

UNIVERSITÀ DI TRENTO
Department of
Information Engineering and Computer Science

Doctoral Programme in
Information Engineering and Computer Science

Efficient and Effective Extensions of SMT-Enumeration Techniques for Weighted Model Integration, Optimization, and Knowledge Compilation

Gabriele Masina

Advisor
Prof. Roberto Sebastiani
DISI, Università di Trento

12 October 2026

Abstract

Satisfiability Modulo Theories (SMT) enumeration is the problem of enumerating the theory-satisfiable truth assignments propositionally satisfying a given SMT formula. It underlies several tasks in formal verification, automated reasoning, and probabilistic inference. However, the number of such assignments may grow exponentially, limiting the applicability of SMT enumeration to large and complex problems.

This dissertation studies how suitable encodings and application-specific strategies can make SMT enumeration effective in practice and enable its use in novel problems. A central focus is the enumeration of few, short partial truth assignments, which compactly represent large sets of total assignments and reduce both enumeration costs and application-specific computation. We make the following contributions.

First, we show that standard CNF encodings may inhibit short partial-assignment generation and propose a novel encoding that drastically reduces both the number of assignments and the enumeration time. We then develop a structure-aware encoding and enumeration strategy for Weighted Model Integration, which significantly reduces the number of required integrals and improves upon state-of-the-art methods.

We also adapt partial-assignment enumeration to Optimization Modulo Theories, introducing optimization-specific reduction techniques that speed up the solving process. A related decomposition strategy is applied to constrained Maximum A Posteriori inference, enabling the optimization of highly nonlinear density functions through the combination of SMT enumeration and external numerical optimizers.

The dissertation then addresses the computation of a complete set of theory lemmas for a given SMT formula. We devise several techniques based on parallel cube-and-conquer and projected enumeration, improving scalability while preserving completeness. These lemmas are then used to define a framework for d-DNNFs Modulo Theories, which supports polytime SMT transformations and queries.

Extensive experiments support the practical effectiveness and real-world applicability of our contributions.

Keywords

SMT Enumeration, Weighted Model Integration, Optimization Modulo Theories, Constrained MAP Inference, Knowledge Compilation

Ringraziamenti

Vorrei iniziare ringraziando il mio advisor, Roberto. Grazie per la fiducia, per tutti i consigli e per gli stimoli che mi hai dato. Sei stato un mentore eccezionale, e porterò sempre con me tutto quello che mi hai insegnato.

I would also like to thank Antonio Vergari and all the members of the APRIL lab for giving me the opportunity to engage with topics beyond my usual research area. Thank you for making my stay in Edinburgh such a wonderful experience.

Un pensiero va poi a tutte le persone con cui ho avuto la possibilità di collaborare in questi anni, in particolare Paolo, e gli studenti che ho avuto il piacere di seguire come tesisti: Massimo, Thomas, Luca ed Emanuele. Spero che questo percorso sia stato per voi altrettanto stimolante e formativo quanto lo è stato per me.

Ad Alberto Montresor e a tutta la "cricca" di algoritmi devo alcune delle esperienze più formative e divertenti della mia carriera universitaria, dai tutorati agli incredibili progetti del corso. Grazie per avermi dato l'opportunità di lavorare con voi.

È arrivato il momento di ringraziare chi ha condiviso con me la vita di tutti i giorni in questi anni. Parto dai miei compagni dell'ufficio dei "Micio Miao". Grazie a Giuseppe: mi hai convinto a intraprendere il dottorato e hai condiviso con me tanti momenti di gioia e difficoltà, che hai sempre arricchito con la tua energia e il tuo entusiasmo. Grazie a Emiliano e Sebastiano, gli altri due "mici" arrivati poco dopo di me: siete stati dei compagni di avventura fantastici, e senza di voi questi anni sarebbero stati sicuramente meno folli e divertenti. Thank you also to Jingwen: we shared a short but intense part of our PhD journey, and I wish you all the best. Voglio ringraziare anche tutti i compagni dell'MMLab e non solo, sia quelli attuali —Antonella, Ardan, Chiara, Daniele, Nicola e Raffaele— sia quelli passati —Antonio, Fabio, Giulia, Niccolò e Zeno. Le pause pranzo con voi saranno sicuramente una delle cose che mi mancheranno di più.

Fuori dall'università, tante persone hanno condiviso con me una parte importante di questi anni. Grazie a Sebastiano, per avermi sempre supportato e per aver reso le mie giornate sempre stimolanti, soprattutto durante i tanti viaggi in treno. Grazie a Francesco e Giovanni per esserci sempre, anche a distanza. Grazie a Massimo, Cristina, Juri, Verdiana, Jasin e Sofia, per tutti i momenti passati insieme.

Infine, ringrazio la mia mamma e il mio papà, mia sorella Barbara, Mattia e la piccola Alice, per il vostro sostegno e per avermi sempre incoraggiato a seguire le mie passioni. Senza di voi non sarei arrivato fin qui.

Contents

I Background, State of the Art, and Related Work **7**

1 Background and State of the Art **9**
1.1 Propositional Satisfiability (SAT) . . . 9
1.2 Satisfiability Modulo Theories (SMT) . . . 10
1.2.1 Lazy SMT Solving . . . 14
1.3 SMT Enumeration ($\mathcal{T}$-AllSAT) . . . 14
1.3.1 Problem Definition . . . 14
1.3.2 $\mathcal{T}$-AllSAT Computation . . . 17
1.4 Weighted Model Integration (WMI) . . . 19
1.4.1 Problem Definition . . . 19
1.4.2 SMT-Based WMI Computation . . . 20
1.5 Optimization Modulo Theories (OMT) . . . 22
1.5.1 Problem Definition . . . 22
1.5.2 Lazy OMT Solving . . . 22
1.6 Constrained Maximum-A-Posteriori (MAP) Inference . . . 23
1.6.1 Problem Definition . . . 23
1.7 Propositional Knowledge Compilation (KC) . . . 24
1.7.1 Deterministic Decomposable Negation Normal Form (d-DNNF) 25
1.7.2 Polytime Transformations on d-DNNFs . . . 27
1.7.3 Polytime Queries on d-DNNFs . . . 27
1.7.4 Compilation into d-DNNF . . . 29

2 Related Work **31**
2.1 SAT and SMT Enumeration . . . 31
2.2 CNF Encodings and Enumeration . . . 33
2.3 Weighted Model Integration . . . 33
2.4 Optimization Modulo Theories . . . 34
2.5 Constrained MAP Inference . . . 35
2.6 Theory-Lemma Enumeration . . . 37
2.7 Knowledge Compilation Modulo Theories . . . 37

II Contributions **39**

3 On CNF Conversion for SAT and SMT Enumeration **41**

3.1 The Impact of CNF-izations on Enumeration . . . 42
3.1.1 The Impact of Tseitin CNF-ization . . . 42
3.1.2 The Impact of Plaisted and Greenbaum CNF-ization . . . 45
3.2 Enhancing Enumeration via NNF Preprocessing . . . 46
3.3 Experimental Evaluation . . . 51
3.3.1 An Analysis of Available Solvers . . . 51
3.3.2 Information for the Reproducibility of the Experiments . . . 52
3.3.3 Benchmarks . . . 52
3.3.4 Results . . . 53
3.3.5 Discussion: Using Our Encoding with Related Problems . . . 58
3.4 Conclusions . . . 59

4 SMT-Enumeration Techniques for Structure-Aware WMI 61
4.1 Analysis of the Limitations of State-of-the-Art WMI Solvers . . . 62
4.1.1 Knowledge Compilation . . . 62
4.1.2 WMI-PA . . . 64
4.2 A Structure-Aware WMI Algorithm . . . 65
4.2.1 General Idea: Exploiting a Conditional Skeleton of w . . . 66
4.2.2 Encoding a Conditional Skeleton Explicitly . . . 68
4.2.3 The SAE4WMI Procedure . . . 71
4.3 Experimental Evaluation . . . 73
4.3.1 Information for the Reproducibility of the Experiments . . . 73
4.3.2 Benchmarks . . . 73
4.3.3 Results . . . 74
4.4 Extending SAE4WMI with Different Integration Backends . . . 77
4.5 Conclusions . . . 79

5 SMT-Enumeration Techniques for Optimization Modulo Theories 81
5.1 An Analysis of Enumeration in OMT . . . 82
5.2 Exploiting Partial Truth Assignments in OMT . . . 83
5.2.1 Basic Assignment Reduction . . . 84
5.2.2 Guided Assignment Reduction . . . 84
5.3 Experimental Evaluation . . . 86
5.3.1 Information for the Reproducibility of the Experiments . . . 86
5.3.2 Benchmarks . . . 86
5.3.3 Results . . . 87
5.4 Conclusions and Future Work . . . 92

6 SMT Enumeration for MAP Inference with Non-Convex Constraints 93
6.1 PAMAP: Scaling Local Convex Optimizers via SMT Enumeration . . . 94
6.1.1 The General Scheme . . . 95
6.1.2 Optimization over Convex Polytopes . . . 95
6.1.3 Enumeration of Convex Polytopes . . . 96
6.1.4 Practical Limitations and Scalability . . . 97
6.2 A Scalable Message-Passing Algorithm for Constrained MAP . . . 97

6.3 Experimental Evaluation . . . 98
6.3.1 Baselines and Comparison . . . 99
6.3.2 Trajectory prediction on the Stanford Drone Dataset . . . 99
6.3.3 Data imputation of a VAE . . . 100
6.4 Conclusions . . . 102

7 A Theory-Agnostic Approach to Theory-Lemma Enumeration in SMT 103
7.1 Basic Definitions . . . 104
7.2 Theory-Lemma Enumeration Techniques . . . 105
7.2.1 Enumerating $\mathcal{T}$-lemmas via $\mathcal{T}$-AllSAT . . . 106
7.2.2 Cube and Conquer (and Parallelize) . . . 107
7.2.3 Projection on Theory-Atoms . . . 109
7.2.4 Theory-driven Partitioning . . . 110
7.3 Experimental Evaluation . . . 112
7.3.1 Information for the Reproducibility of the Experiments . . . 112
7.3.2 Benchmarks . . . 113
7.3.3 Results on Enumeration Time . . . 113
7.3.4 Results on the Number of Enumerated $\mathcal{T}$-Lemmas . . . 116
7.3.5 Results on the Size of $\mathcal{T}$-lemmas . . . 118
7.4 Conclusions . . . 120

8 d-DNNF Modulo Theories: A General Framework for Polytime SMT Transformations and Queries 121
8.1 d-DNNF Modulo Theories: An Analysis of the Problem . . . 123
8.2 Theoretical Framework . . . 125
8.2.1 $\mathcal{T}$-reduced and $\mathcal{T}$-extended $\mathcal{T}$-formulas . . . 125
8.2.2 $\mathcal{T}$-reduction and $\mathcal{T}$-extension via $\mathcal{T}$-lemmas . . . 127
8.2.3 Dealing with Extra $\mathcal{T}$-Atoms . . . 129
8.3 d-DNNF $\mathcal{T}$-Formulas and Their Polytime Transformations and Queries 130
8.3.1 Polytime Transformations on d-DNNF $\mathcal{T}$-Formulas by Exploiting $\mathcal{T}$-reduction and $\mathcal{T}$-extension . . . 131
8.3.2 Polytime Queries on d-DNNF $\mathcal{T}$-Formulas by Exploiting $\mathcal{T}$-reduction 132
8.3.3 Polytime Queries on d-DNNF $\mathcal{T}$-Formulas by Exploiting $\mathcal{T}$-extension 132
8.3.4 $\mathcal{T}$-Canonicity of $\mathcal{T}$-reduced and $\mathcal{T}$-extended d-DNNF . . . 133
8.4 Compiling, Transforming, and Querying d-DNNF $\mathcal{T}$-Formulas . . . 133
8.4.1 Compilation . . . 133
8.4.2 Transformations . . . 135
8.4.3 Queries . . . 135
8.5 Experimental Evaluation . . . 135
8.5.1 Information for the Reproducibility of the Experiments . . . 136
8.5.2 Benchmarks . . . 137
8.5.3 Results on Compilation Time . . . 138
8.5.4 Results on NNF Size . . . 140
8.5.5 Results on CE Queries . . . 141

8.5.6 Results on CT Queries . . . 142
8.6 Limitations, Research Avenues, and Potential Applications . . . 143
8.6.1 Limitations and Research Avenues . . . 143
8.6.2 Potential Applications . . . 144
8.7 Conclusions . . . 145

9 Conclusions and Future Work 147

III Appendices 173

A On CNF Conversion for SAT and SMT Enumeration 175
A.1 Proofs . . . 175
A.1.1 Proof of Lemma 1.2.1 . . . 175
A.1.2 Proof of Lemma 3.2.1 . . . 177
A.2 Additional Experimental Results . . . 179
A.2.1 An Analysis of Candidate Solvers . . . 179
A.2.2 Experimental Results on Plain SAT and SMT Solving . . . 180
A.2.3 Experimental Results on d-DNNF-based Enumeration . . . 181
A.2.4 Experimental Results on d-DNNF-based Model Counting . . . 182

B SMT-Enumeration Techniques for Structure-Aware WMI 183
B.1 Description of the datasets for the DET experiments . . . 183
B.2 Details on the program fairness verification task . . . 183

C SMT Enumeration for MAP Inference with Non-Convex Constraints 187
C.1 Particle, Constraint-Aware Adam Optimizer . . . 187
C.2 Details for SDD Experiments . . . 188
C.3 PaMap . . . 188
C.3.1 Enumeration and Upper Bounds . . . 188
C.3.2 SciPy's Optimizers . . . 188
C.4 PCAdam . . . 190
C.5 OMT Encoding . . . 190

D A Theory-Agnostic Approach to Theory-Lemma Enumeration in SMT 193
D.1 Proofs . . . 193
D.1.1 Proof of Theorem 7.2.1 . . . 193
D.1.2 Proof of Theorem 7.2.2 . . . 193
D.2 Experimental Evaluation: Additional Details . . . 193
D.2.1 Encoding of Numeric Planning Problems . . . 193

E d-DNNF Modulo Theories: A General Framework for Polytime SMT Transformations and Queries 195
E.1 A Survey of SMT-Level KC Compilers . . . 195
E.2 Proofs . . . 196

E.2.1 Some Auxiliary Propositions and Lemmas 196
E.2.2 Proof of Theorem 8.2.1 198
E.2.3 Proof of Theorem 8.2.2 198
E.2.4 Proof of Theorem 8.2.3 200
E.2.5 Proof of Theorem 8.2.4 200
E.2.6 Proof of Theorem 8.2.5 202
E.2.7 Proof of Theorem 8.2.6 202
E.2.8 Proof of Theorem 8.2.7 203
E.2.9 Proof of Theorem 8.2.8 204
E.2.10 Proof of Theorem 8.3.1 205
E.2.11 Proof of Theorem 8.3.2 205
E.2.12 Proof of Theorem 8.3.3 205
E.2.13 Proof of Theorem 8.3.4 206

List of Tables

1.1 Polytime transformations and queries satisfied by different target languages. ✓ indicates that the transformation or query is polytime, • indicates that it is not polytime, and ∘ indicates that it is not polytime unless P = NP. ? indicates that it is unknown whether the transformation or query is polytime. 29

A.1 List of AllSAT and $\mathcal{T}$-AllSAT solvers (rows) and supported features (columns). (In the "Available" column, "✗" indicates that the authors confirmed to us the unavailability of their tool, whereas "?" indicates that they did not reply to our inquiry.) 179

B.1 UCI datasets considered in our experiments. For each dataset, we report the number of Boolean and continuous variables, the numbers of training and validation instances, and the size of the resulting density estimation tree in terms of the number of conditions. 184

E.1 SMT-level KC compilers. In **Avail.**, ✓, ✗, and * mean, respectively, that the tool is publicly available, unavailable, or implemented within another tool and not usable. In **Canonical**, "?" means that the DDs may be canonical, but it was not proven. 195

List of Figures

1.1 An example of MAP($\mathcal{LRA}$) inference over non-convex constraints and a non-log-concave density. Orange stars indicate the solutions for the MAP problem for the unconstrained and constrained densities. 24
1.2 Different NNF representations for the formula $(\neg A_1 \vee A_2) \wedge (\neg A_2 \vee A_3)$. 26

3.1 Results on the Boolean benchmarks using MathSAT. Plots in 3.1a and 3.1b compare CNF-izations by $\mathcal{TA}$ size (first row) and execution time (second row). Points on dashed lines represent timeouts, shown in 3.1c. All axes use a logarithmic scale. 54
3.2 Results on the Boolean benchmarks using TabularAllSAT. Plots in 3.2a compare CNF-izations by $\mathcal{TA}$ size (first row) and execution time (second row). Points on dashed lines represent timeouts, shown in 3.2b. All axes use a logarithmic scale. 55
3.3 Results on the SMT($\mathcal{LRA}$) benchmarks using MathSAT. Plots in 3.3a, 3.3b compare CNF-izations by $\mathcal{TA}$ size (first row) and execution time (second row). Points on dashed lines represent timeouts, shown in 3.3c. All axes use a logarithmic scale. 56
3.4 Results on SMT($\mathcal{LRA}$) benchmarks using TabularAllSMT. Plots in 3.4a compare CNF-izations by $\mathcal{TA}$ size (first row) and execution time (second row). Points on dashed lines represent timeouts, shown in 3.4b. All axes use a logarithmic scale. 57

4.1 Example of exponential growth of XADDs for a weight function as in (4.1) with $N = 3$ and $\psi_i \stackrel{\text{def}}{=} A_i$, $i \in \{1, 2, 3\}$. Decision XADD nodes, i.e., if-then-else conditions, are circles labeled with decision atoms. Solid and dashed edges represent true and false branches, respectively. Square leaf nodes contain $\mathsf{FI}^{\mathcal{LRA}}$ weight functions. 63
4.2 Size of XADD diagram (number of nodes) and WMI-PA formula as the problem size increases, in logarithmic scale. Whereas the size of the diagram grows exponentially, WMI-PA encodes the problem into a formula of linear size. 63
4.3 Graphical representation of the effectiveness of different SMT-based WMI solvers on Example 4.2.3. SAE4WMI divides the integration domain into the minimum number of convex subdomains, each with a $\mathsf{FI}^{\mathcal{LRA}}$ weight. 70

4.4 Cactus plots of the synthetic experiments reporting execution time (left) for all methods and the number of integrals (right) for WMI-PA, SA-WMI-PA, and SAE4WMI. 75
4.5 Cactus plots representing average query execution times in seconds on a set of DET problems with $H \in \{0.25, 0.5, 0.75, 1\}$. 75
4.6 Cactus plots representing the number of computed integrals on a set of DET problems with $H \in \{0.25, 0.5, 0.75, 1\}$. 76
4.7 Cactus plots comparing SAE4WMI using different integration backends on the synthetic benchmarks. For SAE4WMI(VolEsti), we plot the median and color the range between the first and third quartiles of the outcomes obtained over 10 runs with different random seeds. Top-left: execution time of SAE4WMI with different integration backends. Bottom-left: zoom on the previous plot, showing only the results for SAE4WMI(VolEsti). Right: relative error of SAE4WMI(VolEsti) w.r.t. SAE4WMI(LattE) (where the latter does not time out). 78
4.8 The ratio in (4.8) computed for the initial (unfair) dec and the modified (fair) one. Average and standard deviation over 5 runs are reported, confirming the results reported in the FairSquare paper with high confidence. 79

5.1 Graphical representation of Example 5.1.1. For each step, the half-planes representing the literals in the truth assignment are delimited by dashed lines and colored in gray. The intersection of these constraints is colored in blue, with a gradient that follows the value of cost (the lower the value of cost, the darker the color), and the red dot represents the optimum model found within this region. 82
5.2 Results on OMT($\mathcal{LRA}$)-encoded optimal temporal planning problems. 88
5.3 Results on OMT($\mathcal{LRA}$)-encoded strip-packing problems. 89
5.4 Results on OMT($\mathcal{LIRA}$)-encoded strip-packing problems. 90
5.5 Results on OMT($\mathcal{LRA} \cup \mathcal{AR}$)-encoded strip-packing problems. 91

6.1 Left: constrained density for an agent's next position. Center-left: 257 convex polytopes considered by PaMap without upper-bound-based pruning. Center-right: piecewise-constant upper bound. Right: only 9 convex polytopes considered by PaMap with pruning. More examples are shown in Figure C.1 in Appendix C.3. 99
6.2 Left: Scatter plot of runtime vs. relative optimality gap. Right: Summary table reporting (SAT) instances solved with a point satisfying φ, (UNSAT) instances where the returned point does not satisfy φ, (ERR) instances terminated with an error, and (T.O.) number of timeouts (600s). Note that OptiMathSAT is anytime and returns the best solution found before timeout. 100
6.3 Left: Scatter plot comparing the relative errors of PaMap(PCAdam) and a particle-enhanced version of Adam. Right: Table comparing average relative error for PaMap(PCAdam), particle-enhanced Adam, and DL2. 101

7.1 $\mathcal{T}$-lemma enumeration time for different strategies on synthetic problems. (Notice that in the cactus plot the green and red lines are almost indistinguishable.) . . . 114
7.2 $\mathcal{T}$-lemma enumeration time for different strategies on temporal planning benchmarks. . . . 115
7.3 $\mathcal{T}$-lemma enumeration time for different strategies on numeric planning benchmarks. . . . 116
7.4 Number of $\mathcal{T}$-lemmas for different strategies on synthetic benchmarks. . 117
7.5 Number of $\mathcal{T}$-lemmas for different strategies on temporal planning benchmarks. . . . 117
7.6 Number of $\mathcal{T}$-lemmas for different strategies on numeric planning benchmarks. . . . 118
7.7 Median $\mathcal{T}$-lemma size for different strategies on synthetic benchmarks. 119
7.8 Median $\mathcal{T}$-lemma size for different strategies on temporal planning benchmarks. . . . 119
7.9 Median $\mathcal{T}$-lemma size for different strategies on numeric planning benchmarks. . . . 120

8.1 d-DNNF $\mathcal{T}$-formula with $\mathcal{T}$-inconsistent satisfying truth assignment (blue paths) from Example 8.1.1 (left), and a $\mathcal{T}$-equivalent d-DNNF $\mathcal{T}$-formula with no $\mathcal{T}$-inconsistent satisfying truth assignment (right). . . . 124
8.2 NNF representations for formulas in Example 8.2.3. Left: the NNF of φ; Center left: $\mathsf{d\text{-}DNNF}(\varphi)$; Center right: $\mathsf{d\text{-}DNNF}^{\mathsf{red}}_{\boldsymbol{\alpha}}(\varphi)$; Right: $\mathsf{d\text{-}DNNF}^{\mathsf{ext}}_{\boldsymbol{\alpha}}(\varphi)$. . . . 130
8.3 Compilation times for $\mathcal{T}$-reduced NNFs on synthetic benchmarks. Top: cactus plots showing the total compilation time for d-DNNF (left), OBDD (center), and SDD (right). We also show the time for lemma enumeration (blue) and Boolean compilation (orange) for each instance. For d-DNNFs, the green and blue lines overlap. We only show instances compiled within the timeout. Bottom: Number of timeouts for each compilation step. . 139
8.4 Compilation times for $\mathcal{T}$-reduced NNFs on numeric planning benchmarks. Top: a cactus plot showing the total compilation time for d-DNNF. We also show the time for lemma enumeration (blue) and Boolean compilation (orange) for each instance. Bottom: Number of timeouts for each compilation step. . . . 139
8.5 Size of the compiled $\mathcal{T}$-reduced NNFs on synthetic benchmarks. We compare the size of d-DNNFs (left), OBDDs (center), and SDDs (right) with the size of the input formulas. Sizes are measured by the number of nodes in their DAG representations. . . . 140
8.6 Size of the compiled $\mathcal{T}$-reduced NNFs on numeric planning benchmarks. We compare the size of d-DNNFs with the size of the input formulas. Sizes are measured by the number of nodes in their DAG representations. 140

8.7 Times for CE queries on synthetic problems. Left: scatter plot for $\mathcal{T}$-reduced d-DNNFs vs SMT with MathSAT. Center: comparison with cactus plot. Right: instances where CE does (CE) and does not (!CE) hold, and total instances (Tot.). 141
8.8 Times for CE queries on numeric planning problems. Left: scatter plot for $\mathcal{T}$-red. d-DNNFs vs SMT with MathSAT. Center: comparison with cactus plot. Right: instances where CE does (CE) and does not (!CE) hold, and total instances (Tot.). 141
8.9 Times for CT queries on synthetic problems. Left: scatter plot for $\mathcal{T}$-reduced d-DNNF vs $\mathcal{T}$-AllSAT-based counting with MathSAT. Center: comparison with cactus plot. Right: number of timeouts. 142
8.10 Times for CT queries on numeric planning problems. Left: scatter plot for $\mathcal{T}$-reduced d-DNNF vs $\mathcal{T}$-AllSAT-based counting with MathSAT. Center: comparison with cactus plot. Right: number of timeouts for each method. 142

A.1 2-root DAGs for the pair $\langle \mathsf{NNF}(\varphi), \mathsf{NNF}(\neg\varphi)\rangle$ for $\varphi \stackrel{\text{def}}{=} \varphi_1 \wedge \varphi_2$ and $\varphi \stackrel{\text{def}}{=} \varphi_1 \leftrightarrow \varphi_2$ (those for $\varphi_1 \vee \varphi_2$ and $\varphi_1 \rightarrow \varphi_2$ are similar to the one for $\varphi_1 \wedge \varphi_2$). 175
A.2 Three-valued semantics of $\varphi|_\mu$ in terms of $\{\top, \bot, *\}$ ("true", "false", "unknown"). 177
A.3 Time taken for plain SAT and SMT solving using the different CNF transformations. The y-axis reports the number of instances for which the solver finished within the cumulative time on the x-axis. 180
A.4 Results on the Boolean benchmarks using D4+decdnnf. Plots in A.4a compare CNF-izations by $\mathcal{TA}$ size (first row) and execution time (second row). Points on dashed lines represent timeouts, shown in A.4b. All axes use a logarithmic scale. 181
A.5 Results on the Boolean benchmarks using D4 for model counting. Plots in A.5a compare CNF-izations by execution time. Points on dashed lines represent timeouts, shown in A.5b. All axes use a logarithmic scale. . . 182

C.1 Visualization of the PaMap pruning process across 10 sample trajectories. Each group shows the density (left), the computed upper bounds used for pruning (center), and the polytopes analyzed by PaMap(SHGO) thanks to pruning (right). 189
C.2 Experiments on the SDD dataset varying the numerical optimizers used by PaMap (top) and the PCAdam hyperparameters (bottom). 191

Introduction

Satisfiability Modulo Theories (SMT) is the problem of deciding the satisfiability of a first-order formula with respect to a decidable first-order theory $\mathcal{T}$ [9]. SMT can be solved via a lazy approach combining a propositional SAT solver with a theory-specific procedure that checks the satisfiability of truth assignments with respect to the theory [191, 9]. SMT *solving* has been widely explored over the last decades, with several efficient solvers being developed for a variety of theories, such as linear arithmetic over real numbers ($\mathcal{LRA}$) [75], establishing SMT as a fundamental tool for formal verification and automated reasoning [9].

