allowing built-in atoms (e.g., $X \neq Y$) in the body of EAICs, as proposed for AICs (see Caroprese et al. (2009)). Thus, a constraint like the one previously mentioned cannot be expressed in these languages.

Universal repairs for databases with TGDs have been introduced in ten Cate et al. (2012), where universality is defined w.r.t. the query. Universal sets of founded repairs introduced in this paper are independent from the query and are defined in the context of EAICs.

Applications. The framework proposed in this paper finds application in several domains from different fields, including knowledge representation and reasoning (in the form of rules to express ontologies in ontology-mediated query answering) and databases (in the form integrity constraints for applications like data integration, data cleaning, and data exchange).

Ontology-mediated query answering (OMQA) is a prominent paradigm in knowledge representation where a set of logical assertions (ontology), usually in the form of existential rules, and a (conjunctive) query are combined together to obtain rich answers from standard databases. Inconsistency between the database and the ontology can naturally and very often arise, due to the open nature of OMQA systems. Such inconsistency is usually identified via dedicated assertions (denial constraints) of the ontology. In this context, maximally consistent versions of the database (repairs) are used to provide meaningful answers to queries via certain answers over all repairs. In this context, EAICs form a powerful formalism for defining richer constraints, that not only generalize denial constraints, but also allow to specify which of the assertions in the EAIC that cause the inconsistency are more desirable to not be entailed. This can become quite useful in such settings, where the users of the ontology, which are usually non-experts of database-related topics, are able to express preferences on how the database should be repaired by only using the higher level vocabulary of the ontology.

Data exchange is the process of taking data structured under a source schema and transforming it into data structured under a target schema (Fagin et al. (2005a,b); Arenas et al. (2014a)). In a such framework, EAICs can be used in place of TGDs obtaining a more general and flexible framework, without renouncing to consider restricted schemas with lower complexity. Moreover, EAICs can also be used to express *Equality Generating Dependencies* (EGDs), the other form of dependencies used in data exchange, but even more general constraints such as negative constraints.

Furthermore, EAICs can be used in other scenarios, such as production (rule) systems, active databases and data integration. For instance, in production systems, consisting of programs with reasoning features implemented by means of logical rules, EAICs can be used to implement mechanisms to respond to states of the world. Therefore, as EAICs generalize classical production rules, they can be used as a basic representation mechanism useful in different contexts such as automated planning, expert

systems and action selection. Regarding data integration, EAICs could be used to implement, using a unified framework for both dataset merging and cleaning mechanisms.

8. Conclusion

We have introduced existential active integrity constraints, a powerful formalism augmenting AICs with existentially quantified variables, and allowing users to express several constraints commonly arising in practice that cannot be expressed by means of AICs. The introduction of existential variables can make the number of founded updates as well as their cardinality infinite. We have shown that for the purpose of query answering, only a representative set of founded updates, called universal, need to be considered. Still, the universal set of founded updates can be infinite, so we have defined restrictions guaranteeing finiteness and a "deterministic" behavior, that is, the existence of at most one representative founded update.

References

Alviano, M., Morak, M., & Pieris, A. (2017). Stable model semantics for tuple-generating dependencies revisited. In *Proc. of the Symposium on Principles of Database Systems (PODS)* (pp. 377–388).

Arenas, M., Barceló, P., Libkin, L., & Murlak, F. (2014a). *Foundations of Data Exchange*. Cambridge University Press.

Arenas, M., Bertossi, L. E., & Chomicki, J. (1999). Consistent query answers in inconsistent databases. In *Proc. of the Symposium on Principles of Database Systems (PODS)* (pp. 68–79).

Arenas, M., Bertossi, L. E., & Chomicki, J. (2003). Answer sets for consistent query answering in inconsistent databases. *Theory and Practice of Logic Programming*, *3*, 393–424.

Arenas, M., Gottlob, G., & Pieris, A. (2014b). Expressive languages for querying the semantic web. In *Proc. of the Symposium on Principles of Database Systems (PODS)* (pp. 14–26).

Arioua, A., & Bonifati, A. (2018). User-guided repairing of inconsistent knowledge bases. In *Proc. of the International Conference on Extending Database Technology (EDBT)* (pp. 133–144).

Bellomarini, L., Gottlob, G., Pieris, A., & Sallinger, E. (2017). Swift logic for big data and knowledge graphs. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 2–10).

Bertossi, L. E. (2006). Consistent query answering in databases. *SIGMOD Record*, *35*, 68–76.

Bertossi, L. E. (2011). *Database Repairing and Consistent Query Answering*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers.

Bogaerts, B., & Cruz-Filipe, L. (2017). Semantics for active integrity constraints using approximation fixpoint theory. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 866–872).