Several applications require not only finding a single solution but also exploring the entire solution space of a given SMT formula. A fundamental problem in this context is SMT *enumeration* ($\mathcal{T}$-AllSAT), which consists of enumerating all the theory-satisfiable truth assignments propositionally satisfying a given SMT formula. The problem has gained attention in recent years, with several works focusing on the development of efficient $\mathcal{T}$-AllSAT solvers, e.g., [129, 48, 207, 203], and on the study of its applications.

Such applications include, e.g., computing predicate abstraction in formal verification [129, 138], and probabilistic inference in hybrid domains via volume computation (#SMT($\mathcal{LRA}$)) [141, 241, 87, 200], and Weighted Model Integration (WMI) [13, 160, 161, 204]. In particular, WMI requires computing the integral of a given probability density over the solution space of a given SMT formula. A solving approach [160, 161] consists of enumerating theory-satisfiable truth assignments satisfying the formula, each defining a convex domain over which the integral of the density is directly computable. The final result is obtained by summing all the integrals thus computed. In this context, enumerating few, short partial truth assignments is crucial, as each of them requires computing an integral, which is a computationally expensive task.

Besides direct applications, several SMT-related problems involve some form of enumeration, though they do not explicitly use $\mathcal{T}$-AllSAT. Among these, Optimization Modulo Theories (OMT) [169, 195, 196] is a prominent example in which an SMT solver is used to enumerate the theory-satisfiable truth assignments of a formula, conjoining increasingly tighter bounds on the objective function until an optimum solution is found.

As SMT enumeration is very computationally expensive, a proper understanding of the problem, its possible variants, and the available techniques and algorithms for solving it efficiently is crucial for applying it in practice. In particular, for an efficient $\mathcal{T}$-AllSAT enumeration, it is of paramount importance to enumerate *short partial truth assignments*, rather than total truth assignments. In fact, a partial truth assignment propositionally satisfying the input formula can represent up to exponentially many

total ones. Hence, the shorter the partial assignments, the fewer assignments need to be enumerated, and the faster the enumeration can be performed.

We note that in several applications, this aspect is often overlooked. In many cases, algorithms rely on total truth assignment enumeration, which is often suboptimal and does not scale to large problems. Moreover, in other contexts where partial truth assignment enumeration is used, no sufficient investigation has been conducted to understand how the choice of encodings and techniques can affect the efficiency of the enumeration. Lastly, when total enumeration is strictly necessary, other techniques, such as projected $\mathcal{T}$-AllSAT [207], can be used by exploiting the structure of the problem.

Contributions

In this dissertation, we address the aforementioned challenges by building upon existing SMT-enumeration techniques and algorithms, and by studying their applicability, scalability, and effectiveness across a variety of problems. Our research focuses on a dual objective: optimizing the use of $\mathcal{T}$-AllSAT techniques for problems where they are already utilized and pioneering their application to domains where they have not been deployed before.

Crucially, this dissertation shows that the way we use existing $\mathcal{T}$-AllSAT tools is of paramount importance, demonstrating that the performance of $\mathcal{T}$-AllSAT can be improved by orders of magnitude simply by using the right encoding and the right techniques, without modifying the underlying solving algorithms.

Overall, this thesis presents the following contributions, all supported by extensive experimental evaluations.

CNF encodings for enumeration. As $\mathcal{T}$-AllSAT depends for its efficiency on the ability to enumerate few, short partial truth assignments, we study how the encoding of the input formula can affect the performance of enumeration. In particular, we focus on the CNF encodings commonly employed by SAT and SMT enumerators. We spotlight how the introduction of auxiliary label atoms and their corresponding definitions —even when the former are projected away during enumeration— can severely inhibit the enumerator from generating short partial assignments, resulting in significant performance degradation.

To address this, we propose a novel CNF encoding that combines the well-known Plaisted and Greenbaum encoding [178] with an upfront conversion into Negation Normal Form (NNF). This approach enables the enumeration of much shorter partial assignments compared to standard encodings. We validate this both in theory and in practice through a comprehensive experimental evaluation, demonstrating an improvement of several orders of magnitude in both the number of partial assignments enumerated and overall enumeration time.

Weighted Model Integration (WMI). We devise a novel WMI computation algorithm that combines a problem-specific encoding of the input formula with a

specialized SMT-enumeration strategy designed to minimize the number of integrals to compute. In particular, the combination of these two elements enables our solver to effectively and efficiently: (a) enumerate the branches of piecewise-defined weight functions, which are often deeply nested and can contain complex $\mathcal{LRA}$ conditions; and (b) handle formulas containing combinations of $\mathcal{LRA}$ and Boolean atoms, where partial truth assignments on the former define the integration domain, while partial truth assignments on the latter define a multiplicative factor for the integral. Overall, this approach dramatically reduces the number of integrals to compute w.r.t. other SMT-enumeration-based approaches [160, 161, 204], thereby reducing the total WMI computation time and achieving state-of-the-art performance on both synthetic and real-world benchmarks.

Optimization Modulo Theories (OMT). We also study the application of SMT-enumeration techniques to Optimization Modulo Theories (OMT), and in particular, we discuss how the OMT loop can be improved by exploiting partial assignment enumeration. We propose techniques to tailor existing partial-assignment reductions to the optimization setting and show how in practice these techniques can be used to improve the efficiency of OMT solving for several theories, such as linear arithmetic over real or integer numbers, as well as combinations of theories. Moreover, this strategy proves particularly effective for anytime OMT solving, i.e., finding the best solution within a given time limit, allowing us to find good-quality solutions much faster than standard OMT approaches.

Maximum-A-Posteriori (MAP) inference. Staying in the optimization context, we pioneer a novel application of SMT-enumeration techniques to Maximum-A-Posteriori (MAP) inference under SMT($\mathcal{LRA}$) constraints. The problem consists of maximizing a probability density function, typically highly nonlinear and possibly not given in explicit form, in the solution space defined by an $\mathcal{LRA}$-formula. Since standard OMT solving techniques for nonlinear arithmetic [18, 113] cannot deal with such density functions, we propose an alternative approach driven by SMT enumeration. Similarly to WMI computation, we use $\mathcal{T}$-AllSAT to partition the solution space into a set of subproblems, each corresponding to a partial truth assignment satisfying the formula, which can then be handled by existing optimizers. Our approach applies to a wide range of density functions, providing exact or approximate solutions depending on the underlying optimizer used.

Theory-lemma enumeration. As another novel application of SMT-enumeration techniques, we study the problem of computing a complete set of theory lemmas that rule out all theory-unsatisfiable truth assignments propositionally satisfying a given SMT formula. This problem poses some hard challenges: partial truth assignment enumeration does not guarantee finding all the required theory lemmas, whereas total truth assignment enumeration is typically infeasible even for relatively small problems. To overcome these issues, we develop efficient techniques to scale to substantially larger formulas than what is possible with standard $\mathcal{T}$-AllSAT techniques. First, we introduce a highly parallelizable cube-and-conquer approach to $\mathcal{T}$-AllSAT, using an existing $\mathcal{T}$-AllSAT

solver as a black box. Second, we show how using advanced enumeration techniques based on projected $\mathcal{T}$-AllSAT can provably find a complete set of $\mathcal{T}$-lemmas. When atom partitioning based on shared symbols is possible, this allows us to dramatically reduce the computational cost. Overall, by combining these improvements, we can scale to much larger and more complex formulas than with standard $\mathcal{T}$-AllSAT.

d-DNNFs Modulo Theories. Computing such a set of theory lemmas serves as the fundamental building block for the last contribution of this dissertation: a comprehensive framework for d-DNNFs Modulo Theories supporting polytime SMT transformations and queries. By augmenting the input formula with theory lemmas, we show how to construct a compiled d-DNNF $\mathcal{T}$-formula. We provide formal proofs of correctness for the associated querying and transformation algorithms, enabling polytime operations such as counting and enumeration of $\mathcal{T}$-satisfiable truth assignments. Experimental results confirm the practical effectiveness of our approach, showing that, e.g., counting queries, which are prohibitively expensive for standard SMT-based approaches, can be answered in negligible time on the compiled d-DNNF $\mathcal{T}$-formulas.

Publications

A significant part of the content of this dissertation has already been published in conferences and journals. Here is a list of all papers published during the Ph.D. program, in chronological order.

Journal papers.

- [205] Giuseppe Spallitta, Gabriele Masina, Paolo Morettin, Andrea Passerini, and Roberto Sebastiani. "Enhancing SMT-based Weighted Model Integration by Structure Awareness". In: *Artificial Intelligence* 328 (2024), p. 104067. DOI: 10.1016/j.artint.2024.104067
- [150] Gabriele Masina, Giuseppe Spallitta, and Roberto Sebastiani. "On CNF Conversion for SAT and SMT Enumeration". In: *Journal of Artificial Intelligence Research* 83 (2025). DOI: 10.1613/jair.1.16870

Conference papers.

- [149] Gabriele Masina, Giuseppe Spallitta, and Roberto Sebastiani. "On CNF Conversion for Disjoint SAT Enumeration". In: *26th International Conference on Theory and Applications of Satisfiability Testing*. Vol. 271. LIPIcs. LZI, 2023, 15:1–15:16. DOI: 10.4230/LIPIcs.SAT.2023.15
- [152] Massimo Michelutti, Gabriele Masina, Giuseppe Spallitta, and Roberto Sebastiani. "Canonical Decision Diagrams Modulo Theories". In: *27th European Conference on Artificial Intelligence*. Vol. 392. FAIA. IOS Press, 2024, pp. 4319–4327. DOI: 10.3233/FAIA241007
- [148] Gabriele Masina and Roberto Sebastiani. "Exploiting Partial-Assignment Enumeration in Optimization Modulo Theories". In: *Frontiers of Combining Systems*. Springer, 2026, pp. 117–134. DOI: 10.1007/978-3-032-04167-8_7
- [123] Leander Kurscheidt, Gabriele Masina, Roberto Sebastiani, and Antonio Vergari. "The Theory and Practice of MAP Inference over Non-Convex Constraints". In: *Proceedings of the 43rd International Conference on Machine Learning*. 2026. URL: https://openreview.net/forum?id=jIZqAemuqk
- [50] Emanuele Civini, Gabriele Masina, Giuseppe Spallitta, and Roberto Sebastiani. "Beyond Eager Encodings: A Theory-Agnostic Approach to Theory-Lemma Enumeration in SMT". in: *International Joint Conference on Automated Reasoning*. Springer, 2026, pp. 286–304. DOI: 10.1007/978-3-032-32589-1_18
- [147] Gabriele Masina, Emanuele Civini, Massimo Michelutti, Giuseppe Spallitta, and Roberto Sebastiani. "d-DNNF Modulo Theories: A General Framework for Polytime SMT Queries". In: *29th International Conference on Theory and Applications of Satisfiability Testing*. Vol. 377. LIPIcs. LZI, 2026, 25:1–25:19. DOI: 10.4230/LIPIcs.SAT.2026.25

Thesis structure

The thesis is structured as follows:

Part I provides an introduction to necessary background knowledge and terminology, and a survey of the related work:

Chapter 1 introduces the notation and the necessary background on SMT, SMT Enumeration, WMI, OMT, and Knowledge Compilation, and presents the problems we address in this thesis.

Chapter 2 provides an overview of the state of the art in the addressed problems.

Part II presents our contributions:

Chapter 3 is based on [149, 150]. It addresses the problem of SAT and SMT enumeration for non-CNF formulas, analyzing the limitations of common CNF encodings for enumeration, and proposing a new CNF encoding that allows for enumerating a much smaller number of short partial assignments.

Chapter 4 is based on [205]. It presents our contributions on improving the efficiency of SMT-based WMI computation, including novel ad-hoc structure-aware encodings and SMT-enumeration techniques.

Chapter 5 is based on [148]. It presents an adaptation of SMT-enumeration techniques to improve the efficiency and effectiveness of OMT solving.

Chapter 6 is based on [123]. It presents an application of SMT-enumeration techniques to compute Maximum-A-Posteriori (MAP) inference under SMT($\mathcal{LRA}$) constraints and highly nonlinear objectives.

Chapter 7 is based on [50]. It presents several methods to exploit SMT Enumeration for computing a complete set of theory lemmas ruling out all theory-unsatisfiable truth assignments propositionally satisfying a given SMT formula. These methods are used as a building block for the framework presented in the next chapter.

Chapter 8 is based on [152, 147]. It presents a general theory-agnostic framework for compiling SMT formulas into d-DNNF $\mathcal{T}$-formulas, in such a way as to allow polytime SMT transformations and queries.

Chapter 9 concludes the thesis and outlines future research directions.

Part I

Background, State of the Art, and Related Work

Chapter 1

Background and State of the Art

This chapter introduces the notation and the background necessary to understand the rest of the thesis. We assume the reader is familiar with the basic syntax, semantics, and results of propositional and first-order logics and refer to standard literature (e.g., [9, 17]) for details.

Notation. In this thesis, we adopt the following notation conventions. A, B denote Boolean atoms; α, β denote first-order (including Boolean) atoms; ℓ denotes a literal; x, y denote first-order variables; φ, ψ, χ denote propositional or first-order formulas; η, ρ, μ denote conjunctions of literals, where η and ρ typically denote total truth assignments, whereas μ denotes a partial truth assignment; C denotes a clause; $\mathbb{B} \stackrel{\text{def}}{=} \{\top, \bot\}$ denotes the set of Boolean constants, and $\mathbb{R}$ denotes the set of real numbers. Bold symbols denote sets: $\mathbf{A} \stackrel{\text{def}}{=} \{A_1, \ldots, A_N\}, \mathbf{B} \stackrel{\text{def}}{=} \{B_1, \ldots, B_M\}$ denote disjoint sets of Boolean atoms; $\boldsymbol{\alpha} \stackrel{\text{def}}{=} \{\alpha_1, \ldots, \alpha_N\}, \boldsymbol{\beta} \stackrel{\text{def}}{=} \{\beta_1, \ldots, \beta_M\}$ denote disjoint sets of first-order (including Boolean) atoms; $\mathbf{x} \stackrel{\text{def}}{=} \{x_1, \ldots, x_N\}, \mathbf{y} \stackrel{\text{def}}{=} \{y_1, \ldots, y_M\}$ denote disjoint sets of first-order variables. $\mathcal{T}$ denotes a first-order theory, and $\mathcal{M}$ denotes a first-order interpretation.

Given a Boolean or first-order formula φ, $\mathit{Atoms}(\varphi)$ denotes the set of atoms occurring in φ. We sometimes adopt the notation "$\varphi[\boldsymbol{\alpha}]$" when we want to emphasize that $\boldsymbol{\alpha}$ is a superset of $\mathit{Atoms}(\varphi)$ whose truth assignments we are interested in. The fact that it is a superset is sometimes necessary for comparing formulas with different sets of atoms: $\varphi[\boldsymbol{\alpha}]$ and $\varphi'[\boldsymbol{\alpha'}]$ can be compared only if they are both considered as formulas on $\boldsymbol{\alpha} \cup \boldsymbol{\alpha'}$. (For example, in order to check that $(A_1 \vee A_2) \wedge (A_1 \vee \neg A_2)$ and $(A_1 \vee A_3) \wedge (A_1 \vee \neg A_3)$ are equivalent, we need to consider them as formulas on $\{A_1, A_2, A_3\}$.)

1.1 Propositional Satisfiability (SAT)

A *propositional* (also *Boolean*) *formula* φ can be defined recursively as follows. The Boolean constants $\top$ and $\bot$ are formulas; a Boolean atom A and its negation $\neg A$ are formulas, also referred to as *literals*; a combination of two formulas using one of the Boolean connectives $(\wedge, \vee, \rightarrow, \leftrightarrow)$ is a formula; the negation $(\neg)$ of a formula is a formula. A *clause* is a disjunction $(\vee)$ of literals. A *cube* is a conjunction $(\wedge)$ of literals.

With a little abuse of notation, if ℓ is the negative literal $\neg A$, then by "$\neg\ell$" we mean "A" rather than "$\neg\neg A$"; also, if C is a clause $\bigvee_i \ell_i$, then by "$\neg C$" we mean the cube $\bigwedge_i \neg\ell_i$ rather than $\neg \bigvee_i \ell_i$, and vice versa.

Propositional satisfiability (SAT) is the problem of deciding the satisfiability of a propositional formula, i.e., whether there is a way to assign truth values to the atoms such that the formula evaluates to $\top$. We adopt the standard definitions of propositional entailment ("$\models$"), equivalence ("$\equiv$"), and validity for propositional formulas.

1.2 Satisfiability Modulo Theories (SMT)

Basics. A *first-order term* is either a variable, a constant symbol, or a combination of terms using function symbols. A *first-order atom* is a predicate symbol applied to a tuple of terms (Boolean atoms can be viewed as zero-arity predicates). The definitions of *first-order literals, clauses, cubes, and formulas* follow from those in propositional logic, except that atoms are now first-order atoms. As it is standard in most SMT literature, we restrict ourselves to *quantifier-free* first-order formulas. Hereafter, unless otherwise specified, "formulas" [resp. atoms, literals] refers to both propositional and first-order formulas [resp. atoms, literals]. We assume that formulas are internally represented as *single-rooted directed acyclic graphs (DAGs)* where internal nodes are labeled with Boolean connectives, and leaves are labeled with literals or atoms. (Some authors call them "circuits", e.g., [83, 82].) In this way, multiple occurrences of the same subformula are represented by only one shared sub-DAG. The *size* of a formula is the number of nodes and arcs of its DAG representation.

Theory ($\mathcal{T}$). A *first-order theory* $\mathcal{T}$ is a (possibly infinite) set of first-order formulas that provides an intended interpretation of constant, function, and predicate symbols. Examples of theories of practical interest are those of equalities ($\mathcal{E}$), equalities and uninterpreted functions ($\mathcal{EUF}$), linear arithmetic over integer ($\mathcal{LIA}$) or real numbers ($\mathcal{LRA}$), nonlinear arithmetic ($\mathcal{NIA}$, $\mathcal{NRA}$), and combinations thereof. With a little abuse of notation, we often refer to a theory $\mathcal{T}$ instead of its corresponding signature, so that by $\mathcal{T}$-formulas [resp. literals, atoms] we mean formulas [resp. literals, atoms] in the signature of $\mathcal{T}$. For example, $\mathcal{LRA}$-atoms are of the form $(c_1 \cdot x_1 + \cdots + c_n \cdot x_n \bowtie c)$, where $c_1, \ldots, c_n$ and c are rational constants, x_i are real variables in $\mathbb{R}$, "$\cdot$" and "$+$" are the standard multiplication and addition operators, and "$\bowtie$" is one of the standard arithmetic operators $\{=, \neq, <, >, \leq, \geq\}$. A *$\mathcal{T}$-model* $\mathcal{M}$ is an interpretation of the symbols of $\mathcal{T}$ that satisfies every formula in $\mathcal{T}$.

Satisfiability Modulo Theories. *Satisfiability Modulo the Theory* $\mathcal{T}$, also $SMT(\mathcal{T})$, is the problem of deciding the satisfiability of a first-order formula w.r.t. some background theory $\mathcal{T}$ (see, e.g., [9]). A $\mathcal{T}$-formula φ is *$\mathcal{T}$-satisfiable* (also *$\mathcal{T}$-consistent*) if there exists a $\mathcal{T}$-model $\mathcal{M}$ such that $\mathcal{M} \models \varphi$ in the standard first-order semantics. Otherwise, it is *$\mathcal{T}$-unsatisfiable* (also *$\mathcal{T}$-inconsistent*).

Example 1.2.1 (SMT). Consider the $\mathcal{LRA}$-formula on the rational variables x, y:

$$\varphi \stackrel{\text{def}}{=} \Big((2x - 3y \leq 6) \vee (x \leq 4) \Big) \wedge \Big((y \leq 2) \vee (y \leq -3x + 9) \vee (x < -2) \Big). \quad (1.1)$$

φ is $\mathcal{LRA}$-satisfiable, e.g., the $\mathcal{LRA}$-model $\mathcal{M} \stackrel{\text{def}}{=} \{x \mapsto 3, y \mapsto 0\}$ satisfies φ.

A $\mathcal{T}$-formula φ is *$\mathcal{T}$-valid* iff it is satisfied by every $\mathcal{T}$-model. A *$\mathcal{T}$-lemma* is a $\mathcal{T}$-valid clause. Given two $\mathcal{T}$-formulas φ and ψ, we say that φ *$\mathcal{T}$-entails* ψ, written $\varphi \models_{\mathcal{T}} \psi$, iff every $\mathcal{T}$-model satisfying φ also satisfies ψ. We say that φ and ψ are *$\mathcal{T}$-equivalent*, written $\varphi \equiv_{\mathcal{T}} \psi$, iff $\varphi \models_{\mathcal{T}} \psi$ and $\psi \models_{\mathcal{T}} \varphi$.

Boolean Abstraction and Refinement. $\mathcal{T}2\mathcal{B}$ denotes a bijective function ("theory to Boolean"), called *Boolean abstraction*, which maps Boolean atoms into themselves, first-order atoms into fresh Boolean atoms, and is homomorphic w.r.t. Boolean operators and set inclusion. $\mathcal{B}2\mathcal{T}$ ("Boolean to theory"), called *refinement*, is the inverse of $\mathcal{T}2\mathcal{B}$. For instance $\mathcal{T}2\mathcal{B}(\{((x_1 - x_2 \leq 3) \vee (x_1 = x_3))\}) = \{(A_1 \vee A_2)\}$, A_1 and A_2 being fresh Boolean atoms, and $\mathcal{B}2\mathcal{T}(\{\neg A_1, A_2\}) = \{\neg(x_1 - x_2 \leq 3), (x_1 = x_3)\}$.

We use φ^p as a synonym for the Boolean abstraction of φ, and vice versa, i.e., φ denotes the refinement of φ^p. This extends to sets of formulas as well.

We say that a formula φ is *propositionally satisfiable* (also $\mathbb{B}$-satisfiable) iff $\mathcal{T}2\mathcal{B}(\varphi)$ is satisfiable. We say that a formula φ is *propositionally valid* (also $\mathbb{B}$-valid) iff $\mathcal{T}2\mathcal{B}(\varphi)$ is valid. We say that φ *propositionally entails* (also $\mathbb{B}$-entails) ψ, denoted $\varphi \models_{\mathbb{B}} \psi$, iff $\mathcal{T}2\mathcal{B}(\varphi) \models \mathcal{T}2\mathcal{B}(\psi)$, and that φ and ψ are *propositionally equivalent* (also $\mathbb{B}$-equivalent), denoted $\varphi \equiv_{\mathbb{B}} \psi$, iff $\mathcal{T}2\mathcal{B}(\varphi)$ and $\mathcal{T}2\mathcal{B}(\psi)$ are equivalent. Notice that $\varphi \models_{\mathbb{B}} \psi$ implies $\varphi \models_{\mathcal{T}} \psi$, but not vice versa, and that $\varphi \equiv_{\mathbb{B}} \psi$ implies $\varphi \equiv_{\mathcal{T}} \psi$, but not vice versa.

Polarity. Given a formula φ, a subformula occurs with *positive* [resp. *negative*] *polarity* (also *positively* [resp. *negatively*]) if it occurs under an even [resp. odd] number of nested negations. Specifically, φ occurs positively in φ; if $\neg\varphi_1$ occurs positively [resp. negatively] in φ, then φ_1 occurs negatively [resp. positively] in φ; if $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$ occur positively [resp. negatively] in φ, then φ_1 and φ_2 occur positively [resp. negatively] in φ; if $\varphi_1 \rightarrow \varphi_2$ occurs positively [resp. negatively] in φ, then φ_1 occurs negatively [resp. positively] and φ_2 occurs positively [resp. negatively] in φ; if $\varphi_1 \leftrightarrow \varphi_2$ occurs in φ, then both φ_1 and φ_2 occur positively and negatively.

Negation Normal Form. A formula is in *Negation Normal Form (NNF)* iff it is given by the recursive applications of $\wedge$ and $\vee$ to literals. Note that all its subformulas occur positively. A formula φ can be converted into a $\mathbb{B}$-equivalent NNF formula via the $\mathsf{NNF}(\varphi)$ transformation, which rewrites implications $(\varphi_1 \rightarrow \varphi_2)$ as $(\neg\varphi_1 \vee \varphi_2)$ and equivalences $(\varphi_1 \leftrightarrow \varphi_2)$ as $(\neg\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \neg\varphi_2)$, and then recursively "pushes down" negations, rewriting $\neg(\varphi_1 \vee \varphi_2)$ as $(\neg\varphi_1 \wedge \neg\varphi_2)$, $\neg(\varphi_1 \wedge \varphi_2)$ as $(\neg\varphi_1 \vee \neg\varphi_2)$ and $\neg\neg\varphi_1$ as φ_1.

Lemma 1.2.1. *Given a formula φ, the size of $\mathsf{NNF}(\varphi)$ is linear w.r.t. the size of φ.*[a]

[a] We recall that we assume formulas are represented as DAGs.

Although we believe this fact is well-known, we provide a formal proof in Appendix A.1.1.

Conjunctive Normal Form. A formula is in *Conjunctive Normal Form (CNF)* iff it is a conjunction of clauses. Numerous CNF transformation procedures, commonly referred to as *CNF-izations*, have been proposed in the literature. We summarize the three most frequently employed techniques.

The *Classic CNF-ization* ($\mathsf{CNF}_{\mathsf{DM}}$) converts a formula into a propositionally equivalent formula in CNF by applying De Morgan's rules. First, it converts the formula into NNF. Then, it recursively rewrites subformulas $\varphi_1 \vee (\varphi_2 \wedge \varphi_3)$ as $(\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \varphi_3)$ to distribute $\vee$ over $\wedge$, until the formula is in CNF. The principal limitation of this transformation lies in the possible exponential growth of the resulting formula compared to the original (e.g., when the formula is a disjunction of conjunctions of subformulas), making it unsuitable for modern SAT and SMT solvers (see, e.g., [180]).

The *Tseitin CNF-ization* ($\mathsf{CNF}_{\mathsf{Ts}}$) [222] avoids this exponential blow-up by labeling each subformula φ_i with a fresh Boolean atom B_i, which is used as a placeholder for the subformula. Specifically, it consists of recursively applying the bottom-up rewriting rule $\varphi \implies \varphi[\varphi_i|B_i] \wedge \mathsf{CNF}_{\mathsf{DM}}(B_i \leftrightarrow \varphi_i)$ until the resulting formula is in CNF, where $\varphi[\varphi_i|B_i]$ is the formula obtained by substituting in φ every occurrence of φ_i with B_i.

The *Plaisted and Greenbaum CNF-ization* ($\mathsf{CNF}_{\mathsf{PG}}$) [178] is a variant of $\mathsf{CNF}_{\mathsf{Ts}}$ that exploits the polarity of subformulas to reduce the number of clauses of the final formula. If a subformula φ_i appears only with positive [resp. negative] polarity, then it can be labeled with a one-way implication as $\mathsf{CNF}_{\mathsf{DM}}(B_i \rightarrow \varphi_i)$ [resp. $\mathsf{CNF}_{\mathsf{DM}}(B_i \leftarrow \varphi_i)$]; if φ_i occurs with both polarities, then it is labeled as $\mathsf{CNF}_{\mathsf{DM}}(B_i \leftrightarrow \varphi_i)$, as with $\mathsf{CNF}_{\mathsf{Ts}}$.

With both $\mathsf{CNF}_{\mathsf{Ts}}$ and $\mathsf{CNF}_{\mathsf{PG}}$, due to the introduction of the label atoms, the final formula is not propositionally equivalent to the original formula but only equisatisfiable. Moreover, they have a stronger property. If $\varphi[\boldsymbol{\alpha}]$ is a non-CNF formula and $\psi[\boldsymbol{\alpha} \cup \mathbf{B}]$ is either the $\mathsf{CNF}_{\mathsf{Ts}}$ or the $\mathsf{CNF}_{\mathsf{PG}}$ encoding of φ, where $\mathbf{B}$ are the fresh Boolean atoms introduced by the transformation, then $\varphi \equiv_{\mathbb{B}} \exists \mathbf{B}.\psi$.

Total and partial truth assignments. Given a set $\boldsymbol{\alpha}$ of atoms, a *total truth assignment* is a total map $\eta : \boldsymbol{\alpha} \longmapsto \mathbb{B}$. A *partial truth assignment* is a partial map $\mu : \boldsymbol{\alpha} \longmapsto \mathbb{B}$. Notice that a partial truth assignment represents (also "covers") 2^k total truth assignments extending it, where k is the number of unassigned atoms in μ. We often represent a truth assignment either as a set, s.t. $\mu \stackrel{\text{def}}{=} \{\alpha \mid \mu(\alpha) = \top\} \cup \{\neg\alpha \mid \mu(\alpha) = \bot\}$, or as a conjunction of literals, s.t. $\mu \stackrel{\text{def}}{=} \bigwedge_{\mu(\alpha)=\top} \alpha \wedge \bigwedge_{\mu(\alpha)=\bot} \neg\alpha$. If $\mu_1 \subseteq \mu_2$ [resp. $\mu_1 \subset \mu_2$] we say that μ_1 is a *subassignment* [resp. *strict subassignment*] of μ_2 and that μ_2 is a *super-assignment* [resp. *strict super-assignment*] of μ_1.

We denote with $\varphi|_\mu$ the *residual of φ under μ*, i.e., the formula obtained by substituting in φ each $\alpha_i \in \boldsymbol{\alpha}$ with $\mu(\alpha_i)$, and by recursively applying the standard propagation rules of truth values through Boolean operators.

Definition 1.2.1 (Satisfiability by partial assignment). Let $\varphi[\boldsymbol{\alpha}]$ be a formula.

We say that a *(partial or total) truth assignment* μ *on* $\boldsymbol{\alpha}$ *propositionally satisfies* φ, denoted $\mu \models_{\mathbb{B}} \varphi$, iff $\mathcal{T}2\mathcal{B}(\mu) \models \mathcal{T}2\mathcal{B}(\varphi)$.

A Boolean formula is satisfiable iff there exists a truth assignment propositionally satisfying it. A $\mathcal{T}$-formula is $\mathcal{T}$-satisfiable iff there exists a $\mathcal{T}$-satisfiable truth assignment propositionally satisfying it. A partial truth assignment μ is *minimal* for φ iff $\mu \models_{\mathbb{B}} \varphi$ and every strict subassignment $\mu' \subset \mu$ is such that $\mu' \not\models_{\mathbb{B}} \varphi$.

Remark 1.2.1. The definition of *satisfiability by partial assignment* may be ambiguous for non-CNF and existentially-quantified formulas [193, 156, 194]. In this thesis, unless otherwise specified, we use the *verification* semantics, i.e., we say that $\mu \models_{\mathbb{B}} \varphi$ iff $\varphi|_\mu = \top$. This definition is the easiest and most efficient to compute, and is the one typically used by state-of-the-art SAT and SMT solvers. Alternatively, the *entailment* semantics defines $\mu \models_{\mathbb{B}} \varphi$ iff $\varphi|_\mu \equiv_{\mathbb{B}} \top$, as done by OBDDs [32], SDDs [60], or dual-reasoning techniques [155, 82] (see also Section 2.1).

Remark 1.2.2. The above notion of satisfiability by partial assignment is such that if $\varphi_1 \equiv_{\mathbb{B}} \varphi_2$, then $\mu \models_{\mathbb{B}} \varphi_1$ *does not* imply that $\mu \models_{\mathbb{B}} \varphi_2$ [193, 156, 194]. Consider, e.g., $\varphi_1 \stackrel{\text{def}}{=} (A \vee B) \wedge (A \vee \neg B)$ and $\varphi_2 \stackrel{\text{def}}{=} (A \wedge B) \vee (A \wedge \neg B)$, and the partial assignment $\mu \stackrel{\text{def}}{=} \{A\}$. Although $\varphi_1 \equiv_{\mathbb{B}} \varphi_2$, we have that $\varphi_1|_\mu = \top$, but $\varphi_2|_\mu = B \vee \neg B$, which, although *logically* equivalent to $\top$, is *syntactically* different from it.

Sometimes, we are interested in finding truth assignments propositionally satisfying a formula w.r.t. a subset of atoms.

Definition 1.2.2 (Projected propositional satisfiability). Let $\varphi[\boldsymbol{\alpha}]$ be a formula, and let $\boldsymbol{\alpha'}, \boldsymbol{\alpha''}$ be s.t. $\boldsymbol{\alpha} \stackrel{\text{def}}{=} \boldsymbol{\alpha'} \cup \boldsymbol{\alpha''}$ and they are *symbol-disjoint*, i.e., they do not share any Boolean atom, first-order variable, or uninterpreted function. We call $\boldsymbol{\alpha'}$ the set of *projection atoms*. Let $\mathbf{B}$, $\mathbf{y}$, and $\mathbf{f}$ be, respectively, the sets of Boolean atoms, first-order variables, and uninterpreted functions occurring in $\boldsymbol{\alpha''}$.

We say that a *(partial or total) truth assignment* $\mu^{\boldsymbol{\alpha'}}$ *on* $\boldsymbol{\alpha'}$ *propositionally satisfies* $\exists\mathbf{B}.\exists\mathbf{y}.\exists\mathbf{f}.\varphi[\boldsymbol{\alpha}]$, denoted as $\mu^{\boldsymbol{\alpha'}} \models_{\mathbb{B}} \exists\mathbf{B}.\exists\mathbf{y}.\exists\mathbf{f}.\varphi[\boldsymbol{\alpha}]$, iff there exists a $\mathcal{T}$-satisfiable total truth assignment $\eta^{\boldsymbol{\alpha''}}$ on $\boldsymbol{\alpha''}$ s.t. $\mu^{\boldsymbol{\alpha'}} \cup \eta^{\boldsymbol{\alpha''}} \models_{\mathbb{B}} \varphi[\boldsymbol{\alpha}]$.

We note that the notation "$\exists\mathbf{f}$" is not compliant with standard first-order logic syntax. It is intended to denote the fact that there exists a $\mathcal{T}$-model for $\mathbf{B}$, $\mathbf{y}$, and $\mathbf{f}$ inducing a $\mathcal{T}$-satisfiable truth assignment on $\boldsymbol{\alpha''}$ such that $\mu^{\boldsymbol{\alpha'}} \cup \eta^{\boldsymbol{\alpha''}} \models_{\mathbb{B}} \varphi$. We use this notation to keep the definition as general as possible, so as to include uninterpreted functions, though in practice the latter never appear in the rest of the thesis. Instead, we frequently consider the special cases where $\mathbf{B}$, $\mathbf{f}$, or $\mathbf{y}$ are empty. In particular, if $\mathbf{f} = \mathbf{y} = \emptyset$ and $\boldsymbol{\alpha''} = \mathbf{B}$, then $\mu^{\boldsymbol{\alpha'}} \models_{\mathbb{B}} \exists\mathbf{B}.\varphi$ iff there exists an $\eta^{\mathbf{B}}$ such that $\mu^{\boldsymbol{\alpha'}} \cup \eta^{\mathbf{B}} \models_{\mathbb{B}} \varphi$.

1.2.1 Lazy SMT Solving

State-of-the-art SMT solvers implement the CDCL($\mathcal{T}$) algorithm [143]. CDCL($\mathcal{T}$) is based on the so-called lazy approach to SMT (see e.g., [191, 9]), which exploits the fact that a $\mathcal{T}$-formula φ is $\mathcal{T}$-satisfiable iff there exists a truth assignment μ that propositionally satisfies φ and μ is $\mathcal{T}$-satisfiable. It combines a CDCL-based SAT-solver with a $\mathcal{T}$-specialized decision procedure called $\mathcal{T}$-solver to decide the $\mathcal{T}$-consistency of a set of $\mathcal{T}$-literals. Whenever the SAT-solver finds a truth assignment μ propositionally satisfying φ, it invokes the $\mathcal{T}$-solver to check the $\mathcal{T}$-satisfiability of μ. If μ is $\mathcal{T}$-satisfiable, then the $\mathcal{T}$-solver returns a $\mathcal{T}$-model $\mathcal{M}$, which therefore satisfies φ. Otherwise, the $\mathcal{T}$-solver returns a subset of μ that causes the $\mathcal{T}$-unsatisfiability, so that the SAT-solver can learn a new $\mathcal{T}$-lemma to be used in subsequent iterations to prune the search space.

To maximize efficiency, most $\mathcal{T}$-solvers can be called incrementally via a stack-based interface, keeping the status of the search between calls. E.g., an efficient, incremental $\mathcal{LRA}$-solver can be built on a variant of the Simplex algorithm designed to be integrated within a lazy SMT framework [75]. The combination of theories can be handled efficiently by delayed theory combination [29].

Another important feature of CDCL-based SMT solvers is that they provide a stack-based incremental interface, which allows clauses to be pushed and popped and the satisfiability of the formula conjoined with the pushed clauses to be checked incrementally while maintaining most of the learned information between calls.

1.3 SMT Enumeration ($\mathcal{T}$-AllSAT)

AllSAT is the task of enumerating all the truth assignments propositionally satisfying a Boolean formula. A generalization to the SMT case is $\mathcal{T}$-AllSAT, defined as the task of enumerating all the *$\mathcal{T}$-satisfiable* truth assignments propositionally satisfying a $\mathcal{T}$-formula.[1] In the following, we present definitions and algorithms referring to $\mathcal{T}$-AllSAT. The AllSAT case can be obtained as a special case of $\mathcal{T}$-AllSAT when all atoms are Boolean and no reference to $\mathcal{T}$-satisfiability is needed.

1.3.1 Problem Definition

Different versions of $\mathcal{T}$-AllSAT can be defined, depending on whether:

- truth assignments are *total* or *partial*;
- truth assignments are on all atoms or *projected* on a subset of atoms;
- in the case of partial truth assignments, they are pairwise mutually inconsistent (*disjoint*) or not (*non-disjoint*).

[1] In the literature, this task is also referred to as "AllSMT" [48, 160, 161, 204, 205, 207, 150, 50, 123]. In this dissertation, we prefer "$\mathcal{T}$-AllSAT" to "AllSMT" to avoid confusion, because we are enumerating $\mathcal{T}$-satisfiable truth assignments rather than theory-specific FOL models.

In the following, we give definitions for all these versions and discuss their relationships, as well as relevant subcases. In Section 1.3.2, we describe a general algorithm that can be used to solve all these versions, by suitably instantiating its parameters.

Total $\mathcal{T}$-AllSAT. Given a formula $\varphi[\boldsymbol{\alpha}]$, *total $\mathcal{T}$-AllSAT* is the task of enumerating the set $\mathcal{TTA}(\varphi) \stackrel{\text{def}}{=} \{\eta_1^{\boldsymbol{\alpha}}, \ldots, \eta_M^{\boldsymbol{\alpha}}\}$ of all $\mathcal{T}$-satisfiable total truth assignments on $\boldsymbol{\alpha}$ propositionally satisfying φ.

Partial $\mathcal{T}$-AllSAT. Given a formula $\varphi[\boldsymbol{\alpha}]$, *partial $\mathcal{T}$-AllSAT* is the task of enumerating a set $\mathcal{TA}(\varphi) \stackrel{\text{def}}{=} \{\mu_1^{\boldsymbol{\alpha}}, \ldots, \mu_N^{\boldsymbol{\alpha}}\}$ of $\mathcal{T}$-satisfiable partial truth assignments on $\boldsymbol{\alpha}$ s.t.:

(i) every $\mu_i^{\boldsymbol{\alpha}} \in \mathcal{TA}(\varphi) \models_{\mathbb{B}} \varphi$;

(ii) every $\eta_j^{\boldsymbol{\alpha}} \in \mathcal{TTA}(\varphi)$ is an extension of some $\mu_i^{\boldsymbol{\alpha}} \in \mathcal{TA}(\varphi)$;

and, only in the *disjoint* case:

(iii) every pair $\mu_i^{\boldsymbol{\alpha}}, \mu_j^{\boldsymbol{\alpha}} \in \mathcal{TA}(\varphi)$ assigns opposite truth value to at least one atom.

Whereas $\mathcal{TTA}(\varphi)$ is unique, multiple $\mathcal{TA}(\varphi)$ sets are admissible for the same formula φ, including $\mathcal{TTA}(\varphi)$. Typically, $\mathcal{T}$-AllSAT solvers aim at *enumerating a set $\mathcal{TA}(\varphi)$ which is as small as possible*, since every partial truth assignment prevents the enumeration of a number of total truth assignments that is exponential w.r.t. the number of unassigned atoms, thereby saving substantial computational time and space. In particular, it is desirable for the $\mu_i^{\boldsymbol{\alpha}}$ to be *minimal*, meaning that no literal can be dropped from them without losing properties (i) and (iii) (if disjointness is required).

Projected total $\mathcal{T}$-AllSAT. Let $\varphi[\boldsymbol{\alpha}]$ be a formula on $\boldsymbol{\alpha}$, and let $\boldsymbol{\alpha}$, $\boldsymbol{\alpha'}$, $\boldsymbol{\alpha''}$, $\mathbf{B}$, $\mathbf{y}$, and $\mathbf{f}$ be as in Definition 1.2.2, where $\mathbf{B}$, $\mathbf{y}$, and $\mathbf{f}$ are, respectively, the sets of Boolean atoms, first-order variables, and uninterpreted function symbols in $\boldsymbol{\alpha''}$. Following [129], *projected total $\mathcal{T}$-AllSAT* is the task of enumerating a set $\mathcal{TTA}(\exists\mathbf{B}.\exists\mathbf{y}.\exists\mathbf{f}.\varphi) \stackrel{\text{def}}{=} \{\eta_1^{\boldsymbol{\alpha'}}, \ldots, \eta_M^{\boldsymbol{\alpha'}}\}$ of $\mathcal{T}$-satisfiable *total* truth assignments on $\boldsymbol{\alpha'}$ s.t.:

(i) every $\eta_i^{\boldsymbol{\alpha'}} \models_{\mathbb{B}} \exists\mathbf{B}.\exists\mathbf{y}.\exists\mathbf{f}.\varphi$ (see Definition 1.2.2);

(ii) every $\eta_j^{\boldsymbol{\alpha}} \in \mathcal{TTA}(\varphi)$ is an extension of some $\eta_i^{\boldsymbol{\alpha'}}$.

Projected partial $\mathcal{T}$-AllSAT. Given $\varphi[\boldsymbol{\alpha}]$, $\boldsymbol{\alpha}$, $\boldsymbol{\alpha'}$, $\boldsymbol{\alpha''}$, $\mathbf{y}$, $\mathbf{B}$, and $\mathbf{f}$ as above, *projected partial $\mathcal{T}$-AllSAT* is the task of enumerating a set $\mathcal{TA}(\exists\mathbf{B}.\exists\mathbf{y}.\exists\mathbf{f}.\varphi) \stackrel{\text{def}}{=} \{\mu_1^{\boldsymbol{\alpha'}}, \ldots, \mu_N^{\boldsymbol{\alpha'}}\}$ of $\mathcal{T}$-satisfiable *partial* truth assignments on $\boldsymbol{\alpha'}$ s.t. properties (i) and (ii) above hold (by replacing $\eta^{\boldsymbol{\alpha'}}$ with $\mu^{\boldsymbol{\alpha'}}$). Moreover, only in the *disjoint* case:

(iii) every pair $\mu_i^{\boldsymbol{\alpha'}}, \mu_j^{\boldsymbol{\alpha'}}$ assigns opposite truth value to at least one atom.

We note that this formulation generalizes all the previously discussed cases: when every $\mu_i^{\boldsymbol{\alpha'}}$ is a total assignment, we recover the total version(s), and when $\boldsymbol{\alpha'} = \boldsymbol{\alpha}$, we recover the non-projected version(s).

Example 1.3.1 ($\mathcal{T}$-AllSAT). Let $\boldsymbol{\alpha} \stackrel{\text{def}}{=} \mathbf{A} \cup \mathbf{B} \cup \boldsymbol{\alpha}^{\mathcal{LRA}}$, where $\mathbf{A} \stackrel{\text{def}}{=} \{A_1\}, \mathbf{B} \stackrel{\text{def}}{=} \{B_1\}$ are sets of Boolean atoms, and $\boldsymbol{\alpha}^{\mathcal{LRA}} \stackrel{\text{def}}{=} \{(y \leq 1), (y \geq 2)\}$ is a set of $\mathcal{LRA}$-atoms on y. Notice that $\mathbf{A}, \mathbf{B}$, and $\boldsymbol{\alpha}^{\mathcal{LRA}}$ are pairwise symbol-disjoint.