Bogaerts, B., & Cruz-Filipe, L. (2018). Fixpoint semantics for active integrity constraints. *Artificial Intelligence*, *255*, 43–70.

Bogaerts, B., Vennekens, J., & Denecker, M. (2015). Grounded fixpoints and their applications in knowledge representation. *Artif. Intell.*, *224*, 51–71.

Calautti, M., Greco, S., Molinaro, C., & Trubitsyna, I. (2016). Exploiting equality generating dependencies in checking chase termination. *Proceedings of the VLDB Endowment*, *9*, 396–407.

Cali, A., Gottlob, G., & Kifer, M. (2013). Taming the infinite chase: Query answering under expressive relational constraints. *Journal of Artificial Intelligence Research*, *48*, 115–174.

Cali, A., Gottlob, G., & Lukasiewicz, T. (2012). A general datalog-based framework for tractable query answering over ontologies. *Journal of Web Semantics*, *14*, 57–83.

Caroprese, L., Greco, S., Sirangelo, C., & Zumpano, E. (2006). Declarative semantics of production rules for integrity maintenance. In *Proc. of the International Conference on Logic Programming, (ICLP)* (pp. 26–40).

Caroprese, L., Greco, S., & Zumpano, E. (2009). Active integrity constraints for database consistency maintenance. *IEEE Transactions on Knowledge and Data Engineering*, *21*, 1042–1058.

Caroprese, L., Trubitsyna, I., Truszczynski, M., & Zumpano, E. (2012). The view-update problem for indefinite databases. In *Proc. of the European Conference on Logics in Artificial Intelligence (JELIA)* (pp. 134–146).

Caroprese, L., Trubitsyna, I., Truszczynski, M., & Zumpano, E. (2014). A measure of arbitrariness in abductive explanations. *Theory and Practice of Logic Programming*, *14*, 665–679.

Caroprese, L., & Truszczynski, M. (2011). Active integrity constraints and revision programming. *Theory and Practice of Logic Programming*, *11*, 905–952.

ten Cate, B., Fontaine, G., & Kolaitis, P. G. (2012). On the data complexity of consistent query answering. In *Proc. of the International Conference on Database Theory (ICDT)* (pp. 22–33).

Ceri, S., Cochrane, R., & Widom, J. (2000). Practical applications of triggers and constraints: Success and lingering issues (10-year award). In *Proc. of the International Conference on Very Large Data Bases (VLDB)* (pp. 254–262).

Ceri, S., Fraternali, P., Paraboschi, S., & Tanca, L. (1994). Automatic generation of production rules for integrity maintenance. *ACM Transactions on Database Systems*, *19*, 367–422.

Ceri, S., & Widom, J. (1990). Deriving production rules for constraint maintainance. In *Proc. of the International Conference on Very Large Data Bases (VLDB)* (pp. 566–577).

Chomicki, J. (2007). Consistent query answering: Five easy pieces. In *Proc. of the International Conference on Database Theory (ICDT)* (pp. 1–17).

Cruz-Filipe, L. (2014). Optimizing computation of repairs from active integrity constraints. In C. Beierle, & C. Meghini (Eds.), *FoIKS* (pp. 361–380). Springer volume 8367.

Cruz-Filipe, L. (2016a). Grounded fixpoints and active integrity constraints. In M. Carro, A. King, N. Saeedloei, & M. D. Vos (Eds.), *ICLP* (pp. 11:1–11:14). volume 52.

Cruz-Filipe, L. (2016b). Grounded fixpoints and active integrity constraints. In *Technical Communications of the International Conference on Logic Programming (ICLP)* (pp. 11:1–11:14).

Cruz-Filipe, L., Franz, M., Hakhverdyan, A., Ludovico, M., Nunes, I., & Schneider-Kamp, P. (2015). repairc: A tool for ensuring data consistency. In *KMIS* (pp. 17–26).

Cruz-Filipe, L., Gaspar, G., Engrácia, P., & Nunes, I. (2013). Computing repairs from active integrity constraints. In *TASE* (pp. 183–190). IEEE Computer Society.

Cruz-Filipe, L., Gaspar, G., Nunes, I., & Schneider-Kamp, P. (2018). Active integrity constraints for general-purpose knowledge bases. *Ann. Math. Artif. Intell.*, *83*, 213–246.

Deutsch, A., Nash, A., & Remmel, J. B. (2008). The chase revisited. In *Proc. of the Symposium on Principles of Database Systems (PODS)* (pp. 149–158).

Fagin, R., Kolaitis, P. G., Miller, R. J., & Popa, L. (2005a). Data exchange: semantics and query answering. *Theoretical Computer Science*, *336*, 89–124.