Let $\varphi[\boldsymbol{\alpha}] \stackrel{\text{def}}{=} (A_1 \vee (y \leq 1)) \wedge (\neg A_1 \vee B_1) \wedge (B_1 \leftrightarrow (y \geq 2))$. Then:

$$\begin{aligned}\mathcal{TTA}(\varphi) =& \{A_1 \wedge B_1 \wedge \neg(y \leq 1) \wedge (y \geq 2), \neg A_1 \wedge \neg B_1 \wedge (y \leq 1) \wedge \neg(y \geq 2)\}\\ \mathcal{TA}(\varphi) =& \{A_1 \wedge B_1 \wedge \quad\quad\quad (y \geq 2), \neg A_1 \wedge \neg B_1 \wedge (y \leq 1) \wedge \neg(y \geq 2)\}.\end{aligned}$$

All other total assignments propositionally satisfying φ assign both $(y \leq 1)$ and $(y \geq 2)$ to true, and are thus $\mathcal{LRA}$-unsatisfiable. The *projected $\mathcal{T}$-AllSAT* on $\mathbf{A}$ is

$$\mathcal{TTA}(\exists y.\exists B_1.\varphi) = \mathcal{TA}(\exists y.\exists B_1.\varphi) = \{A_1, \neg A_1\}.$$

In fact,

- for $\mu^{\mathbf{A}} = \quad A_1$, let $\mu^{\mathbf{B}} = \quad B_1$ and $\mu^{\mathcal{LRA}} = \neg(y \leq 1) \wedge \quad (y \geq 2)$;
- for $\mu^{\mathbf{A}} = \neg A_1$, let $\mu^{\mathbf{B}} = \neg B_1$ and $\mu^{\mathcal{LRA}} = \quad (y \leq 1) \wedge \neg(y \geq 2)$.

These cases correspond to the two elements in $\mathcal{TTA}(\varphi)$; in both cases, $\mu^{\mathbf{A}} \wedge \mu^{\mathbf{B}} \wedge \mu^{\mathcal{LRA}} \models_{\mathbb{B}} \varphi$ and $\mu^{\mathcal{LRA}}$ is $\mathcal{LRA}$-satisfiable. The *projected $\mathcal{T}$-AllSAT* on $\mathbf{A} \cup \boldsymbol{\alpha}^{\mathcal{LRA}}$ is

$$\begin{aligned}\mathcal{TTA}(\exists B_1.\varphi) =& \{A_1 \wedge \neg(y \leq 1) \wedge (y \geq 2), \quad \neg A_1 \wedge (y \leq 1) \wedge \neg(y \geq 2)\},\\ \mathcal{TA}(\exists B_1.\varphi) =& \{A_1 \wedge \quad\quad\quad (y \geq 2), \quad \neg A_1 \wedge (y \leq 1) \wedge \neg(y \geq 2)\}.\end{aligned}$$

Remark 1.3.1. We remark that *partial* $\mathcal{T}$-AllSAT may be dramatically faster than total $\mathcal{T}$-AllSAT, since a single $\mathcal{T}$-satisfiable partial assignment represents up to 2^k total ones, where k is the number of unassigned atoms.

For the same reason, *projected* $\mathcal{T}$-AllSAT may be dramatically faster than non-projected $\mathcal{T}$-AllSAT, as a single $\mathcal{T}$-satisfiable assignment on $\boldsymbol{\alpha}'$ represents up to 2^k $\mathcal{T}$-satisfiable assignments on $\boldsymbol{\alpha}$, where k is the number of non-projection atoms in $\boldsymbol{\alpha} \setminus \boldsymbol{\alpha}'$.

Enumeration for non-CNF formulas. For standard SMT *solving*, SMT solvers and enumerators typically assume that the input formula is in CNF (see Section 1.2). If it is not, it is converted into an equisatisfiable CNF formula via either $\mathsf{CNF_{Ts}}$ or $\mathsf{CNF_{PG}}$.

For SMT *enumeration*, if $\varphi[\boldsymbol{\alpha}]$ is not in CNF, then $\mathcal{TA}(\varphi)$ can be computed by first converting φ into an equisatisfiable CNF formula $\psi[\boldsymbol{\alpha} \cup \mathbf{B}]$ via either $\mathsf{CNF_{Ts}}$ or $\mathsf{CNF_{PG}}$, and then by enumerating $\mathcal{TA}(\exists \mathbf{B}.\psi)$, where $\mathbf{B}$ is the set of fresh Boolean atoms introduced by the CNF transformation (see Section 1.2). Such a set can be enumerated by means of $\mathcal{T}$-AllSAT projected on the original atoms $\boldsymbol{\alpha}$ (see Section 1.3.1).

1.3.2 $\mathcal{T}$-AllSAT Computation

Algorithm 1 Projected-$\mathcal{T}$-AllSAT($\varphi[\boldsymbol{\alpha}]$, $\boldsymbol{\alpha'}$)
Input: $\varphi[\boldsymbol{\alpha}]$: a CNF $\mathcal{T}$-formula; $\boldsymbol{\alpha'}$: set of atoms to project on, s.t. $\boldsymbol{\alpha'} \subseteq \boldsymbol{\alpha}$
Output: $M_{\boldsymbol{\alpha'}}$: set of truth assignments; $Cl_{\boldsymbol{\alpha}}$: set of $\mathcal{T}$-lemmas found during enumeration

```
1:  M_α' ← ∅
2:  ψ_0 ← φ
3:  i ← 0
4:  while SMT.Solve(ψ_i[α]) = SAT do
5:      η_i^α ← SMT.GetAssignment()          {find a T-sat truth assignment on α for φ}
6:      μ_i^α ← MinimizeAssignment(ψ_i, η_i^α, α')          {minimize η^α w.r.t. α'}
7:      μ_i^α' ← Project(μ_i^α, α')          {keep literals on α' only}
8:      M_α' ← M_α' ∪ {μ_i^α'}
9:      ψ_{i+1} ← ψ_i ∧ ¬μ_i^α'          {block μ_i^α'}
10:     i ← i + 1
11: end while
12: Cl_α ← SMT.GetLemmas()          {get the learned T-lemmas}
13: return M_α', Cl_α
```

Algorithm 1 describes a procedure for computing projected partial $\mathcal{T}$-AllSAT, as implemented, e.g., in MathSAT [48]. The procedure can be easily adapted to compute the other versions of $\mathcal{T}$-AllSAT, as we detail in the following. In particular, we denote by Projected-$\mathcal{T}$-AllSAT$_{\mathsf{total}}$ the version of Algorithm 1 for projected total $\mathcal{T}$-AllSAT, and by Projected-$\mathcal{T}$-AllSAT$_{\mathsf{partial}}$ the version for projected partial $\mathcal{T}$-AllSAT.

The algorithm takes as input a formula $\varphi[\boldsymbol{\alpha}]$ and a set $\boldsymbol{\alpha'}$ of projection atoms and enumerates a set $M_{\boldsymbol{\alpha'}}$ of truth assignments on $\boldsymbol{\alpha'}$. We note that, in general, the algorithm does not assume any particular relationship between $\boldsymbol{\alpha'}$ and $\boldsymbol{\alpha}$, except that $\boldsymbol{\alpha'} \subseteq \boldsymbol{\alpha}$; in particular, it does not require $\boldsymbol{\alpha'}$ and $\boldsymbol{\alpha''} \stackrel{\text{def}}{=} \boldsymbol{\alpha} \setminus \boldsymbol{\alpha'}$ to be symbol-disjoint, as in Definition 1.2.2. However, if this condition is not satisfied, the set of enumerated truth assignments is not easily characterized.

The algorithm is based on the blocking-clause approach, which iteratively checks for the satisfiability of ψ_i, i.e., φ conjoined with the negation of the previously found truth assignments, until it becomes unsatisfiable (lines 4-11).

In each iteration, it first finds a $\mathcal{T}$-satisfiable total truth assignment $\eta_i^{\boldsymbol{\alpha}}$ on all the atoms $\boldsymbol{\alpha}$, which propositionally satisfies $\psi_i[\boldsymbol{\alpha}]$ (line 5). This guarantees that all its subassignments are also $\mathcal{T}$-satisfiable.

The assignment is then minimized (line 6) by removing literals over $\boldsymbol{\alpha'}$ from $\eta_i^{\boldsymbol{\alpha}} \stackrel{\text{def}}{=} \eta_i^{\boldsymbol{\alpha'}} \wedge \eta_i^{\boldsymbol{\alpha''}}$, to find a partial assignment $\mu_i^{\boldsymbol{\alpha}} \stackrel{\text{def}}{=} \mu_i^{\boldsymbol{\alpha'}} \wedge \eta_i^{\boldsymbol{\alpha''}} \subseteq \eta_i^{\boldsymbol{\alpha}}$; for disjoint enumeration, $\mu_i^{\boldsymbol{\alpha'}}$ is such that $\mu_i^{\boldsymbol{\alpha'}} \models_{\mathbb{B}} \psi_i$ to satisfy (iii), whereas for non-disjoint enumeration, $\mu_i^{\boldsymbol{\alpha'}}$ is only required to be such that $\mu_i^{\boldsymbol{\alpha'}} \models_{\mathbb{B}} \varphi$. We describe an example of a minimization procedure in the next paragraph. For total (projected) $\mathcal{T}$-AllSAT, this step is omitted and $\mu_i^{\boldsymbol{\alpha}} = \eta_i^{\boldsymbol{\alpha}}$. Then, the assignment is projected on $\boldsymbol{\alpha'}$ (line 7), obtaining $\mu_i^{\boldsymbol{\alpha'}}$ satisfying (i). If $\boldsymbol{\alpha'} = \boldsymbol{\alpha}$, then $\mu_i^{\boldsymbol{\alpha'}} = \mu_i^{\boldsymbol{\alpha}}$, and projected $\mathcal{T}$-AllSAT reduces to $\mathcal{T}$-AllSAT.

Algorithm 2 MinimizeAssignment$(\psi_i, \eta_i, \boldsymbol{\alpha'})$
Input: ψ_i: CNF formula; η_i: total truth assignment s.t. $\eta_i \models_{\mathbb{B}} \psi_i$; $\boldsymbol{\alpha'}$: projection atoms
$\psi_i[\boldsymbol{\alpha}] \stackrel{\text{def}}{=} \varphi \wedge \bigwedge_{j=1}^{i-1} \neg\mu_j^{\boldsymbol{\alpha'}}$ if disjoint, $\psi_i \stackrel{\text{def}}{=} \varphi$ if non-disjoint
$\eta_i^{\boldsymbol{\alpha}} = \eta_i^{\boldsymbol{\alpha'}} \cup \eta_i^{\boldsymbol{\alpha''}}$, where $\boldsymbol{\alpha''} \stackrel{\text{def}}{=} \boldsymbol{\alpha} \setminus \boldsymbol{\alpha'}$
Output: minimal partial truth assignment $\mu_i^{\boldsymbol{\alpha'}}$ s.t. $\mu_i^{\boldsymbol{\alpha'}} \cup \eta_i^{\boldsymbol{\alpha''}} \models_{\mathbb{B}} \psi_i$

1: $\mu_i^{\boldsymbol{\alpha'}} \leftarrow \eta_i^{\boldsymbol{\alpha'}}$
2: **for** $\ell \in \mu_i^{\boldsymbol{\alpha'}}$ **do**
3: **if** $\psi_i|_{\mu_i^{\boldsymbol{\alpha'}} \setminus \{\ell\} \cup \eta_i^{\boldsymbol{\alpha''}}} = \top$ **then**
4: $\mu_i^{\boldsymbol{\alpha'}} \leftarrow \mu_i^{\boldsymbol{\alpha'}} \setminus \{\ell\}$
5: **end if**
6: **end for**
7: **return** $\mu_i^{\boldsymbol{\alpha'}}$

The projected assignment $\mu_i^{\boldsymbol{\alpha'}}$ is then added to the set $M_{\boldsymbol{\alpha'}}$ (line 8), and it is blocked by conjoining the *blocking clause* $\neg\mu_i^{\boldsymbol{\alpha'}}$ to ψ_i (line 9). (In actual implementations, $\neg\mu_i^{\boldsymbol{\alpha'}}$ may be fed to the conflict-analysis mechanism of the SAT solver, and the resulting conflict clause may be used as a blocking clause [129]; in Algorithm 1 we omit this step to simplify the narration.) The loop terminates when ψ_i is unsatisfiable, which guarantees property (ii).

Finally, we return the set of enumerated assignments $M_{\boldsymbol{\alpha'}}$. We also return the set of $\mathcal{T}$-lemmas $Cl_{\boldsymbol{\alpha}}$ learned by the SMT solver during the enumeration (line 12), since we will use this algorithm as a subroutine for enumerating $\mathcal{T}$-lemmas in Chapter 8 for knowledge compilation modulo theories.

We remark that the above procedure is provided only as an example and that other procedures can be used to enumerate the truth assignments, including those not based on blocking clauses, such as those based on chronological backtracking [206, 207].

Minimization of truth assignments. A possible implementation of the assignment-minimization procedure is outlined in Algorithm 2. The procedure consists of iteratively dropping literals one by one from $\eta_i^{\boldsymbol{\alpha}}$, checking if it still satisfies the formula. Each minimization step is $O(\#clauses \cdot \#vars)$.

Notice that, since we are in the context of *projected* AllSAT, the minimization algorithm only minimizes the projection atoms in $\boldsymbol{\alpha'}$, and the truth value of existentially quantified atoms in $\boldsymbol{\alpha''}$ is still used to check the satisfiability of the formula by the current partial assignment. Moreover, in the disjoint case, to enforce the pairwise disjointness between the assignments, ψ_i in Algorithm 2 refers to the original formula conjoined with all current blocking clauses $\bigwedge_{j=1}^{i-1} \neg\mu_j^{\boldsymbol{\alpha}}$, whereas in the non-disjoint case ψ_i refers to the original formula only, allowing for a more effective minimization while renouncing the disjointness property.

The resulting partial assignment is *minimal*, i.e., it cannot be further reduced without violating properties (i) and (iii) (if disjointness is required). Note that the order of literal removal can have a significant impact on the effectiveness of the reduction procedure.

1.4 Weighted Model Integration (WMI)

In the context of probabilistic reasoning, events can be expressed as $\mathcal{LRA}$-formulas φ over a set of random variables. In particular, it is often necessary to compute "the probability that φ is satisfied", according to some given probability distribution. *Weighted Model Integration (WMI)* [11] is a general framework for computing such probabilities. The paradigm extends Weighted Model Counting (WMC) [41] to the case of $\mathcal{LRA}$-formulas.

1.4.1 Problem Definition

For some positive integers N and M, let $\mathbf{x} \stackrel{\text{def}}{=} \{x_1, \dots, x_N\} \in \mathbb{R}^N$ and $\mathbf{A} \stackrel{\text{def}}{=} \{A_1, \dots, A_M\} \in \mathbb{B}^M$. Let φ be an $\mathcal{LRA}$-formula over $\mathbf{x}$ and $\mathbf{A}$, and let w be a non-negative weight function s.t. $w(\mathbf{x}, \mathbf{A}) : \mathbb{R}^N \times \mathbb{B}^M \longmapsto \mathbb{R}^+$. Intuitively, w encodes a (possibly unnormalized) density function over $\mathbf{A}$ and $\mathbf{x}$. Given a total truth assignment $\mu^{\mathbf{A}}$ over $\mathbf{A}$, $w|_{\mu^{\mathbf{A}}}(\mathbf{x})$ is w restricted to the truth values assigned by $\mu^{\mathbf{A}}$.

The *Weighted Model Integral* of w over φ is defined as [161]:

$$\mathsf{WMI}\Big(\varphi, w \;\Big|\; \mathbf{x}, \mathbf{A}\Big) \stackrel{\text{def}}{=} \sum_{\mu^{\mathbf{A}} \in 2^{\mathbf{A}}} \mathsf{WMI}_{\mathsf{nb}}\Big(\varphi|_{\mu^{\mathbf{A}}}, w|_{\mu^{\mathbf{A}}} \;\Big|\; \mathbf{x}\Big), \tag{1.2a}$$

$$\mathsf{WMI}_{\mathsf{nb}}\Big(\varphi, w \;\Big|\; \mathbf{x}\Big) \stackrel{\text{def}}{=} \int_{\varphi(\mathbf{x})} w(\mathbf{x}) \, \mathrm{d}\mathbf{x} \tag{1.2b}$$

where $\mathsf{WMI}_{\mathsf{nb}}\Big(\varphi, w \;\Big|\; \mathbf{x}\Big)$ is the integral of $w(\mathbf{x})$ over the set $\{\mathbf{x} \mid \varphi(\mathbf{x}) \text{ is true}\}$.[2]

Example 1.4.1 (WMI). Let $\mathbf{x} \stackrel{\text{def}}{=} \{x_1, x_2\}$, $\mathbf{A} \stackrel{\text{def}}{=} \{A\}$, and let

$$\begin{aligned}\varphi \stackrel{\text{def}}{=} &(0 \leq x_1 \leq 3) \wedge (0 \leq x_2 \leq 2) \wedge (\neg (x_1 \geq 1) \vee (x_2 \leq 1)) \wedge \\ &(A \rightarrow (x_2 \leq \frac{5}{2} - x_1)) \wedge (\neg A \rightarrow (x_2 \leq \frac{1}{3} x_1)).\end{aligned}$$

According to (1.2), $\mathsf{WMI}\Big(\varphi, w \;\Big|\; \mathbf{x}, \mathbf{A}\Big)$ is computed as

$$\begin{aligned}&\mathsf{WMI}_{\mathsf{nb}}\Big(\varphi|_A, w|_A \;\Big|\; \mathbf{x}\Big) + \mathsf{WMI}_{\mathsf{nb}}\Big(\varphi|_{\neg A}, w|_{\neg A} \;\Big|\; \mathbf{x}\Big) \\ &= \int_{(0 \leq x_1 \leq 3) \wedge (0 \leq x_2 \leq 2) \wedge (\neg (x_1 \geq 1) \vee (x_2 \leq 1)) \wedge (x_2 \leq \frac{5}{2} - x_1)} w|_A(\mathbf{x}) \, \mathrm{d}\mathbf{x} + \int_{(0 \leq x_1 \leq 3) \wedge (0 \leq x_2 \leq 1) \wedge (x_2 \leq \frac{1}{3} x_1)} w|_{\neg A}(\mathbf{x}) \, \mathrm{d}\mathbf{x}.\end{aligned}$$

Support of a weight function. A *support* of a weight function $w(\mathbf{x}, \mathbf{A})$ is any subset of $\mathbb{R}^N \times \mathbb{B}^M$ outside which $w(\mathbf{x}, \mathbf{A}) = 0$. We represent it as an $\mathcal{LRA}$-formula χ over $\mathbf{x}$ and $\mathbf{A}$. Thus, $\mathsf{WMI}\Big(\varphi \wedge \chi, w \;\Big|\; \mathbf{x}, \mathbf{A}\Big) = \mathsf{WMI}\Big(\varphi, w \;\Big|\; \mathbf{x}, \mathbf{A}\Big)$.

[2] "$_{\mathsf{nb}}$" means "no-Booleans".

Feasibly integrable weight functions under $\mathcal{LRA}$ conditions. A conditionless weight function $w(\mathbf{x})$ is *feasibly integrable on* $\mathcal{LRA}$ ($\mathsf{FI}^{\mathcal{LRA}}$) if there exists a procedure to compute $\mathsf{WMI}_{\mathsf{nb}}\big(\mu^{\mathcal{LRA}}, w \;\big|\; \mathbf{x}\big)$ over every convex polytope described by a set $\mu^{\mathcal{LRA}}$ of $\mathcal{LRA}$-literals on $\mathbf{x}$. For example, polynomials are $\mathsf{FI}^{\mathcal{LRA}}$.

A weight function $w(\mathbf{x}, \mathbf{A})$ is *feasibly integrable under $\mathcal{LRA}$-conditions* ($\mathsf{FIUC}^{\mathcal{LRA}}$) iff it can be described by a support formula χ and a set $\mathbf{\Psi} \stackrel{\text{def}}{=} \{\psi_i\}_i$ of $\mathcal{LRA}$-conditions over $\mathbf{x}$ and $\mathbf{A}$, such that every total truth assignment $\mu^{\mathbf{\Psi}}$ to $\mathbf{\Psi}$[3] is such that $w|_{\mu^{\mathbf{\Psi}}}(\mathbf{x})$ is total and $\mathsf{FI}^{\mathcal{LRA}}$ on the region satisfying $(\chi \wedge \mu^{\mathbf{\Psi}})$. Intuitively, the conditions $\mathbf{\Psi}$ partition χ into regions. Each $\mu^{\mathbf{\Psi}}$ identifies one such region, over which $w|_{\mu^{\mathbf{\Psi}}}(\mathbf{x})$ is $\mathsf{FI}^{\mathcal{LRA}}$.

In practice, we assume w.l.o.g. that $\mathsf{FIUC}^{\mathcal{LRA}}$ functions are described as combinations of (i) constants; (ii) variables; (iii) standard arithmetic operators $+, -, \cdot$; (iv) conditionless real-valued functions (e.g., exp); (v) conditional expressions in the form $\mathsf{If}\ \psi_i\ \mathsf{Then}\ t_{1i}\ \mathsf{Else}\ t_{2i}$ whose conditions ψ_i are $\mathcal{LRA}$-formulas and whose terms t_{1i}, t_{2i} are $\mathsf{FIUC}^{\mathcal{LRA}}$. For example, piecewise polynomials are $\mathsf{FIUC}^{\mathcal{LRA}}$.

Example 1.4.2 ($\mathsf{FIUC}^{\mathcal{LRA}}$ [161]). Let $\mathbf{x} \stackrel{\text{def}}{=} \{x_1, x_2\}$, $\mathbf{A} \stackrel{\text{def}}{=} \{A\}$, and let

$$\chi \stackrel{\text{def}}{=} (-1 \leq x_1 \leq 1) \wedge (-1 \leq x_2 \leq 1) \wedge (A \leftrightarrow (x_2 \geq 0))$$
$$w(\mathbf{x}, \mathbf{A}) \stackrel{\text{def}}{=} \mathsf{If}\ x_1 \geq 0\ \mathsf{Then}\ x_1^3\ \mathsf{Else}\ -2x_1 + \mathsf{If}\ A\ \mathsf{Then}\ 3x_2\ \mathsf{Else}\ -x_2^5$$

w is a piecewise polynomial and hence $\mathsf{FIUC}^{\mathcal{LRA}}$. In fact, its value depends on the combination of the truth values of the conditions $\mathbf{\Psi} \stackrel{\text{def}}{=} \{x_1 \geq 0, A\}$, as follows:

$$\begin{array}{lcrlcl}
w|_{\{\ A,\ (x_1\geq 0)\}} & = & x_1^3 + 3x_2 & \text{s.t.} & (0 \leq x_1 \leq 1) & \wedge\ (0 \leq x_2 \leq 1) \\
w|_{\{\ A,\neg(x_1\geq 0)\}} & = & -2x_1 + 3x_2 & \text{s.t.} & (-1 \leq x_1 < 0) & \wedge\ (0 \leq x_2 \leq 1) \\
w|_{\{\neg A,\ (x_1\geq 0)\}} & = & x_1^3 - x_2^5 & \text{s.t.} & (0 \leq x_1 \leq 1) & \wedge\ (-1 \leq x_2 < 0) \\
w|_{\{\neg A,\neg(x_1\geq 0)\}} & = & -2x_1 - x_2^5 & \text{s.t.} & (-1 \leq x_1 < 0) & \wedge\ (-1 \leq x_2 < 0)
\end{array}$$

Each of the four functions above is $\mathsf{FI}^{\mathcal{LRA}}$ on the corresponding region.

A challenge for WMI computation is that, in order to compute (1.2), one needs to decompose the integral into a sum of integrals with no Boolean components, and over regions where the weight function is $\mathsf{FI}^{\mathcal{LRA}}$. Moreover, since each such region can be a non-convex polytope, it may need to be further decomposed into convex polytopes, over which the integral can be computed in closed form.

1.4.2 SMT-Based WMI Computation

WMI-PA [160, 161] is an efficient WMI algorithm based on SMT projected enumera-

[3] Notice that the conditions in $\mathbf{\Psi}$ can be non-atomic. In this case, with a slight abuse of terminology, by "truth assignment to $\mathbf{\Psi}$" we mean a mapping from the formulas in $\mathbf{\Psi}$ to $\mathbb{B}$.

Algorithm 3 WMI-PA(φ, w , $\mathbf{x}, \mathbf{A}$)

1: $\langle \varphi^*, w^*, \mathbf{A}^* \rangle \leftarrow \mathsf{LabelConditions}(\varphi, w, \mathbf{x}, \mathbf{A})$
2: $wmi \leftarrow 0$
3: $M^{\mathbf{A}^*} \leftarrow \mathcal{TTA}(\exists \mathbf{x}.\varphi^*)$
4: **for** $\mu^{\mathbf{A}^*} \in M^{\mathbf{A}^*}$ **do**
5: $\quad \mathsf{Simplify}(\varphi^*|_{\mu^{\mathbf{A}^*}})$
6: $\quad$ **if** $\mathsf{LiteralConjunction}(\varphi^*|_{\mu^{\mathbf{A}^*}})$ **then**
7: $\quad\quad wmi \leftarrow wmi + \mathsf{WMI_{nb}}\big(\varphi^*|_{\mu^{\mathbf{A}^*}}, w^*|_{\mu^{\mathbf{A}^*}} \;\big|\; \mathbf{x}\big)$
8: $\quad$ **else**
9: $\quad\quad M^{\mathcal{LRA}} \leftarrow \mathcal{TA}(\varphi^*|_{\mu^{\mathbf{A}^*}})$
10: $\quad\quad$ **for** $\mu^{\mathcal{LRA}} \in M^{\mathcal{LRA}}$ **do**
11: $\quad\quad\quad wmi \leftarrow wmi + \mathsf{WMI_{nb}}\big(\mu^{\mathcal{LRA}}, w^*|_{\mu^{\mathbf{A}^*}} \;\big|\; \mathbf{x}\big)$
12: $\quad\quad$ **end for**
13: $\quad$ **end if**
14: **end for**
15: **return** wmi

tion. Let $w(\mathbf{x}, \mathbf{A})$ be $\mathsf{FIUC}^{\mathcal{LRA}}$. WMI-PA is based on the fact that [161]:

$$\varphi^* \stackrel{\text{def}}{=} \varphi \wedge \chi \wedge \bigwedge_{k=1}^{K} (B_k \leftrightarrow \psi_k) \tag{1.3a}$$

$$\mathsf{WMI}\big(\varphi, w \;\big|\; \mathbf{x}, \mathbf{A}\big) = \sum_{\mu^{\mathbf{A}^*} \in \mathcal{TTA}(\exists \mathbf{x}.\varphi^*)} \mathsf{WMI_{nb}}\big(\varphi^*|_{\mu^{\mathbf{A}^*}}, w^*|_{\mu^{\mathbf{A}^*}} \;\big|\; \mathbf{x}\big) \tag{1.3b}$$

$$\mathsf{WMI_{nb}}\big(\varphi, w \;\big|\; \mathbf{x}\big) = \sum_{\mu^{\mathcal{LRA}} \in \mathcal{TA}(\varphi)} \int_{\mu^{\mathcal{LRA}}} w(\mathbf{x}) \; \mathrm{d}\mathbf{x} \tag{1.3c}$$

$\mathbf{B} \stackrel{\text{def}}{=} \{B_1, \ldots, B_K\}$ are fresh Boolean atoms, $\mathbf{A}^* \stackrel{\text{def}}{=} \mathbf{A} \cup \mathbf{B}$, and $w^*(\mathbf{x}, \mathbf{A} \cup \mathbf{B})$ is obtained by substituting in $w(\mathbf{x}, \mathbf{A})$ each non-atomic condition ψ_k in $\mathbf{\Psi}$ with B_k.

The pseudocode of WMI-PA is reported in Algorithm 3. First, the problem is transformed by computing $\mathbf{A}^*$, φ^*, and w^* as above (line 1). Then, the set $M^{\mathbf{A}^*} \stackrel{\text{def}}{=} \mathcal{TTA}(\exists \mathbf{x}.\varphi^*)$ is computed (line 3) by projected partial $\mathcal{T}$-AllSAT (see Section 1.3.1). For each Boolean assignment $\mu^{\mathbf{A}^*}$ in $M^{\mathbf{A}^*}$ (lines 4-14), $\varphi^*|_{\mu^{\mathbf{A}^*}}$ is computed and simplified by the Simplify procedure (line 5), which propagates truth values and applies Boolean-simplification rules. If $\varphi^*|_{\mu^{\mathbf{A}^*}}$ is already a conjunction of literals, the algorithm directly computes its contribution to the integral (lines 6-7). Otherwise, $\mathcal{TA}(\varphi^*|_{\mu^{\mathbf{A}^*}})$ is computed by partial $\mathcal{T}$-AllSAT. Each partial assignment $\mu^{\mathcal{LRA}}$ identifies a convex polytope over which the integral of $w^*|_{\mu^{\mathbf{A}^*}}$ is computed and added to the total integral (line 11).

Remark 1.4.1. Importantly, in the implementation of WMI-PA the potentially large sets $M^{\mathbf{A}^*}$ and $M^{\mathcal{LRA}}$ are not generated explicitly; rather, their elements are generated, integrated, and then dropped one-by-one, so as to avoid space blow-up.

1.5 Optimization Modulo Theories (OMT)

Some SMT problems require not only finding a model for a given formula but also optimizing the value of some term according to a given order relation. These problems are referred to as *Optimization Modulo Theories (OMT)*.

1.5.1 Problem Definition

Let $\mathcal{T}$ be a theory admitting some total order relation "$\leq$" over its domain. Let φ be a $\mathcal{T}$-formula, and let cost be a $\mathcal{T}$-term that we call *objective term*. *Optimization Modulo The Theory* $\mathcal{T}$, also *OMT($\mathcal{T}$)*, is the problem of finding a model for φ that minimizes the value of cost according to the order given by $\leq$ (maximization is dual) [195, 18]. To simplify the presentation, we focus on minimization, but the same concepts apply to maximization as well. Notice that, in general, φ can be built on a combination of $\mathcal{T}$ with other theories (OMT($\mathcal{T} \cup \mathcal{T}'$)), and the same concepts apply to such cases [195, 196].

Example 1.5.1 (OMT). Consider the OMT($\mathcal{LRA}$) problem $\langle \varphi, \mathsf{cost} \rangle$ where φ is as in Example 1.2.1, and $\mathsf{cost} \stackrel{\text{def}}{=} -2x$. Then the model $\mathcal{M} \stackrel{\text{def}}{=} \{x \mapsto 3, y \mapsto 0\}$ has $\mathsf{cost} = -6$. A better model of φ is, e.g., $\mathcal{M}' \stackrel{\text{def}}{=} \{x \mapsto 6, y \mapsto 2\}$, which has $\mathsf{cost} = -12$. This model is also the model of φ with minimum cost.

1.5.2 Lazy OMT Solving

A general optimization strategy implemented by state-of-the-art OMT solvers, and typically used for OMT($\mathcal{LRA}$) and OMT($\mathcal{LIRA}$), is the so-called *linear-search* strategy [169, 195, 196]. It consists of solving a sequence of SMT problems where the space of feasible solutions is progressively tightened by learning unit clauses in the form $(\mathsf{cost} < \mathsf{ub})$, ub being the currently known upper bound for cost.

At each iteration, the solver can either find a model $\mathcal{M}$ whose value of cost is smaller than ub, or detect the unsatisfiability of the current formula. In the first case, the solver invokes a $\mathcal{T}$-specific procedure, called $\mathcal{T}$-*minimizer*, to find an optimum model $\mathcal{M}'$ within the truth assignment induced by $\mathcal{M}$. Then, ub is set to $\mathcal{M}'(\mathsf{cost})$, and the search continues. In the second case, the formula has no models with cost lower than ub, and the search terminates as the last model found is optimum.

Alternatively, the solver could also follow a *binary-search* strategy [195]. In this case, a lower and upper bound lb and ub are kept s.t. the optimum model lies in the interval $(\mathsf{lb}, \mathsf{ub}]$. At each iteration, an intermediate value $\mathsf{pivot} \in (\mathsf{lb}, \mathsf{ub}]$ is chosen, and the solver checks whether there exists a model with cost lower than pivot. If so, pivot becomes the new upper bound; otherwise, it becomes the new lower bound. The search terminates when lb and ub are equal, and the last model found is optimum. (In continuous domains, e.g., OMT($\mathcal{LRA}$), in order to guarantee termination, it is necessary to interleave binary-search steps with a linear-search step [195]).

If φ is built on a combination of theories, then the $\mathcal{T}$-minimizer is invoked on $\mu_{\mathcal{T}} \cup \mu_{ed}$

—i.e., the subset of μ containing only the atoms in $\mathcal{T}$ and the interface (dis-)equalities [195, 196]. The implementation of $\mathcal{T}$-minimizers for $\mathcal{LRA}$ and $\mathcal{LIRA}$ is briefly described in the next paragraph.

$\mathcal{T}$-minimizers. A $\mathcal{LRA}$-minimizer [195, 196] can be implemented as a simple extension of the Simplex-based $\mathcal{LRA}$-solver [75]. For $\mathcal{LIRA}$, a minimizer can be built on top of a branch-and-bound $\mathcal{LIRA}$-solver [21, 22, 197], by replacing the $\mathcal{LRA}$-solver with a $\mathcal{LRA}$-solver&minimizer to solve each relaxed subproblem. To find an optimum model within the truth assignment, once a $\mathcal{LIRA}$-model $\mathcal{M}$ is found, a constraint $(\mathsf{cost} < \mathcal{M}(\mathsf{cost}))$ is pushed onto the $\mathcal{LIRA}$-solver, and the search is iteratively refined until no better model exists. An alternative strategy, implemented, e.g., in OptiMathSAT, is the so-called *truncated* optimization [197], where the $\mathcal{LIRA}$-minimizer stops after finding the first model $\mathcal{M}$. Although this model may be suboptimal for the current truth assignment —allowing the same assignment to be found again by the CDCL($\mathcal{T}$) procedure— this approach is typically much faster and remains effective in practice.

> **Remark 1.5.1.** Importantly, the lazy OMT solving approach allows for an *anytime* behavior, i.e., we can interrupt the search at any time and return the best model found so far. This capability is especially valuable in industrial applications where finding the optimum solution may be computationally impractical and it is more important to obtain high-quality solutions quickly.

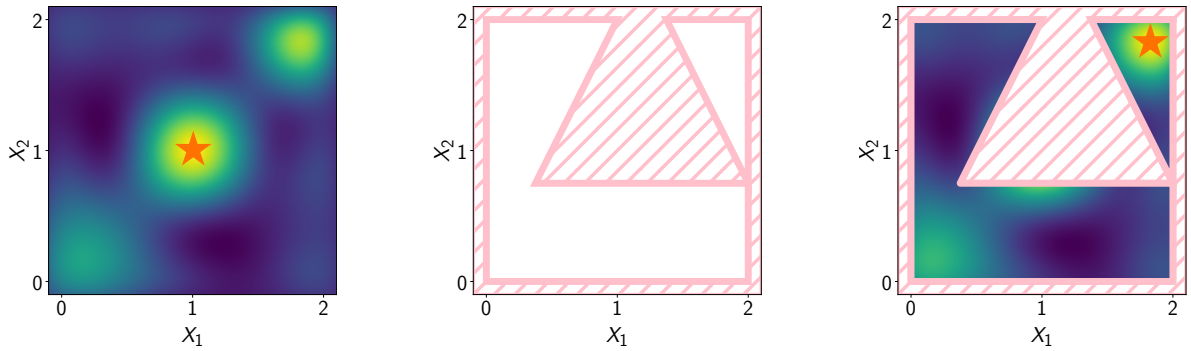
1.6 Constrained Maximum-A-Posteriori (MAP) Inference

Not every optimization problem under SMT constraints can be easily expressed or solved as an OMT problem. For example, in probabilistic inference, we are often interested in finding the most probable assignment to a set of random variables w.r.t. a given probability distribution —usually referred to as *maximum a posteriori (MAP)* [19]— while satisfying an SMT formula. In practice, such distributions are frequently learned with machine-learning techniques, making them highly nonlinear and thus intractable for state-of-the-art OMT solvers. Moreover, some probabilistic models are represented only implicitly. For example, in Variational Autoencoders (VAEs) [119], the marginal density is generally unavailable in closed form, preventing its direct use as an OMT objective.

In this context, we focus on problems over real-valued random variables, where constraints are $\mathcal{LRA}$-formulas and the objective is a probability density function. We refer to this problem as *constrained MAP inference*.

1.6.1 Problem Definition

Notation. Given a set of real variables $\mathbf{x} = \{x_1, \ldots, x_n\}$ and an interpretation $\mathcal{M}$, we denote by $\mathbf{x}^{\mathcal{M}} \stackrel{\text{def}}{=} (\mathcal{M}(x_1), \ldots, \mathcal{M}(x_n))$ the vector value induced by $\mathcal{M}$.



(a) Unconstrained density (b) Feasible region (c) Constrained density

Figure 1.1: An example of MAP($\mathcal{LRA}$) inference over non-convex constraints and a non-log-concave density. Orange stars indicate the solutions for the MAP problem for the unconstrained and constrained densities.

Constrained MAP inference. Given a (possibly unnormalized) probability density $p(\mathbf{x})$ over continuous random variables $\mathbf{x} = \{x_1, \ldots, x_n\}$ and a $\mathcal{LRA}$-formula φ over $\mathbf{x}$, *constrained MAP inference*, also *MAP*($\mathcal{LRA}$), is the problem of finding the model $\mathcal{M}^\star \models \varphi$ inducing the vector $\mathbf{x}^{\mathcal{M}^\star}$ that maximizes the density:[4]

$$\mathcal{M}^\star \stackrel{\text{def}}{=} \arg\max_{\mathcal{M} \models \varphi} p(\mathbf{x}^{\mathcal{M}}). \tag{1.4}$$

Unlike standard OMT, where both the objective and the constraints are handled within the same symbolic framework, constrained MAP inference requires a hybrid approach: φ must be satisfied exactly under standard SMT semantics, whereas p may only be accessible via numerical evaluation or floating-point approximations.

Example 1.6.1 (Constrained MAP). Let $\mathbf{x} = \{x_1, x_2\}$ and let

$$\begin{aligned}\varphi \stackrel{\text{def}}{=} &(0 \leq x_1) \wedge (x_1 \leq 2) \wedge (0 \leq x_2) \wedge (x_2 \leq 2) \wedge \\ &((x_2 \leq 1) \vee (x_2 > 2x_1) \vee (x_2 > 4.75 - 2x_1))\,.\end{aligned}$$

φ defines a square feasible region containing an inner infeasible triangle, as shown in Figure 1.1b. Consider the unnormalized density in Figure 1.1a. The unconstrained optimum is $(1.0, 1.0)$, but the constrained one is $(1.83, 1.83)$, shown in Figure 1.1c.

1.7 Propositional Knowledge Compilation (KC)

For propositional formulas, several problems that are intractable in general can be solved in polynomial time (polytime) if the formula is represented in a suitable compiled form. Given a *propositional* formula φ, the goal of *knowledge compilation* (*KC*) is to transform

[4] If the feasible region defined by φ is not closed, the maximum may not be attained; in such cases, we seek the supremum or its best finite-precision approximation.

φ into a representation that allows for efficient reasoning, i.e., answering queries about φ in polytime [62]. We recall some definitions and results from [62].

1.7.1 Deterministic Decomposable Negation Normal Form (d-DNNF)

Most target compilation languages considered in the literature are variants of NNF (see Section 1.2). In this dissertation, we focus on subsets of NNF obtained by imposing one or more of the following properties:

- *decomposability* [56, 57]: an NNF formula is *decomposable* if every $\wedge$-node $(\psi_1 \wedge \ldots \wedge \psi_k)$ is such that $Atoms(\psi_i) \cap Atoms(\psi_j) = \emptyset$ for every $i \neq j$;
- *determinism* [58]: an NNF formula is *deterministic* if every $\vee$-node $(\psi_1 \vee \ldots \vee \psi_k)$ is such that $\psi_i \wedge \psi_j \models \bot$ for every $i \neq j$;
- *smoothness* [58]: an NNF formula is *smooth* if every $\vee$-node $(\psi_1 \vee \ldots \vee \psi_k)$ is such that $Atoms(\psi_i) = Atoms(\psi_j)$ for every $i \neq j$.

DNNF is the subset of NNF satisfying decomposability [56, 57]; d-NNF is the subset satisfying determinism [58]; sNNF is the subset satisfying smoothness; d-DNNF is the subset satisfying decomposability and determinism [58]; and sd-DNNF is the subset satisfying decomposability, determinism and smoothness [58].

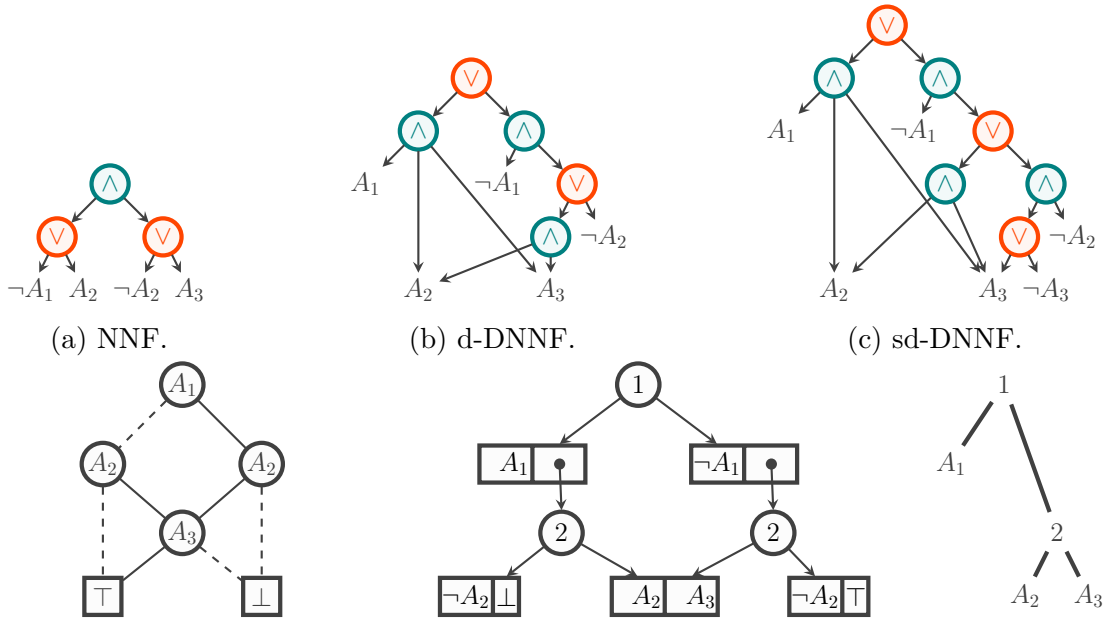
Every d-DNNF φ can be converted into an equivalent sd-DNNF in polytime by applying the transformation $\mathsf{Smooth}(\varphi)$, which replaces bottom-up every non-smooth $\vee$-node $\psi \stackrel{\text{def}}{=} \psi_1 \vee \ldots \vee \psi_K$ with $\bigvee_{i=1}^{K} \psi_i \wedge \bigwedge_{\alpha \in Atoms(\psi) \setminus Atoms(\psi_i)} (\alpha \vee \neg\alpha)$ (Lemma A.1 in [62]). Figure 1.2 shows a graphical representation of an NNF formula (Figure 1.2a) and equivalent d-DNNF (Figure 1.2b) and sd-DNNF (Figure 1.2c) formulas.

d-DNNFs have a number of subclasses, which are obtained by imposing further structural properties on their DAG representation [62]. Notable ones are OBDDs [32] and SDDs [60]. Importantly, under suitable canonicity conditions (discussed next), OBDDs and SDDs are *canonical*: any two propositionally equivalent formulas φ and ψ have syntactically identical OBDD or SDD representations [31].

Ordered Binary Decision Diagrams (OBDDs). A *decision BDD node* is either a constant $\top, \bot$, or is of the form $(A \wedge \psi_1) \vee (\neg A \wedge \psi_2)$, where A is an atom, called the *decision atom*, and ψ_1, ψ_2 are decision BDD nodes representing the "true" and "false" branches, respectively.

An OBDD [32] is a d-DNNF whose nodes are decision BDD nodes and which respects a total atom ordering "$<$": whenever decision node φ_1 with decision atom A_1 is an ancestor of decision node φ_2 with decision atom A_2, then $A_1 < A_2$.

In the rest of the dissertation, by OBDDs we mean *reduced* OBDDs, i.e., OBDDs where the two outgoing branches of each decision node are distinct, and identical subgraphs are merged. For a fixed atom ordering, OBDDs are *canonical*. Figure 1.2d shows a graphical representation of an OBDD.



(a) NNF. (b) d-DNNF. (c) sd-DNNF.

(d) OBDD. Decision BDD nodes are circles labeled with decision atoms; solid and dashed edges denote the true and false branches, respectively.

(e) SDD. Decision SDD nodes are circles labeled with the vtree node; prime-sub pairs $(\varphi_i \wedge \psi_i)$ are represented as paired boxes.

(f) vtree for 1.2e.

Figure 1.2: Different NNF representations for the formula $(\neg A_1 \vee A_2) \wedge (\neg A_2 \vee A_3)$.

Sentential Decision Diagrams (SDDs). SDDs [60] generalize OBDDs by branching on formulas rather than atoms. Let $\mathbf{A}$ and $\mathbf{B}$ be disjoint sets of atoms. An $(\mathbf{A}, \mathbf{B})$-*partition* is a formula $\bigvee_{i=1}^{n}(\varphi_i \wedge \psi_i)$, where the φ_i, called *primes*, are formulas over $\mathbf{A}$ and the ψ_i, called *subs*, are formulas over $\mathbf{B}$; the primes φ_i are satisfiable, mutually exclusive, i.e., $\varphi_i \wedge \varphi_j \models \bot$ for every $i \neq j$, and exhaustive, i.e., $\bigvee_{i=1}^{n} \varphi_i \models \top$.

A *decision SDD node* is either a constant $\top, \bot$, a literal, or an $(\mathbf{A}, \mathbf{B})$-partition where primes and subs are decision SDD nodes. An SDD is a d-DNNF whose nodes are decision SDD nodes and which respects a vtree, i.e., a full binary tree whose leaves are labeled with atoms. An SDD φ respects a vtree v if one of the following holds: (i) φ is a constant; (ii) φ is a literal over atom A and v is a leaf labeled with A; (iii) φ is an $(\mathbf{A}, \mathbf{B})$-partition s.t. every prime respects the left subtree of v, whose leaves are labeled by $\mathbf{A}$, and every sub respects the right subtree of v, whose leaves are labeled by $\mathbf{B}$.

In the rest of the dissertation, by SDDs we mean *compressed*, *trimmed* SDDs, i.e., SDDs where the subs of each partition are pairwise distinct, identical subgraphs are merged, and no partition has the form $(\top \wedge \varphi)$ or $(\varphi \wedge \top) \vee (\neg\varphi \wedge \bot)$. For a fixed vtree, SDDs are *canonical* [60, 31]. Figures 1.2e and 1.2f show a graphical representation of an SDD and its vtree, respectively.

Different target compilation languages support different polytime transformations and queries [62]. We next summarize the transformations and queries supported by d-DNNFs and the subclasses we consider, i.e., OBDDs and SDDs.

1.7.2 Polytime Transformations on d-DNNFs

Transformations. Let $\mathcal{L}$ be a subset of NNF. We say that $\mathcal{L}$ satisfies a transformation if the latter can be performed in *polytime* w.r.t. the size of the input formula and its *output remains in* $\mathcal{L}$. We consider only transformations that are polytime for at least one of the considered languages. Hence, we do not consider forgetting multiple atoms or conjoining and disjoining an unbounded number of formulas [62, 60, 31]. Given $\varphi, \psi \in \mathcal{L}$, we consider:

CD (conditioning). Compute a formula in $\mathcal{L}$ equivalent to $\varphi|_\mu$, for a satisfiable cube μ.

¬C (negation). Compute a formula in $\mathcal{L}$ equivalent to $\neg\varphi$.

SFO (singleton forgetting). Compute a formula in $\mathcal{L}$ equivalent to $\exists A.\varphi$ for $A \in \mathit{Atoms}(\varphi)$.

∧BC (bounded conjunction). Compute a formula in $\mathcal{L}$ equivalent to $\varphi \wedge \psi$.

∨BC (bounded disjunction). Compute a formula in $\mathcal{L}$ equivalent to $\varphi \vee \psi$.

Polytime transformations on d-DNNFs. Table 1.1 reports the transformations satisfied and not satisfied by d-DNNFs, OBDDs, and SDDs. We provide a brief intuition of the positive results, and refer the reader to [62, 31] for a complete discussion.