Fagin, R., Kolaitis, P. G., & Popa, L. (2005b). Data exchange: getting to the core. *ACM Transactions on Database Systems*, *30*, 174–210.

Feuillade, G., Herzig, A., & Rantsoudis, C. (2019). A dynamic logic account of active integrity constraints. *Fundam. Inform.*, *169*, 179–210.

Flesca, S., Furfaro, F., & Parisi, F. (2010). Range-consistent answers of aggregate queries under aggregate constraints. In *Proc. of the International Conference on Scalable Uncertainty Management (SUM)* (pp. 163–176).

Flesca, S., Greco, S., & Zumpano, E. (2004). Active integrity constraints. In *Proc. of the ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP)* (pp. 98–107).

Furfaro, F., Greco, S., & Molinaro, C. (2007). A three-valued semantics for querying and repairing inconsistent databases. *Annals of Mathematics and Artificial Intelligence*, *51*, 167–193.

Gebser, M., Kaminski, R., Kaufmann, B., & Schaub, T. (2012a). *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Gebser, M., Kaufmann, B., & Schaub, T. (2012b). Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, *187*, 52–89.

Geerts, F., Mecca, G., Papotti, P., & Santoro, D. (2013). The LLUNATIC data-cleaning framework. *Proceedings of the VLDB Endowment*, *6*, 625–636.

Geerts, F., Mecca, G., Papotti, P., & Santoro, D. (2014). Mapping and cleaning. In *Proc. of the IEEE International Conference on Data Engineering (ICDE)* (pp. 232–243).

Gottlob, G., Lukasiewicz, T., & Pieris, A. (2014). Datalog+/-: Questions and answers. In *Proc. of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*.

Gottlob, G., & Pieris, A. (2015). Beyond SPARQL under OWL 2 QL entailment regime: Rules to the rescue. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 2999–3007).

Greco, G., Greco, S., & Zumpano, E. (2003). A logical framework for querying and repairing inconsistent databases. *IEEE Transactions on Knowledge and Data Engineering*, *15*, 1389–1408.

Greco, S., & Molinaro, C. (2008). Approximate probabilistic query answering over inconsistent databases. In *Proc. of the International Conference on Conceptual Modeling (ER)* (pp. 311–325).

Greco, S., & Molinaro, C. (2012). Probabilistic query answering over inconsistent databases. *Annals of Mathematics and Artificial Intelligence*, *64*, 185–207.

Greco, S., Molinaro, C., & Spezzano, F. (2012). *Incomplete Data and Data Dependencies in Relational Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers.

Greco, S., Molinaro, C., & Trubitsyna, I. (2018). Computing approximate query answers over inconsistent knowledge bases. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 1838–1846).

Greco, S., Molinaro, C., Trubitsyna, I., & Zumpano, E. (2010). NP datalog: A logic language for expressing search and optimization problems. *Theory and Practice of Logic Programming*, *10*, 125–166.

Greco, S., Spezzano, F., & Trubitsyna, I. (2015). Checking chase termination: Cyclicity analysis and rewriting techniques. *IEEE Transactions on Knowledge and Data Engineering*, *27*, 621–635.

Greco, S., & Zumpano, E. (2000). Querying inconsistent databases. In *Proc. of the International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)* (pp. 308–325).

Imielinski, T., & Lipski Jr., W. (1984). Incomplete information in relational databases. *Journal of the ACM*, *31*, 761–791.

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., & Scarcello, F. (2006). The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, *7*, 499–562.

Lukasiewicz, T., Malizia, E., & Molinaro, C. (2018). Complexity of approximate query answering under inconsistency in datalog+/-. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 1921–1927).

May, W., & Ludascher, B. (2002). Understanding the global semantics of referential actions using logic rules. *ACM Transactions on Database Systems*, *27*, 343–397.

Medeiros, C. B., & Andrade, M. J. (1994). Implementing integrity control in active data bases. *Journal of Systems and Software*, *27*, 171–181.

Meier, M., Schmidt, M., & Lausen, G. (2009). On chase termination beyond stratification. *Proc. of the International Conference on Very Large Data Bases (VLDB)*, *2*, 970–981.

Paton, N. W., & Diaz, O. (1999). Active database systems. *ACM Computing Surveys*, *31*, 63–103.

Rantsoudis, C., Feuillade, G., & Herzig, A. (2017a). Repairing aboxes through active integrity constraints. In A. Artale, B. Glimm, & R. Kontchakov (Eds.), *DL*. CEUR-WS.org volume 1879 of *CEUR Workshop Proceedings*.

Rantsoudis, C., Feuillade, G., & Herzig, A. (2017b). Repairing aboxes through active integrity constraints. In *Proc. of the International Workshop on Description Logics (DL)*.