OBDDs and d-DNNFs satisfy CD [32, 62].[5] For d-DNNFs, replacing literals of μ in φ with Boolean constants (i) trivially preserves NNF; (ii) it preserves decomposability, since if φ_1 and φ_2 do not share any atoms, then so do $\varphi_1|_\mu$ and $\varphi_2|_\mu$; (iii) it preserves determinism, since if $\varphi_1 \wedge \varphi_2 \models \bot$, then $(\varphi_1 \wedge \varphi_2) \wedge \mu \models \bot$; since $(\varphi_1 \wedge \varphi_2) \wedge \mu \equiv (\varphi_1 \wedge \varphi_2)|_\mu \wedge \mu$, and since μ is satisfiable and shares no atom with $(\varphi_1 \wedge \varphi_2)|_\mu$, then it follows that $(\varphi_1 \wedge \varphi_2)|_\mu \equiv (\varphi_1|_\mu \wedge \varphi_2|_\mu) \models \bot$ [62]. For OBDDs, one can apply the same procedure, and then reduce the resulting OBDD, which can be done in polytime [32]. CD is polytime for non-canonical SDDs, but not for canonical ones [31].

OBDDs and SDDs satisfy ¬C [62, 31]. An OBDD can be negated by swapping its $\top$ and $\bot$ leaves [62]. An SDD φ can be negated in polytime by applying the binary "xor" operator to the two SDDs φ and $\top$ [31].

OBDDs satisfy SFO, ∧BC, ∨BC [31].[5] SFO follows from the fact that $\exists A.\varphi \equiv \varphi|_A \vee \varphi|_{\neg A}$, and OBDDs satisfy CD and ∨BC. For ∧BC and ∨BC, see the Apply algorithm in [32], which computes the conjunction or disjunction of two OBDDs in time $O(|\varphi| \cdot |\psi|)$. Hence, iteratively applying this operation N times *is not* polytime.

1.7.3 Polytime Queries on d-DNNFs

Queries. Let $\mathcal{L}$ be a subset of NNF. We say that $\mathcal{L}$ satisfies a query if the query can be performed in *polytime* w.r.t. the size of the input formula and the output of the query. We consider only queries that are polytime for at least one of the considered languages [62, 60, 31]. Let $\varphi, \psi \in \mathcal{L}$.

[5] CD, SFO, ∧BC, and ∨BC are polytime for non-compressed SDDs, which, however, are not canonical [60].

CO (consistency). Decide whether φ is satisfiable.

VA (validity). Decide whether φ is valid.

CE (clausal entailment). Decide whether $\varphi \models C$ for a clause C.

IM (implicant). Decide whether $\mu \models \varphi$ for a cube μ.

CT (counting). Compute the number of total truth assignments satisfying φ.

ME (enumeration). Enumerate all total truth assignments satisfying φ.

EQ (equivalence). Decide whether $\varphi \equiv \psi$.

SE (sentential entailment). Decide whether $\varphi \models \psi$.

Polytime queries on d-DNNFs. Table 1.1 reports the queries satisfied and not satisfied by d-DNNFs, OBDDs, and SDDs. We provide a brief intuition of the positive results, and refer the reader to [62, 31] for a complete discussion.

d-DNNFs satisfy CO, VA, CE, IM, CT, and ME [62].

CO is computed by a bottom-up DAG traversal, returning SAT for a $\top$-node or a literal; UNSAT for a $\bot$-node; by decomposability, SAT for a $\vee$-node if at least one child returns SAT and UNSAT otherwise; and SAT for a $\wedge$-node iff all its children return SAT and UNSAT otherwise [57].

VA reduces to checking whether the model count of φ is equal to $2^{|Atoms(\varphi)|}$. Since d-DNNFs satisfy CT, this can be verified in polytime [58].

CE reduces to checking whether $\varphi|_{\neg C}$ is unsatisfiable. Since d-DNNFs satisfy CD and CO, this can be verified in polytime [57].[6]

IM reduces to checking whether $\varphi|_{\mu}$ is valid. Since d-DNNFs satisfy CD and VA, this can be verified in polytime [62].

CT and ME are easier to prove if we assume the input d-DNNF is smooth, which we recall can be ensured in polytime [62].

CT is computed by a bottom-up DAG traversal, returning 1 for a $\top$-node or a literal; 0 for a $\bot$-node; the sum of the children's values for a $\vee$-node (thanks to determinism); and the product of the children's values for a $\wedge$-node (thanks to decomposability) [58].

ME can be computed by constructing a decision-tree representation of the assignments satisfying φ. Starting with a tree T consisting of a single root node, we consider $Atoms(\varphi) \stackrel{\text{def}}{=} \{A_i\}_i$ in a fixed order, and repeat the following for each leaf node in T, which corresponds to a satisfiable truth assignment μ: (a) if $\varphi|_{\mu \wedge A_i}$ is satisfiable, we add A_i as a child to μ; (b) if $\varphi|_{\mu \wedge \neg A_i}$ is satisfiable, we add $\neg A_i$ as a child to μ. By CD and CO, checks (a) and (b) can be performed in polytime [62]. A more efficient algorithm is proposed in [125], where assignments are enumerated lazily using polynomial space w.r.t. the size of the input.

[6] For SDDs, CD and $\wedge$BC are polytime if the output is allowed to be non-compressed, and hence non-canonical. Thus, for query answering, we can apply the transformation without compression and then check CO in polytime [31].

$\mathcal{L}$	Transformations					Queries							
	CD	**¬C**	**SFO**	**∧BC**	**∨BC**	**CO**	**VA**	**CE**	**IM**	**CT**	**ME**	**EQ**	**SE**
d-DNNF	✓	?	○	○	○	✓	✓	✓	✓	✓	✓	•	•
OBDD	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SDD	•	✓	•	•	•	✓	✓	✓	✓	✓	✓	✓	✓

Table 1.1: Polytime transformations and queries satisfied by different target languages. ✓ indicates that the transformation or query is polytime, • indicates that it is not polytime, and ○ indicates that it is not polytime unless P = NP. ? indicates that it is unknown whether the transformation or query is polytime.

OBDDs and SDDs satisfy EQ and SE [31].

EQ reduces to checking whether the two DAGs are syntactically identical, which can be done in constant time by using hashing techniques to store the DAGs.

SE reduces to checking whether $\varphi \wedge \neg\psi$ is unsatisfiable. For OBDDs, this follows from their polytime CO, ¬C, and ∧BC. For SDDs, a similar argument applies.[6]

1.7.4 Compilation into d-DNNF

Any formula can be converted into an equivalent d-DNNF, though the resulting DAG might be exponentially larger than the input formula in the worst case. Algorithm 4 shows a much simplified procedure for d-DNNF compilation. The algorithm assumes the input formula φ to be in CNF.[7]

It is first invoked as $\mathsf{dDNNFcompile}(\varphi, \top)$ and it works recursively, as a classic DPLL-style AllSAT enumeration procedure, except for the *partitioning step* (lines 10-13): if the current residual formula $\varphi|_\mu$ can be partitioned into the conjunction of $k > 1$ residual formulas $\varphi_1|_\mu, \ldots, \varphi_k|_\mu$ which do not share atoms, then each $\varphi_i|_\mu$ is independently compiled into a d-DNNF, and the resulting d-DNNFs are conjoined. (Notice that in Algorithm 4 the "∧" and "∨" node constructors are implicitly assumed to simplify and propagate the $\top$ and $\bot$ constants: $\psi \wedge \top \Rightarrow \psi$, $\psi \wedge \bot \Rightarrow \bot$, $\psi \vee \top \Rightarrow \top$, $\psi \vee \bot \Rightarrow \psi$.) Actual compilers, e.g., [126], use more sophisticated enumeration schemas (e.g., CDCL-style) and, most importantly, adopt forms of *component caching*: the d-DNNF encodings of all residuals $\varphi|_\mu$ are cached and reused through the search, so that multiple instances of the same d-DNNF subformula are shared, yielding a DAG structure rather than a tree.

[7] If not, the formula can be converted into CNF in linear time by introducing fresh atoms (see Section 1.2); in this case, the resulting d-DNNF must be projected onto the original set of atoms. We omit these steps for simplicity.

Algorithm 4 $\mathsf{dDNNFcompile}(\varphi|_\mu, \mu)$

Input: $\varphi|_\mu$: residual of input formula φ w.r.t. current truth assignment μ (initially φ);
μ: current truth assignment to $Atoms(\varphi)$ (initially $\top$)

Output: $\mathsf{d\text{-}DNNF}(\varphi|_\mu)$

1: **if** $\varphi|_\mu = \top$ **then**
2: **return** $\top$
3: **end if**
4: **if** $\varphi|_\mu = \bot$ **then**
5: **return** $\bot$
6: **end if**
7: **if** (ℓ) is a unit clause in $\varphi|_\mu$ **then**
8: **return** $(\ell \wedge \mathsf{dDNNFcompile}(\varphi|_{\mu\wedge\ell}, \mu \wedge \ell))$
9: **end if**
10: $\{\varphi_1|_\mu, \ldots, \varphi_k|_\mu\} \leftarrow \mathsf{Partition}(\varphi|_\mu)$ {partition $\varphi|_\mu$ into atom-disjoint conjuncts}
11: **if** $k > 1$ **then**
12: **return** $\bigwedge_{i=1}^{k} \mathsf{dDNNFcompile}(\varphi_i|_\mu, \mu)$
13: **end if**
14: $\ell \leftarrow \mathsf{Select}(\varphi|_\mu)$ {Select one literal ℓ on $Atoms(\varphi|_\mu)$ }
15: **return** $(\ell \wedge \mathsf{dDNNFcompile}(\varphi|_{\mu\wedge\ell}, \mu \wedge \ell)) \vee (\neg\ell \wedge \mathsf{dDNNFcompile}(\varphi|_{\mu\wedge\neg\ell}, \mu \wedge \neg\ell))$

Chapter 2

Related Work

2.1 SAT and SMT Enumeration

Applications. SAT and SMT enumeration plays an important role in a variety of applications, ranging from artificial intelligence to formal verification. AllSAT and $\mathcal{T}$-AllSAT, mainly in their disjoint version, play a foundational role in several frameworks for *probabilistic reasoning*, such as model counting in SMT (#SMT) [43] and WMI [11, 161]. Specifically, #SMT($\mathcal{LRA}$) [141, 241, 87, 200] computes the volume of the feasible region of an $\mathcal{LRA}$-formula. It can be computed by summing the volumes of the convex polytopes defined by each of the $\mathcal{LRA}$-satisfiable truth assignments propositionally satisfying the formula. WMI can be seen as a generalization of #SMT($\mathcal{LRA}$) that additionally considers a probability distribution that has to be integrated over each such polytope. See Section 2.3 for applications of #SMT($\mathcal{LRA}$) and WMI.

AllSAT, both disjoint and non-disjoint, also has applications in *data mining*, where the problem of frequent itemsets can be encoded into a Boolean formula whose satisfying assignments are the frequent itemsets [26, 71]. It has also been used for *software test generation* [116], and in *circuit design*, to convert CNF formulas to DNF [154, 153, 14], and for static timing analysis to determine the inputs that can trigger a timing violation in a circuit [83]. AllSAT and $\mathcal{T}$-AllSAT have also been applied in *network verification* for checking network reachability and for analyzing the correctness and consistency of network connectivity policies [139, 111, 138]. Moreover, they have been used for computing images and preimages in *unbounded model checking* [151, 99, 133], and they are also at the core of algorithms for computing *predicate abstraction*, a concept widely used in formal verification for automatically computing finite-state abstractions for systems with potentially infinite state space [128, 53, 129].

AllSAT. Most of the works on AllSAT have focused on the enumeration of satisfying assignments for CNF formulas (e.g., [164, 106, 235, 220, 219, 135, 206]), with several efforts to develop efficient and effective techniques for minimizing partial assignments (e.g., [184, 164, 219]). The problem of minimizing truth assignments for Tseitin-encoded problems was addressed in [108], where the authors proposed to make iterative calls to a SAT solver imposing increasingly tighter cardinality constraints to obtain a minimal

assignment. Although this approach can be used to find a single short truth assignment, it can be very expensive, and thus it is unsuitable for enumeration.

Other works have concentrated on the enumeration of satisfying assignments for combinatorial circuits, exploiting the structural information of the circuits to minimize the partial assignments over the input atoms (e.g., [114, 115, 217, 83, 82]).

In [125], the authors propose a tool for enumerating disjoint partial satisfying assignments of formulas represented as d-DNNFs. The tool efficiently traverses a d-DNNF to enumerate partial assignments using polynomial space. For end-to-end enumeration, however, formulas must first be compiled into d-DNNF, and the shape and size of the resulting d-DNNF significantly affect the effectiveness and efficiency of the enumeration. In particular, non-CNF inputs are typically CNF-ized via the Tseitin transformation before compilation. If compilation is projected onto the original atoms only, the resulting d-DNNF may no longer allow one to enumerate short partial assignments [127]. Later work observed that Tseitin labels can instead be existentially quantified *after* compilation in polynomial time, while preserving both decomposability and determinism [117]. In [66], the authors study how to remove "Tseitin artifacts" resulting from such post-hoc existential quantification, in order to reduce the size of the resulting d-DNNF. More recently, the authors of [67] proposed a circuit-aware compilation approach that directly compiles circuits into d-DNNF, avoiding undesired effects of CNF-ization.

A problem closely related to AllSAT is finding all the prime implicants of a formula (e.g., [181, 109, 140]). AllSAT is a much simpler problem, as it requires finding a subset of implicants (i.e., partial satisfying assignments), not necessarily prime (i.e., minimal), covering all the total assignments satisfying the formula.

Projected AllSAT. Projected enumeration, i.e., enumeration of satisfying assignments over a set of relevant atoms, has been studied mainly for CNF formulas, e.g., in [99, 133, 129, 218, 207]. The ambiguity of the notion of "satisfiability by partial assignment" for non-CNF and existentially-quantified formulas has been raised in [193, 156, 194], highlighting the difference between "evaluation to true", which is simpler to check and typically used by SAT solvers, and "logical entailment", which allows for producing shorter assignments. The approach based on dual reasoning in [155, 157], although able to detect logical entailment and thus to produce shorter partial assignments, is very inefficient even for small formulas.

$\mathcal{T}$-AllSAT. The literature on $\mathcal{T}$-AllSAT is very limited, and $\mathcal{T}$-AllSAT algorithms are largely based on AllSAT techniques and tools. For instance, MathSAT5 [48] implements a $\mathcal{T}$-AllSAT functionality based on the procedure in [129]. A similar procedure has been described in [176]. A $\mathcal{T}$-AllSAT algorithm based on chronological backtracking was recently proposed in [207].

Enumeration in related areas. In Answer Set Programming (ASP), answer sets can be seen as truth assignments satisfying a logic program. ASP programs can be translated into CNF formulas and vice versa, so as to have a one-to-one correspondence between

satisfying assignments and answer sets [52, 136]. Answer sets, however, correspond to *total* truth assignments. Modern ASP solvers are indeed based on AllSAT solvers (e.g., [94, 88, 89]).[1] Thus, our work is not directly applicable to ASP.

Propositional Model Counting (#SAT) is the task of *counting* the number of satisfying assignments of a propositional formula. Counting is simpler than enumeration, since only the number of truth assignments is relevant, but a complete exploration of the space of solutions is still needed. (This is different from #SMT, where each truth assignment is "consumed" to compute a volume.) Efficient algorithms have been developed for #SAT on CNF formulas, mostly based either on DPLL-like exhaustive search, or on Knowledge Compilation (see [97]). Notably, the model counter #CLASP [6] is based on the enumeration of minimal partial assignments.

2.2 CNF Encodings and Enumeration

The role of CNF encodings. Although most AllSAT solvers assume that the formulas are in CNF, little or no work has been done to investigate the impact of the different CNF encodings on their effectiveness and efficiency. The role of CNF transformations has been widely studied for SAT solving (e.g., [27, 110, 20, 122]) and, more recently, for propositional model counting in the context of feature model analysis [122].

Dual-rail encoding. The *dual-rail* encoding [33, 172] has been successfully employed for prime implicant enumeration [181, 140] and recently also for AllSAT on circuits [83].

Given a CNF formula, this encoding maps each atom A_i into a pair of dual-rail atoms $\langle A_i^+, A_i^-\rangle$, s.t. $A_i, \neg A_i$ are substituted with A_i^+, A_i^-, respectively, and $(\neg A_i^+ \vee \neg A_i^-)$ is conjoined with the formula. The resulting CNF formula is equisatisfiable to the original one, and every total assignment η satisfying the dual-rail encoding corresponds to a partial assignment μ satisfying the original formula. Such μ assigns $\mu(A_i)$ to $\top$ or $\bot$ if the pair $\langle \eta(A_i^+), \eta(A_i^-)\rangle$ is $\langle \top, \bot\rangle$ or $\langle \bot, \top\rangle$, respectively, and leaves it unassigned if $\langle \eta(A_i^+), \eta(A_i^-)\rangle = \langle \bot, \bot\rangle$. Short partial assignments can be obtained by maximizing the number of pairs assigned to $\langle \bot, \bot\rangle$, which can be done either exactly by solving a MaxSAT problem, or heuristically by assigning negative value first to decision atoms [83].

2.3 Weighted Model Integration

Motivation and applications. WMI [11, 160] has recently emerged as a unifying formalism for probabilistic inference in hybrid domains, characterized by both continuous and discrete variables and their relationships. These domains are extremely common in real-world scenarios, ranging from transport modeling [101] to probabilistic robotics [216] and cyber-physical systems [131]. Recent applications of WMI include machine learning (ML) with constraints [124], marginal inference on ML models such as Bayesian networks [11] and Density Estimation Trees [204], and probabilistic ML

[1] See also `https://github.com/potassco/clingo/issues/512#issuecomment-2223162682`.

verification [162]. WMI generalizes #SMT($\mathcal{LRA}$), which has been applied to value estimation of probabilistic programs [43] and quantitative program analysis [137].

In the following, we summarize the main approaches for WMI computation. A comprehensive survey can be found in [163].

Exact WMI computation. Exact WMI algorithms primarily leverage advancements in KC [62] or SMT [9].

The use of KC for hybrid probabilistic inference was pioneered by [188] and further refined in a series of later works [121, 72, 120, 78]. KC-based approaches compile the problem into a tractable circuit representation and then compute the WMI, e.g., via repeated symbolic variable elimination [188, 121] or Algebraic Model Counting [120].

Probabilistic inference modulo theories [64] used representations similar to XADDs together with a modified DPLL procedure. The resulting algorithm, called PRAiSE, outperformed previous solutions that did not exploit any SMT-enumeration technique.

Modern SMT-based WMI frameworks [160, 161] rely on SMT enumeration and predicate abstraction [129] to efficiently enumerate models while reducing the total number of polytopes that must be integrated over (see also Section 1.4.2).

On problems with tree-shaped variable dependencies, WMI can be solved exactly and efficiently using a message-passing algorithm [239, 238]. This idea was later extended to approximate WMI on problems with circular dependencies [237].

Component caching techniques, widely used for WMC [187, 7], are difficult to apply to WMI due to the tight coupling induced by arithmetic constraints. Indeed, existing component caching approaches for WMI are restricted to fully-factorized densities with few dependencies among continuous variables [12].

Approximate WMI computation. Approximate WMI algorithms exploit both SMT-based [13] and KC-based [72] approaches. While generally applicable, the latter does not provide any guarantee on the approximation error. Instead, if the constraints can be expressed in disjunctive form, WMI admits a fully polynomial randomized approximation scheme (FPRAS) under mild assumptions [1].

More recently, the authors of [200] proposed an approximate #SMT($\mathcal{LRA}$) algorithm with guarantees based on non-disjoint partial $\mathcal{T}$-AllSAT, combined with an estimate of the overlap between the resulting polytopes.

2.4 Optimization Modulo Theories

Applications. OMT has been successfully applied to a wide range of problems, such as verification of timed and hybrid systems [195, 104], numeric [132] and temporal planning [173, 174], optimal scheduling [23], constrained goal modeling [168], hybrid machine learning [215], GAS optimization for smart contracts [3], and optimum encodings for quantum annealing [15, 70], establishing OMT solvers as powerful tools for solving complex constraint optimization problems in various domains.

State of the art in OMT. OMT techniques have been developed for $\mathcal{LRA}$ [22, 196], $\mathcal{LIRA}$ [22, 197], $\mathcal{BV}$ [165, 221], and $\mathcal{FP}$ [221]. One of the main challenges in OMT is the extension to $\mathcal{NRA}$ and $\mathcal{NIA}$, which is still an active area of research. OptiMathSAT [198] uses an incomplete approach, solving a linear overapproximation of the original nonlinear problem that is iteratively refined [18]. Recently, the authors of [113] proposed CDCL-OCAC, a complete algorithm for OMT($\mathcal{NRA}$) based on a variant of CAD, hence suffering from the same scalability limitations.

OMT has been extended to deal with multiple objectives including lexicographic OMT [22, 197], boxed OMT [134, 22, 197], min-max OMT [198], and Pareto OMT [22].

Recently, a Generalized OMT calculus has been proposed, extending the definition to objectives over partially ordered sets [223].

Lazy OMT solving and enumeration. The idea of using partial assignments in OMT($\mathcal{LIRA}$) has been previously considered in [142, 21]. In [21], the authors mention that in lazy OMT-solving, the truth assignments should preferably be prime implicants. A similar idea was proposed in [142], where the truth assignments are reduced to prime implicants before invoking the $\mathcal{T}$-solver and $\mathcal{T}$-minimizer. None of these works focuses on ad-hoc truth assignment reduction techniques for the optimization setting.

2.5 Constrained MAP Inference

Applications. MAP inference is a fundamental problem in machine learning, as it allows one to compute the output configuration with the highest likelihood according to the distribution learned by the model [19]. Constrained MAP inference is mandatory if the model is deployed in safety-critical scenarios, where some configurations are infeasible and should be neither predicted nor sampled [93, 25]. Other applications include physics [102, 42], fair predictions [92], and "what-if" time-series analysis [166].

Solving MAP($\mathcal{LRA}$) exactly is generally computationally hard, as even optimizing a quadratic function over a convex polytope is NP-hard [186]. Nevertheless, efficient methods for specific problem classes exist, each trading off expressiveness, efficiency, and optimality guarantees in a different way.

Convex constraints, log-concave densities. If the constraint φ is *convex*, e.g., it is defined as a *conjunction* of linear constraints, and the objective is *(log-)concave*, any local optimum is also a global optimum. This property ensures that algorithms such as sequential quadratic programming [24] and interior-point methods [55] are guaranteed to efficiently converge to the global optimum.

Convex constraints and piecewise densities. When the density decomposes into simple pieces, one can go beyond log-concavity and use symbolic bucket elimination for constrained piecewise functions as in [234]. The latter tackles the problem of optimizing a sum of piecewise linear or univariate quadratic functions, i.e., only one variable can be quadratic in each piece, under convex linear constraints. The approach is based on

XADDs [189] and leverages the symbolic maximization algorithm of [236]. A similar line of work by [112] deals with mixed-integer linear programming over convex linear constraints that decompose into two sets of constraints over disjoint sets of variables.

General polynomial densities. For non-convex problems, exact global constrained optimization is notoriously challenging, even if p is a polynomial and φ is a convex polytope. For example, *cylindrical algebraic decomposition* (CAD) [54, 5, 231] has doubly exponential worst-case complexity in the number of variables. A classical way to approximate MAP($\mathcal{LRA}$) with guarantees for polynomial densities over convex polytopes is the Lasserre moment hierarchy [130], which solves a sequence of semidefinite programming (SDP) relaxations. As the relaxation order increases, the solution provides increasingly tight lower and upper bounds, converging to the global optimum in the limit. The dual problem can be solved via *sum-of-squares* optimization (SoS), which also provides a sequence of improving bounds converging to the global maximum.

Local optimizers for general densities and convex constraints. A practical way to approximate MAP($\mathcal{LRA}$) for general densities over convex polytopes is to run a local optimizer starting from multiple initial points (particles) to increase the chances of finding the global optimum. For example, Basin Hopping [229] combines local minimization with stochastic perturbations of the current solution, whereas SHGO [76] exploits a simplicial complex to identify locally convex subdomains corresponding to distinct minima, so to systematically explore the objective landscape. These methods scale well with problem dimensionality and make minimal assumptions about the shape of the constraints or objective function —typically requiring only smoothness or Lipschitz continuity— but provide no guarantees of global optimality in the general case.

Optimization under general SMT constraints. MAP inference under a general $\mathcal{LRA}$-formula can be reduced to OMT [169, 195, 196] when the density can be expressed as a term in a theory supported by an OMT solver, e.g., a piecewise polynomial. For $\mathcal{LRA}$-formulas and objectives, OMT can be solved efficiently [195, 22] (see also Sections 1.5.2 and 2.4). Dealing with nonlinear polynomial objectives, however, is nontrivial and remains an active area of research (see Section 2.4).

The extension of common optimization methods that guarantee feasibility, such as barrier methods [170], to SMT constraints with a complex Boolean structure is not straightforward. Indeed, to the best of our knowledge, no such extension exists in the literature, making these methods not directly applicable to MAP($\mathcal{LRA}$).

Soft-constrained optimization only "encourages" constraint satisfaction by adding a penalty term to the objective [232, 201, 80], without formal guarantees. As the indicator function over the constraints is not differentiable and therefore unsuitable for gradient-based optimization, various differentiable relaxations have been proposed. [232] introduces a semantic loss for propositional logic; [209] encodes physical laws into penalties; and [69] leverages fuzzy logic to translate logical statements into differentiable objectives. For SMT($\mathcal{LRA}$) constraints, DL2 [80] provides a framework to translate algebraic constraints into penalties suitable for optimization.

2.6 Theory-Lemma Enumeration

Motivation and applications. State-of-the-art approaches for SMT solving are based on the lazy approach (see Section 1.2.1). For some harder SMT tasks, such as (minimal) unsat-core extraction [49, 100], MaxSMT [47, 77], and most importantly the KC Modulo Theories framework we present in Chapter 8, it may be beneficial or even necessary to "eagerly" precompute *a whole set of $\mathcal{T}$-lemmas* ruling out all the $\mathcal{T}$-inconsistent truth assignments propositionally satisfying the formula.

Eager encodings. With SMT via eager encodings (e.g., [228, 34, 212, 211]), an input $\mathcal{T}$-formula φ is encoded into SAT by (i) enumerating a complete set of $\mathcal{T}$-lemmas ruling out all possible $\mathcal{T}$-unsatisfiable truth assignments on $Atoms(\varphi)$, (ii) conjoining them to φ and (iii) producing the Boolean abstraction of the result. For SMT solving, these techniques ultimately proved to be much less efficient than the lazy approach [9].[2] Moreover, they are restricted to specific theories (e.g., $\mathcal{EUF}$ [228, 34] and fragments of arithmetic [212, 211]), do not support theory combination, and may produce many unnecessary lemmas.

Lemma-lifting. "Lemma-lifting" approaches lift theory information to the Boolean level on demand, rather than enumerating a complete set of theory lemmas upfront. For unsat-core extraction [49], a lazy SMT solver first proves the formula $\mathcal{T}$-unsatisfiable while generating the necessary theory lemmas; these lemmas are then conjoined to the input formula, and the Boolean abstraction of the result is passed to a propositional core extractor. For MaxSMT [47, 77], the same idea is used iteratively: a MaxSAT solver optimizes over the current Boolean abstraction, enriched with the theory lemmas collected so far, while an SMT solver checks each candidate optimum and, whenever it is $\mathcal{T}$-inconsistent, produces new lemmas that refine the Boolean abstraction.

2.7 Knowledge Compilation Modulo Theories

Applications. KC for SMT has received little attention, and its applications in the literature are still limited. Early work on SMT-level KC mostly focused on theory-satisfiability checking, with several theory-specific variants of OBDDs [96, 95, 34, 84, 225, 8], which have been applied to the verification of hardware [96, 95], software [65, 36], and infinite-state systems [40, 158]. More recently, XSDDs [72, 120] have been used for WMI computation.

The application landscape of propositional KC is substantially broader. This suggests that an efficient SMT-level KC framework (see Chapter 8) could enable the adaptation of successful SAT-level KC applications to the SMT setting. Propositional KC applications

[2] In the SMT literature, the expression "eager encoding" is also used to denote another successful approach to SMT solving for finite-domain theories, such as bit-vectors, floating-point arithmetic or arithmetic over bounded integers, which encodes domain variables via sequences of Boolean atoms, and encodes functions and predicates into Boolean functions accordingly. We emphasize that this is *not* what we refer to here.

include, e.g., probabilistic inference [59, 44, 226, 45], probabilistic logic programming [73, 79], planning [171, 208], feature model reasoning [213, 214], and explainable AI [61].

In the following, we review the main approaches for SMT-level KC and provide a comprehensive summary in Table E.1 in Appendix E.1.

OBDDs for $\mathcal{EUF}$. Two techniques for extending OBDDs to the theory of equality ($\mathcal{E}$) are described in [96, 95]. The first uses a logarithmic encoding of the variables, reducing to the Boolean case. The second approach maps each equality $x_i = x_j$ to a Boolean atom e_{ij} and builds an OBDD over the resulting Boolean formula. These OBDDs can encode theory-inconsistent paths. This issue was addressed in [34] by eagerly conjoining transitivity lemmas with the Boolean formula.

EQ-BDDs [84] use a different approach to leverage OBDDs for $\mathcal{E}$, based on a set of rewriting rules that prune inconsistent paths. EUF-BDDs [225] extend EQ-BDDs to $\mathcal{EUF}$, and $(0, S, =)$-BDDs [8] extend EQ-BDDs to the $(0, S, =)$ theory, involving the constant 0 and the successor function S.

OBDDs and SDDs for arithmetic. Difference Decision Diagrams (DDDs) [158] generalize OBDDs to the theory of difference logic ($\mathcal{DL}$). The approach is based on a set of rewriting and reduction rules that prune inconsistent paths. LDDs [36] generalize DDDs to $\mathcal{LIRA}$, though the implementation is restricted to the $\mathcal{TVPI}$ theory of two real or integer variables per inequality. Since only local reductions are applied, LDDs can encode theory-inconsistent paths.

The procedure in [40] extends OBDDs to $\mathcal{NRA}$, using an incomplete quadratic constraint solver to prune inconsistent paths after compilation.

XSDDs [72, 120] extend SDDs to $\mathcal{LRA}$. No strategy to prune inconsistent paths is used, and simplifications are only done at later stages during WMI computation.

OBDDs for arbitrary theories. The approach in [81] and the HARVEY tool [65] both build the OBDD of the Boolean abstraction of the SMT formula and then identify and prune theory-inconsistent paths by conjoining theory lemmas.

The methods in [35, 46] tightly integrate an OBDD compiler with an SMT solver, which is invoked to check theory consistency during compilation.

d-DNNFs modulo theories. To the best of our knowledge, there is no previous work on leveraging general d-DNNFs to the SMT level, with the exception of [68], which sketches a possible procedure to produce a decision d-DNNF by DPLL-like lazy SMT enumeration (without partitioning, see Section 1.7.4).

In [227], the authors propose a KC framework for WMC for a function-free finite-domain fragment of uninterpreted first-order logic, thus addressing a related problem that is nevertheless substantially different from ours. In contrast, our setting targets SMT, requiring reasoning modulo background theories through theory lemmas.

Part II

Contributions

Chapter 3

On CNF Conversion for SAT and SMT Enumeration

> **Full Disclosure.** This chapter is based on the papers *"On CNF Conversion for Disjoint SAT Enumeration"* [149] and *"On CNF Conversion for SAT and SMT Enumeration"* [150] co-authored by the Ph.D. candidate with Dr. Giuseppe Spallitta, and Prof. Roberto Sebastiani. In both papers, the candidate is the main contributor. His contributions include part of the conceptualization, the core implementation, and the execution of the experimental evaluation.

State-of-the-art SAT and SMT solvers deal very efficiently with formulas expressed in Conjunctive Normal Form (CNF). In real-world scenarios, however, it is common for problems to be expressed as non-CNF formulas. Hence, these problems are converted into CNF before being processed by the solver. This conversion is generally done using variants of the Tseitin [222] or the Plaisted and Greenbaum [178] transformations, which generate a linear-size equisatisfiable CNF formula by labeling subformulas with fresh Boolean atoms (see Section 1.2). These transformations can also be employed for SAT and SMT enumeration (AllSAT and $\mathcal{T}$-AllSAT, respectively), by projecting the truth assignments onto the original atoms only (see Section 1.3.1).

When passing from plain *solving* to *enumeration*, however, the capability of enumerating *partial* satisfying assignments that are as small as possible is crucial, because each prevents the enumeration of a number of total assignments that is exponential w.r.t. the number of unassigned atoms. This raises the question of whether CNF encodings conceived for solving are also effective for enumeration. To the best of our knowledge, however, no research has yet been published investigating how the different CNF encodings may affect the effectiveness of the solvers for AllSAT and $\mathcal{T}$-AllSAT.

Contributions. In this chapter, we investigate both theoretically and empirically the effectiveness of different CNF transformations for SAT and SMT enumeration, both in the disjoint and non-disjoint cases. The contribution of this chapter is twofold.

First, on the negative side, we show that the CNF transformations commonly employed for solving are not suitable for enumeration. In particular, we notice that the Tseitin transformation introduces top-level label definitions for subformulas with double implications, which need to be satisfied as well, and thus prevent the solver from producing short partial assignments. We also notice that the Plaisted and Greenbaum transformation solves this problem only in part by labeling subformulas with one-way implications if they occur with single polarity, but it has similar issues to the Tseitin transformation when subformulas occur with both polarities.

Second, on the positive side, we prove theoretically and show empirically that converting the formula into NNF before applying the Plaisted and Greenbaum transformation can fix the problem and drastically improve the effectiveness of the enumeration process by up to orders of magnitude.

This analysis is confirmed by an experimental evaluation on non-CNF problems originating from both synthetic and real-world-inspired applications. The results confirm the theoretical analysis, showing that the proposed combination of NNF with the Plaisted and Greenbaum CNF transformation allows for a drastic reduction in both the number of partial assignments and the execution time.

Organization. This chapter is organized as follows. In Section 3.1, we analyze the problems caused by classical CNF transformations when used for AllSAT and $\mathcal{T}$-AllSAT. In Section 3.2, we propose one possible solution, whose effectiveness is evaluated on both synthetic and real-world-inspired benchmarks in Section 3.3. Finally, in Section 3.4, we draw our conclusions and outline future research directions.

3.1 The Impact of CNF-izations on Enumeration

In this section, we present a theoretical analysis of the impact of different CNF-izations on the enumeration of short partial truth assignments. In particular, we focus on the Tseitin CNF-ization ($\mathsf{CNF}_{\mathsf{Ts}}$) [222] and on the Plaisted and Greenbaum CNF-ization ($\mathsf{CNF}_{\mathsf{PG}}$) [178]. We point out how CNF-izing the problems using these transformations can introduce unexpected drawbacks for enumeration. In fact, we show that the resulting encodings can force the enumerator to produce partial assignments that are larger than necessary in size and in number. We notice that this problem is a direct consequence of Definition 1.2.2 and Remarks 1.2.1 and 1.2.2. In order to simplify the presentation, we focus on AllSAT, but the analysis applies to $\mathcal{T}$-AllSAT as well by restricting to theory-satisfiable truth assignments. Moreover, the analysis applies to both disjoint and non-disjoint enumeration.

3.1.1 The Impact of Tseitin CNF-ization

We begin by showing that using the $\mathsf{CNF}_{\mathsf{Ts}}$ transformation [222] can be problematic for enumeration. In particular, we point out a fundamental weakness of $\mathsf{CNF}_{\mathsf{Ts}}$.

Fact 3.1.1. *If a partial assignment $\mu^{\mathbf{A}}$ satisfies φ, this does not imply that $\mu^{\mathbf{A}}$ satisfies $\exists\mathbf{B}.\mathsf{CNF}_{\mathsf{Ts}}(\varphi)$.*

We recall that $\mathsf{CNF}_{\mathsf{Ts}}$ works by recursively applying the rewriting step (Section 1.2):

$$\varphi \Longrightarrow \varphi[\varphi_i|B_i] \wedge (B_i \leftrightarrow \varphi_i) \tag{3.1}$$

and then recursively CNF-izing the two conjuncts. A *partial* assignment $\mu^{\mathbf{A}}$ may satisfy a non-CNF formula $\varphi(\mathbf{A})$ because it does not need to assign a truth value to the atoms in *all* subformulas of φ. (E.g., $\mu^{\mathbf{A}}$ can satisfy $\varphi \stackrel{\text{def}}{=} \varphi_1 \vee \varphi_2$ without assigning values to the atoms in φ_2 if it satisfies φ_1.)

Consider (3.1) s.t. φ_i is a subformula of φ whose atoms are not assigned by $\mu^{\mathbf{A}}$. Although $\mu^{\mathbf{A}}$ satisfies φ, it does not satisfy $\exists B_i.(\varphi[\varphi_i|B_i] \wedge (B_i \leftrightarrow \varphi_i))$. Indeed, satisfying the second conjunct requires assigning not only B_i, but also some of the unassigned atoms of φ_i, so that φ_i evaluates to the truth value assigned to B_i.

As a consequence of Fact 3.1.1, given $\mu^{\mathbf{A}}$ satisfying φ, to find a $\mu^{\mathbf{A}'}$ satisfying $\exists\mathbf{B}.\mathsf{CNF}_{\mathsf{Ts}}(\varphi)$, the enumerator is most often forced to assign other atoms in $\mathbf{A}$, so that $\mu^{\mathbf{A}'} \supset \mu^{\mathbf{A}}$. Given that the number of total assignments covered by a partial assignment decreases exponentially with its length (see Section 1.2), the above issue causes a blow-up in the number of partial assignments needed to cover all the satisfying total assignments. This may drastically affect the effectiveness and efficiency of the enumeration.

This is illustrated in the following example, where instead of a single short partial assignment the enumerator is forced to enumerate 9 longer ones.

Example 3.1.1. Consider the Boolean formula over $\mathbf{A} \stackrel{\text{def}}{=} \{A_1, A_2, A_3, A_4, A_5, A_6, A_7\}$:

$$\varphi \stackrel{\text{def}}{=} \overbrace{(A_1 \wedge A_2)}^{B_1} \vee (\overbrace{\overbrace{(\overbrace{(A_3 \vee A_4)}^{B_2} \wedge \overbrace{(A_5 \vee A_6)}^{B_3})}^{B_4} \leftrightarrow A_7}^{B_5}) \,. \tag{3.2}$$

φ is not in CNF, and thus it must be CNF-ized before starting the enumeration process. If $\mathsf{CNF}_{\mathsf{Ts}}$ is used, then the following CNF formula is obtained:

$$\begin{aligned}
&\mathsf{CNF}_{\mathsf{Ts}}(\varphi) \stackrel{\text{def}}{=} \\
&(\neg B_1 \vee \ A_1) \wedge (\neg B_1 \vee \ A_2) \wedge (\ B_1 \vee \neg A_1 \vee \neg A_2) \wedge \ //(B_1 \leftrightarrow (A_1 \wedge A_2)) && \text{(3.3a)}\\
&(\ B_2 \vee \neg A_3) \wedge (\ B_2 \vee \neg A_4) \wedge (\neg B_2 \vee \ A_3 \vee \ A_4) \wedge \ //(B_2 \leftrightarrow (A_3 \vee A_4)) && \text{(3.3b)}\\
&(\ B_3 \vee \neg A_5) \wedge (\ B_3 \vee \neg A_6) \wedge (\neg B_3 \vee \ A_5 \vee \ A_6) \wedge \ //(B_3 \leftrightarrow (A_5 \vee A_6)) && \text{(3.3c)}\\
&(\neg B_4 \vee \ B_2) \wedge (\neg B_4 \vee \ B_3) \wedge (\ B_4 \vee \neg B_2 \vee \neg B_3) \wedge \ //(B_4 \leftrightarrow (B_2 \wedge B_3)) && \text{(3.3d)}\\
&(\neg B_5 \vee \ B_4 \vee \neg A_7) \wedge (\neg B_5 \vee \neg B_4 \vee \ A_7) \qquad \wedge \ //(B_5 \leftrightarrow (B_4 \leftrightarrow A_7)) && \text{(3.3e)}\\
&(\ B_5 \vee \ B_4 \vee \ A_7) \wedge (\ B_5 \vee \neg B_4 \vee \neg A_7) \qquad \wedge \\
&(\ B_1 \vee \ B_5) && \text{(3.3f)}
\end{aligned}$$

where the fresh atoms $\mathbf{B} \stackrel{\text{def}}{=} \{B_1, B_2, B_3, B_4, B_5\}$ label subformulas as in (3.2).

Consider the (minimal) partial truth assignment:

$$\mu^{\mathbf{A}} \stackrel{\text{def}}{=} \{\neg A_3, \neg A_4, \neg A_7\}. \tag{3.4}$$

The assignment $\mu^{\mathbf{A}}$ satisfies φ (3.2), even though it does not assign a truth value to the subformulas $(A_1 \wedge A_2)$ and $(A_5 \vee A_6)$. Yet, $\mu^{\mathbf{A}}$ does not satisfy $\exists\mathbf{B}.\mathsf{CNF}_{\mathsf{Ts}}(\varphi)$. In fact, there is no total truth assignment $\eta^{\mathbf{B}}$ on $\mathbf{B}$ such that $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_{\mathbb{B}} \mathsf{CNF}_{\mathsf{Ts}}(\varphi)$, because (3.3a) and (3.3c) cannot be satisfied by assigning only variables in $\mathbf{B}$; rather, it is necessary to further assign at least one atom in $\{A_1, A_2\}$ to satisfy (3.3a) and at least one in $\{A_5, A_6\}$ to satisfy (3.3c).

Suppose that an enumerator first assigns the literals in $\mu^{\mathbf{A}}$ (3.4), which also force it to assign $\mu^{\mathbf{B}} \stackrel{\text{def}}{=} \{\neg B_2, \neg B_4, B_5\}$ due to (3.3b), (3.3d), and (3.3e), respectively.

Since $\mu^{\mathbf{A}} \cup \mu^{\mathbf{B}}$ satisfies all clauses except those in (3.3a) and (3.3c), the enumerator needs to extend $\mu^{\mathbf{A}} \cup \mu^{\mathbf{B}}$ by enumerating the partial assignments on the unassigned atoms $\{A_1, A_2, A_5, A_6, B_1, B_3\}$ which satisfy (3.3a) and (3.3c). Regardless of the search strategy adopted, this requires generating at least 9 satisfying partial assignments on $\{A_1, A_2, A_5, A_6\}$.[a]

For instance, in the case of disjoint AllSAT, instead of the single partial assignment $\mu^{\mathbf{A}}$ (3.4), the solver may return the following list of 9 partial assignments satisfying $\exists\mathbf{B}.\mathsf{CNF}_{\mathsf{Ts}}(\varphi)$ which extend $\mu^{\mathbf{A}}$:

$$\begin{array}{llll}
\{\overbrace{\neg A_1,}^{B_1} & \neg A_3, \neg A_4, & \overbrace{\neg A_5, \neg A_6,}^{B_3} & \neg A_7\} \; //\{\neg B_1, \neg B_2, \neg B_3, \neg B_4, B_5\} \\
\{\neg A_1, & \neg A_3, \neg A_4, & A_5, & \neg A_7\} \; //\{\neg B_1, \neg B_2, \; B_3, \neg B_4, B_5\} \\
\{\neg A_1, & \neg A_3, \neg A_4, & \neg A_5, \; A_6, & \neg A_7\} \; //\{\neg B_1, \neg B_2, \; B_3, \neg B_4, B_5\} \\
\{\; A_1, \neg A_2, & \neg A_3, \neg A_4, & \neg A_5, \neg A_6, & \neg A_7\} \; //\{\neg B_1, \neg B_2, \neg B_3, \neg B_4, B_5\} \\
\{\; A_1, \neg A_2, & \neg A_3, \neg A_4, & A_5, & \neg A_7\} \; //\{\neg B_1, \neg B_2, \; B_3, \neg B_4, B_5\} \\
\{\; A_1, \neg A_2, & \neg A_3, \neg A_4, & \neg A_5, \; A_6, & \neg A_7\} \; //\{\neg B_1, \neg B_2, \; B_3, \neg B_4, B_5\} \\
\{\; A_1, \; A_2, & \neg A_3, \neg A_4, & \neg A_5, \neg A_6, & \neg A_7\} \; //\{\; B_1, \neg B_2, \neg B_3, \neg B_4, B_5\} \\
\{\; A_1, \; A_2, & \neg A_3, \neg A_4, & A_5, & \neg A_7\} \; //\{\; B_1, \neg B_2, \; B_3, \neg B_4, B_5\} \\
\{\; A_1, \; A_2, & \neg A_3, \neg A_4, & \neg A_5, \; A_6, & \neg A_7\} \; //\{\; B_1, \neg B_2, \; B_3, \neg B_4, B_5\}
\end{array} \tag{3.5}$$

In the case of non-disjoint AllSAT, the solver may enumerate a similar set of partial assignments, with $\{\neg A_2\}$ instead of $\{A_1, \neg A_2\}$ and $\{A_6\}$ instead of $\{\neg A_5, A_6\}$.

[a] The set of models for (3.3a) is $\{\{B_1, A_1, A_2\}, \{\neg B_1, A_1, \neg A_2\}, \{\neg B_1, \neg A_1, A_2\}, \{\neg B_1, \neg A_1, \neg A_2\}\}$, which can be covered only either by $\{\{B_1, A_1, A_2\}, \{\neg B_1, \neg A_1\}, \{\neg B_1, A_1, \neg A_2\}\}$ or by $\{\{B_1, A_1, A_2\}, \{\neg B_1, \neg A_2\}, \{\neg B_1, \neg A_1, A_2\}\}$ in the case of disjoint enumeration, and by $\{\{B_1, A_1, A_2\}, \{\neg B_1, \neg A_1\}, \{\neg B_1, \neg A_2\}\}$ in the case of non-disjoint enumeration. In all cases, we need at least 3 distinct partial assignments on $\{A_1, A_2\}$. Similar considerations hold for (3.3c). Thus, we need at least $3 \times 3 = 9$ partial assignments on $\{A_1, A_2\} \cup \{A_5, A_6\}$.

3.1.2 The Impact of Plaisted and Greenbaum CNF-ization

We point out that the $\mathsf{CNF}_{\mathsf{PG}}$ transformation [178] suffers from the same weakness as $\mathsf{CNF}_{\mathsf{Ts}}$ —that is, Fact 3.1.1 also holds for $\mathsf{CNF}_{\mathsf{PG}}$— although its effects are mitigated. We recall that $\mathsf{CNF}_{\mathsf{PG}}$ works by recursively applying the rewriting step (Section 1.2):

$$\varphi \Longrightarrow \varphi[\varphi_i|B_i] \wedge \left\{ \begin{array}{ll} (B_i \rightarrow \varphi_i) & \text{if } \varphi_i \text{ occurs only positively in } \varphi \\ (B_i \leftarrow \varphi_i) & \text{if } \varphi_i \text{ occurs only negatively in } \varphi \\ (B_i \leftrightarrow \varphi_i) & \text{if } \varphi_i \text{ occurs both positively and negatively in } \varphi \end{array} \right\} \quad (3.6)$$

and then recursively CNF-izing the two conjuncts. As with $\mathsf{CNF}_{\mathsf{Ts}}$, consider (3.6) s.t. φ_i is some subformula of φ whose atoms are not assigned by $\mu^{\mathbf{A}}$.

If φ_i occurs both positively and negatively in φ, then (3.6) reduces to (3.1) and $\mathsf{CNF}_{\mathsf{PG}}$ behaves like $\mathsf{CNF}_{\mathsf{Ts}}$, so that $\mu^{\mathbf{A}}$ does not satisfy $\exists B_i.(\varphi[\varphi_i|B_i] \wedge (B_i \leftrightarrow \varphi_i))$.

If, instead, φ_i occurs only positively [resp. negatively] in φ, then it is possible to extend $\mu^{\mathbf{A}}$ by assigning B_i to $\bot$ [resp. $\top$] to satisfy $(B_i \rightarrow \varphi_i)$ [resp. $(B_i \leftarrow \varphi_i)$] without assigning any atom in φ_i. Thus $\mu^{\mathbf{A}}$ satisfies $\exists B_i.(\varphi[\varphi_i|B_i] \wedge (B_i \rightarrow \varphi_i))$ [resp. $\exists B_i.(\varphi[\varphi_i|B_i] \wedge (B_i \leftarrow \varphi_i))$].