Subrahmanian, V. S. (1994). Amalgamating knowledge bases. *ACM Transactions on Database Systems*, *19*, 291–331.

Widom, J., & Ceri, S. (Eds.) (1996). *Active Database Systems: Triggers and Rules For Advanced Database Processing*. Morgan Kaufmann.

Wijsen, J. (2005). Database repairing using updates. *ACM Transactions on Database Systems*, *30*, 722–768.

9. Appendix

Theorem 8. *For every database D and set of AICs Σ, every founded update for $\langle D, \Sigma\rangle$ is also founded according to the definition given in Caroprese et al. (2009).*

Proof. In the following, founded updates according to the definition given by Caroprese et al. (2009) are called *CGZ-founded* updates, while founded updates according to our definition are simply called founded updates.

Let $\mathcal{R}$ be a founded update for $\langle D, \Sigma\rangle$.

Reasoning by contradiction, suppose that $\mathcal{R}$ is not a CGZ-founded update for $\langle D, \Sigma\rangle$. Then, by definition of CGZ-founded update, there is a non-founded update atom $\pm A$ in $\mathcal{R}$. Let $\mathcal{R}' = \mathcal{R} \setminus \{\pm A\}$. By definition of non-founded update atom, there is no AIC $\sigma \in \mathit{ground}(\Sigma)$ s.t. $\pm A \in \mathit{head}(\sigma)$ and $\mathcal{R}'(D) \models \mathit{body}(\sigma)$.

Consider an arbitrary (ground) AIC $\sigma \in \Sigma[\mathcal{R}]$. Two cases can occur:

1. $\pm A \notin \mathit{head}(\sigma)$. Then, by definition of $\Sigma[\mathcal{R}]$, it has to be the case that $\mathit{head}(\sigma)$ contains a ground update atom $\pm B \neq \pm A$ that belongs to $\mathcal{R}$. Since $\pm B \neq \pm A$ and $\mathcal{R}' = \mathcal{R} \setminus \{\pm A\}$, then $\pm B$ belongs to $\mathcal{R}'$ too. Thus, $\mathcal{R}'(D) \not\models \mathit{body}(\sigma)$ by condition *(v)* of Definition 1, that is, $\mathcal{R}'(D)$ satisfies σ.
2. $\pm A \in \mathit{head}(\sigma)$. Two (sub)cases can possibly occur:
   (a) $\mathcal{R}'(D) \models \mathit{body}(\sigma)$. By definition of $\Sigma[\mathcal{R}]$ there must be a ground AIC $\sigma' \in \mathit{ground}(\Sigma)$ such that $\mathit{body}(\sigma') = \mathit{body}(\sigma)$—indeed, σ has been derived from σ' (by possibly deleting some ground update atoms from the head of σ'). Then, $\mathcal{R}'(D) \models \mathit{body}(\sigma')$. This contradicts the fact that $\pm A$ is not founded: notice that σ' belongs to $\mathit{ground}(\Sigma)$, it has $\pm A$ in the head, and $\mathcal{R}'(D) \models \mathit{body}(\sigma')$, so this case cannot actually occur.
   (b) $\mathcal{R}'(D) \not\models \mathit{body}(\sigma)$. Thus, $\mathcal{R}'(D)$ satisfies σ.

From the analysis above, $\mathcal{R}'(D)$ satisfies every $\sigma \in \Sigma[\mathcal{R}]$, which means that $\mathcal{R}$ is not an update for $\langle D, \Sigma[\mathcal{R}]\rangle$, as it is not minimal. Hence, $\mathcal{R}$ is not founded for $\langle D, \Sigma\rangle$, which contradicts the initial hypothesis. □

The next example shows that the opposite case does not hold.

Example 22. Consider the database $D = \{\mathtt{a}, \mathtt{b}\}$ and the set of AICs Σ_{22}

$$\begin{aligned} \mathtt{a} \wedge \neg \mathtt{b} &\Rightarrow -\mathtt{a} \\ \mathtt{b} \wedge \neg \mathtt{a} &\Rightarrow -\mathtt{b} \\ \mathtt{a} \wedge \neg \mathtt{c} &\Rightarrow +\mathtt{c} \\ \mathtt{b} \wedge \neg \mathtt{c} &\Rightarrow +\mathtt{c} \end{aligned}$$

There are two updates for $\langle D, \Sigma_{22}\rangle$: $\mathcal{R}_1 = \{-\mathtt{a}, -\mathtt{b}\}$ and $\mathcal{R}_2 = \{+\mathtt{c}\}$. According to the definition given in this paper only $\mathcal{R}_2$ is founded, whereas according to the definition of founded update given in Caroprese et al. (2009) both updates are founded. □