As with $\mathsf{CNF}_{\mathsf{Ts}}$, given $\mu^{\mathbf{A}}$ satisfying φ, in order to produce an assignment $\mu^{\mathbf{A}'}$ satisfying $\exists \mathbf{B}.\mathsf{CNF}_{\mathsf{PG}}(\varphi)$ the enumerator is most often forced to assign other atoms in $\mathbf{A}$, so that $\mu^{\mathbf{A}'} \supset \mu^{\mathbf{A}}$. We notice, however, that the effect of this problem is mitigated by the presence of single-polarity subformulas among those left unassigned by $\mu^{\mathbf{A}}$. As an extreme case, if all subformulas in φ occur with single polarity, then no further assignment to atoms in $\mathbf{A}$ is needed.

Example 3.1.2. Consider φ (3.2) as in Example 3.1.1. Suppose that φ is converted into CNF using $\mathsf{CNF}_{\mathsf{PG}}$. Then, we have:

$$\begin{array}{l} \mathsf{CNF}_{\mathsf{PG}}(\varphi) \stackrel{\text{def}}{=} \\ (\neg B_1 \vee \ A_1) \wedge (\neg B_1 \vee \ A_2) \qquad\qquad\qquad\qquad \wedge \ //(B_1 \rightarrow (A_1 \wedge A_2)) \quad (3.7a) \\ (\ B_2 \vee \neg A_3) \wedge (\ B_2 \vee \neg A_4) \wedge (\neg B_2 \vee \ A_3 \vee \ A_4) \wedge \ //(B_2 \leftrightarrow (A_3 \vee A_4)) \quad (3.7b) \\ (\ B_3 \vee \neg A_5) \wedge (\ B_3 \vee \neg A_6) \wedge (\neg B_3 \vee \ A_5 \vee \ A_6) \wedge \ //(B_3 \leftrightarrow (A_5 \vee A_6)) \quad (3.7c) \\ (\neg B_4 \vee \ B_2) \wedge (\neg B_4 \vee \ B_3) \wedge (\ B_4 \vee \neg B_2 \vee \neg B_3) \wedge \ //(B_4 \leftrightarrow (B_2 \wedge B_3)) \quad (3.7d) \\ (\neg B_5 \vee \ B_4 \vee \neg A_7) \wedge (\neg B_5 \vee \neg B_4 \vee \ A_7) \qquad \wedge \ //(B_5 \rightarrow (B_4 \leftrightarrow A_7)) \quad (3.7e) \\ (\ B_1 \vee \ B_5). \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (3.7f) \end{array}$$

We remark that (3.7a) and (3.7e) are shorter than (3.3a) and (3.3e) respectively, since the corresponding subformulas $(A_1 \wedge A_2)$ and $((\dots) \leftrightarrow A_7)$ occur only with positive polarity in φ (3.2), so that only the one-way implication $(B_i \rightarrow \varphi_i)$ is needed.

As in Example 3.1.1, consider the partial assignment $\mu^{\mathbf{A}} \stackrel{\text{def}}{=} \{\neg A_3, \neg A_4, \neg A_7\}$ (3.4) which satisfies φ (3.2). As before, $\mu^{\mathbf{A}}$ does not satisfy $\exists \mathbf{B}.\mathsf{CNF}_{\mathsf{PG}}(\varphi)$. In fact, there is no total truth assignment $\eta^{\mathbf{B}}$ on $\mathbf{B}$ s.t. $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_{\mathbb{B}} \mathsf{CNF}_{\mathsf{PG}}(\varphi)$, since (3.7c) cannot be satisfied by assigning only variables in $\mathbf{B}$; rather, at least one atom in $\{A_5, A_6\}$ to

satisfy (3.7c) must be assigned. Notice that, unlike in Example 3.1.1, to satisfy (3.7a) we can set $B_1 = \bot$ with no need to assign any atom in $\{A_1, A_2\}$.

Suppose an enumerator assigns first the literals in $\mu^{\mathbf{A}}$, which force it to assign also $\mu^{\mathbf{B}} \stackrel{\text{def}}{=} \{\neg B_2, \neg B_4\}$ due to (3.7b) and (3.7d), respectively.

Since $\mu^{\mathbf{A}} \cup \mu^{\mathbf{B}}$ satisfies all clauses except those in (3.7a), (3.7c), and (3.7f), the enumerator needs to extend $\mu^{\mathbf{A}} \cup \mu^{\mathbf{B}}$ by enumerating partial assignments on the unassigned atoms $\{A_1, A_2, A_5, A_6, B_1, B_3, B_5\}$ to satisfy them. Regardless of the search strategy adopted, to satisfy (3.7c) it is necessary to generate at least 3 partial assignments on $\{A_5, A_6\}$; to satisfy (3.7a), and (3.7f), instead, the enumerator only needs to assign $B_1 = \bot$, which forces it to assign $B_5 = \top$ due to (3.7f).

For instance, in the case of disjoint AllSAT, instead of the single partial assignment $\mu^{\mathbf{A}}$, the solver may return the following list of 3 partial assignments satisfying $\exists\mathbf{B}.\mathsf{CNF}_{\mathsf{PG}}(\varphi)$ which extend $\mu^{\mathbf{A}}$:

$$\begin{array}{llll}
\{\neg A_3, \neg A_4, & \overbrace{\neg A_5, \neg A_6,}^{B_3} & \neg A_7\} & //\{\neg B_1, \neg B_2, \neg B_3, \neg B_4, B_5\} \\
\{\neg A_3, \neg A_4, & A_5, & \neg A_7\} & //\{\neg B_1, \neg B_2, \quad B_3, \neg B_4, B_5\} \\
\{\neg A_3, \neg A_4, & \neg A_5, \quad A_6, & \neg A_7\} & //\{\neg B_1, \neg B_2, \quad B_3, \neg B_4, B_5\}
\end{array} \tag{3.8}$$

In the case of non-disjoint AllSAT, the solver may enumerate a similar set of partial assignments, with $\{A_6\}$ instead of $\{\neg A_5, A_6\}$ in the third assignment.

Notice that, to maximize the benefits of $\mathsf{CNF}_{\mathsf{PG}}$, the subformulas occurring only positively [resp. negatively] must have their label assigned to $\bot$ [resp. $\top$]. In practice, this can be partly achieved by instructing the solver to split on negative values in decision branches.[1] While this does not guarantee that the solver always assigns these labels to $\bot$, we observed empirically that it provides a good approximation.

3.2 Enhancing Enumeration via NNF Preprocessing

In this section, we propose a solution to address the shortcomings of $\mathsf{CNF}_{\mathsf{Ts}}$ and $\mathsf{CNF}_{\mathsf{PG}}$ for AllSAT and $\mathcal{T}$-AllSAT described in Section 3.1. The idea is simple: *we first transform the formula φ into NNF, and then apply $\mathsf{CNF}_{\mathsf{PG}}$ to $\mathsf{NNF}(\varphi)$.* NNF guarantees that every non-atomic subformula of $\mathsf{NNF}(\varphi)$ occurs only positively, since every subformula φ_i occurring with double polarity in φ is converted into two syntactically different subformulas $\varphi_i^+ \stackrel{\text{def}}{=} \mathsf{NNF}(\varphi_i)$ and $\varphi_i^- \stackrel{\text{def}}{=} \mathsf{NNF}(\neg\varphi_i)$, each occurring only positively. Thus, when computing $\mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$, φ_i^+ and φ_i^- are labeled with two distinct atoms B_i^+ and B_i^-, respectively, adding the one-way implications $(B_i^+ \rightarrow \varphi_i^+)$ and $(B_i^- \rightarrow \varphi_i^-)$.

We remark that with this preprocessing we maintain the correctness and completeness of the enumeration process, because φ is equivalent to $\mathsf{NNF}(\varphi)$, which is equivalent to $\exists\mathbf{B}.\mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$. Also, this encoding has linear size w.r.t. φ, since $\mathsf{NNF}(\varphi)$ has linear size w.r.t. φ (by Lemma 1.2.1) and $\mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$ has linear size w.r.t. $\mathsf{NNF}(\varphi)$.

[1] To exploit this heuristic also for subformulas occurring only negatively, the latter can be labeled with a negative label $\neg B_i$ as $(\neg B_i \leftarrow \varphi_i)$.

We have the following fact, for which we provide a complete proof in Appendix A.1.2.

Lemma 3.2.1. *Consider a formula φ, and a partial truth assignment μ. Then $\varphi|_\mu = v$ iff $\mathsf{NNF}(\varphi)|_\mu = v$, for $v \in \{\top, \bot\}$.*

Notice that Lemma 3.2.1 *is not* a direct consequence of the fact that $\mathsf{NNF}(\varphi)$ is equivalence-preserving; see Remark 1.2.2.

We prove the following fact: *if a partial truth assignment $\mu^{\mathbf{A}}$ satisfies φ, then it also satisfies $\exists\mathbf{B}.\mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$.* (The vice versa holds trivially.) We remark that, as shown in Section 3.1, this fact does not hold for $\mathsf{CNF}_{\mathsf{Ts}}$ and $\mathsf{CNF}_{\mathsf{PG}}$.

Theorem 3.2.1. *Consider a formula φ, and a* partial *truth assignment $\mu^{\mathbf{A}}$ such that $\mu^{\mathbf{A}} \models_{\mathbb{B}} \varphi$. Then $\mu^{\mathbf{A}} \models_{\mathbb{B}} \exists\mathbf{B}.\mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$.*

Proof. We show that, for every $\mu^{\mathbf{A}}$ such that $\mu^{\mathbf{A}} \models_{\mathbb{B}} \varphi$, there exists a total truth assignment $\eta^{\mathbf{B}}$ such that $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_{\mathbb{B}} \mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$. We first show how such an $\eta^{\mathbf{B}}$ can be built; then we prove that it satisfies $\mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$ if conjoined with $\mu^{\mathbf{A}}$. In the following, the symbol "$*$" denotes any formula that is not in $\{\top, \bot\}$, so that "$\varphi_i|_{\mu^{\mathbf{A}}} = *$" means that the residual of φ_i under $\mu^{\mathbf{A}}$ is not a constant.

For each subformula φ_i of φ, let variables B_i^+ and B_i^- denote its positive and negative occurrences in $\mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$, respectively. We then proceed as follows:

(a) if $\varphi_i|_{\mu^{\mathbf{A}}} = \top$, and hence $\mathsf{NNF}(\varphi_i)|_{\mu^{\mathbf{A}}} = \top$ and $\mathsf{NNF}(\neg\varphi_i)|_{\mu^{\mathbf{A}}} = \bot$ by Lemma 3.2.1, then add $\{B_i^+, \neg B_i^-\}$ to $\eta^{\mathbf{B}}$;

(b) if $\varphi_i|_{\mu^{\mathbf{A}}} = \bot$, and hence $\mathsf{NNF}(\varphi_i)|_{\mu^{\mathbf{A}}} = \bot$ and $\mathsf{NNF}(\neg\varphi_i)|_{\mu^{\mathbf{A}}} = \top$ by Lemma 3.2.1, then add $\{\neg B_i^+, B_i^-\}$ to $\eta^{\mathbf{B}}$;

(c) otherwise if $\varphi_i|_{\mu^{\mathbf{A}}} = *$, then add $\{\neg B_i^+, \neg B_i^-\}$ to $\eta^{\mathbf{B}}$;

(If φ_i occurs only positively or negatively, then assign only B_i^+ or B_i^-, respectively.)

We prove that $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_{\mathbb{B}} \mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$ by induction on the structure of the formula $\mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$. Consider a subformula φ_i of φ:

(i) if φ_i occurs positively in φ, then

$$\mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi)) \stackrel{\text{def}}{=} \mathsf{CNF}_{\mathsf{PG}}(\overbrace{\mathsf{NNF}(\varphi)[\mathsf{NNF}(\varphi_i)|B_i^+]}^{(3.9a)} \wedge \overbrace{(B_i^+ \rightarrow \mathsf{NNF}(\varphi_i))}^{(3.9b)}) \tag{3.9}$$

(ii) if φ_i occurs negatively in φ, then

$$\mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi)) \stackrel{\text{def}}{=} \mathsf{CNF}_{\mathsf{PG}}(\overbrace{\mathsf{NNF}(\varphi)[\mathsf{NNF}(\neg\varphi_i)|B_i^-]}^{(3.10a)} \wedge \overbrace{(B_i^- \rightarrow \mathsf{NNF}(\neg\varphi_i))}^{(3.10b)}) \tag{3.10}$$

For each pair of cases we have:

(a) (i) $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_{\mathbb{B}}$ (3.9a) since we substitute $\mathsf{NNF}(\varphi_i)$ with B_i^+ and $\mathsf{NNF}(\varphi_i)|_{\mu^{\mathbf{A}}} = B_i^+|_{\eta^{\mathbf{B}}} = \top$; $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_{\mathbb{B}}$ (3.9b) since $\mu^{\mathbf{A}} \models_{\mathbb{B}} \mathsf{NNF}(\varphi_i)$;

(ii) $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_{\mathbb{B}}$ (3.10a) since we substitute $\mathsf{NNF}(\neg\varphi_i)$ with B_i^- and $\mathsf{NNF}(\neg\varphi_i)|_{\mu^{\mathbf{A}}} = B_i^-|_{\eta^{\mathbf{B}}} = \bot$; $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_{\mathbb{B}}$ (3.10b) since $\eta^{\mathbf{B}} \models_{\mathbb{B}} \neg B_i^-$;

(b) (i) $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_{\mathbb{B}}$ (3.9a) since we substitute $\mathsf{NNF}(\varphi_i)$ with B_i^+ and $\mathsf{NNF}(\varphi_i)|_{\mu^{\mathbf{A}}} = B_i^+|_{\eta^{\mathbf{B}}} = \bot$; $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_{\mathbb{B}}$ (3.9b) since $\eta^{\mathbf{B}} \models_{\mathbb{B}} \neg B_i^+$;

(ii) $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_{\mathbb{B}}$ (3.10a) since we substitute $\mathsf{NNF}(\neg\varphi_i)$ with B_i^- and $\mathsf{NNF}(\neg\varphi_i)|_{\mu^{\mathbf{A}}} = B_i^-|_{\eta^{\mathbf{B}}} = \top$; $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_{\mathbb{B}}$ (3.10b) since $\mu^{\mathbf{A}} \models_{\mathbb{B}} \mathsf{NNF}(\neg\varphi_i)$;

(c) (i) $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_{\mathbb{B}}$ (3.9a) since we substitute $\mathsf{NNF}(\varphi_i)$ with B_i^+, and substituting $\mathsf{NNF}(\varphi_i)|_{\mu^{\mathbf{A}}} = *$ with $B_i^+|_{\eta^{\mathbf{B}}} = \bot$ preserves satisfiability; $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_{\mathbb{B}}$ (3.9b) since $\eta^{\mathbf{B}} \models_{\mathbb{B}} \neg B_i^+$;

(ii) $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_{\mathbb{B}}$ (3.10a) since we substitute $\mathsf{NNF}(\neg\varphi_i)$ with B_i^-, and substituting $\mathsf{NNF}(\neg\varphi_i)|_{\mu^{\mathbf{A}}} = *$ with $B_i^-|_{\eta^{\mathbf{B}}} = \bot$ preserves satisfiability; $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_{\mathbb{B}}$ (3.10b) since $\eta^{\mathbf{B}} \models_{\mathbb{B}} \neg B_i^-$. □

By Theorem 3.2.1, given $\mu^{\mathbf{A}}$ satisfying φ, the enumerator need not assign any further atom in $\mathbf{A}$ to satisfy $\exists\mathbf{B}.\mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$. This prevents it from producing multiple extensions of $\mu^{\mathbf{A}}$, thus avoiding the assignment blow-up for $\mathsf{CNF}_{\mathsf{Ts}}$ and $\mathsf{CNF}_{\mathsf{PG}}$ described in Section 3.1. We stress that Theorem 3.2.1 is agnostic of the AllSAT (or $\mathcal{T}$-AllSAT) algorithm adopted, and that it holds for both disjoint and non-disjoint enumeration.

Remark 3.2.1. Unlike with AllSAT or $\mathcal{T}$-AllSAT, the pre-conversion into NNF is typically not used in plain SAT or SMT *solving*, because it causes the unnecessary duplication of labels B_i^+ and B_i^-, with extra overhead and no benefit for the solver.

We notice that the proof of Theorem 3.2.1 is *constructive*, as it not only shows that an assignment $\eta^{\mathbf{B}}$ s.t. $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_{\mathbb{B}} \mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$ exists, but also how to construct it. In particular, items (a), (b), and (c) implicitly suggest a strategy for assigning the values to the $\mathbf{B}$ atoms given $\mu^{\mathbf{A}}$, by iteratively applying the following steps, interleaved with assignments forced by residual constraints:

(a) if B_i^+ occurs negatively in already-satisfied clauses only, add $\{B_i^+, \neg B_i^-\}$ to $\eta^{\mathbf{B}}$;

(b) if B_i^- occurs negatively in already-satisfied clauses only, add $\{B_i^-, \neg B_i^+\}$ to $\eta^{\mathbf{B}}$;

(c) if B_i^+ [resp. B_i^-] occurs negatively in a not-yet-satisfied clause whose other unassigned literals are all $\mathbf{A}$-literals, then add $\{\neg B_i^+\}$ [resp. $\{\neg B_i^-\}$] to $\eta^{\mathbf{B}}$.

This strategy mimics the application of items (a), (b), and (c) in the proof to the subformulas φ_i^+ and φ_i^- of $\mathsf{NNF}(\varphi)$ in a bottom-up fashion, from the leaves to the root.[2]

[2] B_i^+ [resp. B_i^-] occurs negatively only in the clauses encoding $(B_i^+ \rightarrow \varphi_i^+)$ [resp. $(B_i^- \rightarrow \varphi_i^-)$], since by construction it always occurs positively elsewhere, as it substitutes some φ_i^+ [resp. φ_i^-] occurring only positively in $\mathsf{NNF}(\varphi)$. Also, by construction, we can have at most one negative $\mathbf{B}$-literal per clause.

We also notice that, due to the constraints $(B_i^+ \rightarrow \varphi_i^+)$ and $(B_i^- \rightarrow \varphi_i^-)$ and to the fact that φ_i^+ and φ_i^- are mutually inconsistent by construction, no assignment satisfying $\mathsf{CNF_{PG}}(\mathsf{NNF}(\varphi))$ can assign both B_i^+ and B_i^- to $\top$. Thus, to further improve the efficiency of enumeration without affecting its outcome, we also add to $\mathsf{CNF_{PG}}(\mathsf{NNF}(\varphi))$ the mutual-exclusion clauses $(\neg B_i^+ \vee \neg B_i^-)$ when both B_i^+ and B_i^- are introduced, which prevent the solver from assigning both B_i^+ and B_i^- to $\top$, and thus from wasting time in exploring inconsistent search subtrees for residual formulas like $(\ldots \wedge \varphi_i^+ \wedge \varphi_i^- \wedge \ldots)$.

We illustrate the benefit of our proposed technique with the following example.

Example 3.2.1. Consider φ (3.2) as in Example 3.1.1, and its NNF conversion:

$$\mathsf{NNF}(\varphi) \stackrel{\text{def}}{=} \overbrace{\overbrace{\overbrace{(A_1 \wedge A_2)}^{B_1^+} \vee (((\overbrace{(\neg A_3 \wedge \neg A_4)}^{B_2^-} \vee \overbrace{(\neg A_5 \wedge \neg A_6)}^{B_3^-}) \vee A_7)}^{B_4^-} \wedge \overbrace{(((A_3 \vee A_4) \wedge (A_5 \vee A_6)) \vee \neg A_7)}^{B_4^+}}^{B_5^+}}^{B_7^+}$$

Applying $\mathsf{CNF_{PG}}$ and adding the mutual-exclusion clauses, we obtain:

$$\begin{aligned}
\mathsf{CNF_{PG}}(\mathsf{NNF}(\varphi)) \stackrel{\text{def}}{=} & (\neg B_1^+ \vee A_1) \wedge (\neg B_1^+ \vee A_2) \wedge \mathit{//}(B_1^+ \rightarrow (A_1 \wedge A_2)) && \text{(3.11a)}\\
& (\neg B_2^- \vee \neg A_3) \wedge (\neg B_2^- \vee \neg A_4) \wedge \mathit{//}(B_2^- \rightarrow (\neg A_3 \wedge \neg A_4)) && \text{(3.11b)}\\
& (\neg B_3^- \vee \neg A_5) \wedge (\neg B_3^- \vee \neg A_6) \wedge \mathit{//}(B_3^- \rightarrow (\neg A_5 \wedge \neg A_6)) && \text{(3.11c)}\\
& (\neg B_4^- \vee B_2^- \vee B_3^-) \wedge \mathit{//}(B_4^- \rightarrow (B_2^- \vee B_3^-)) && \text{(3.11d)}\\
& (\neg B_5^+ \vee B_4^- \vee A_7) \wedge \mathit{//}(B_5^+ \rightarrow (B_4^- \vee A_7)) && \text{(3.11e)}\\
& (\neg B_2^+ \vee A_3 \vee A_4) \wedge \mathit{//}(B_2^+ \rightarrow (A_3 \vee A_4)) && \text{(3.11f)}\\
& (\neg B_3^+ \vee A_5 \vee A_6) \wedge \mathit{//}(B_3^+ \rightarrow (A_5 \vee A_6)) && \text{(3.11g)}\\
& (\neg B_4^+ \vee B_2^+) \wedge (\neg B_4^+ \vee B_3^+) \wedge \mathit{//}(B_4^+ \rightarrow (B_2^+ \wedge B_3^+)) && \text{(3.11h)}\\
& (\neg B_6^+ \vee B_4^+ \vee \neg A_7) \wedge \mathit{//}(B_6^+ \rightarrow (B_4^+ \vee \neg A_7)) && \text{(3.11i)}\\
& (\neg B_7^+ \vee B_5^+) \wedge (\neg B_7^+ \vee B_6^+) \wedge \mathit{//}(B_7^+ \rightarrow (B_5^+ \wedge B_6^+)) && \text{(3.11j)}\\
& (B_1^+ \vee B_7^+) \wedge && \text{(3.11k)}\\
& (\neg B_2^+ \vee \neg B_2^-) \wedge \mathit{//mutex\ for\ } B_2^+, B_2^- && \text{(3.11l)}\\
& (\neg B_3^+ \vee \neg B_3^-) \wedge \mathit{//mutex\ for\ } B_3^+, B_3^- && \text{(3.11m)}\\
& (\neg B_4^+ \vee \neg B_4^-). \quad \mathit{//mutex\ for\ } B_4^+, B_4^- && \text{(3.11n)}
\end{aligned}$$

Notice that the labels B_1^-, B_5^-, B_6^-, B_7^-, and their one-way constraints are not introduced, because there is no negative occurrence of their subformulas in φ.

Consider the partial assignment $\mu^{\mathbf{A}} \stackrel{\text{def}}{=} \{\neg A_3, \neg A_4, \neg A_7\}$ (3.4) which satisfies φ, as in Examples 3.1.1 and 3.1.2. First, it is easy to see that $\mu^{\mathbf{A}}$ also satisfies $\mathsf{NNF}(\varphi)$, in compliance with Lemma 3.2.1. Also, $\mu^{\mathbf{A}}$ satisfies $\exists \mathbf{B}.\mathsf{CNF_{PG}}(\mathsf{NNF}(\varphi))$, because $\mu^{\mathbf{A}} \cup$

$\eta^{\mathbf{B}} \models_{\mathbb{B}} \mathsf{CNF_{PG}}(\mathsf{NNF}(\varphi))$ with $\eta^{\mathbf{B}} \stackrel{\text{def}}{=} \{\neg B_1^+, \neg B_2^+, B_2^-, \neg B_3^+, \neg B_3^-, \neg B_4^+, B_4^-, B_5^+, B_6^+, B_7^+\}$.

Such $\eta^{\mathbf{B}}$ can be produced by adopting the above strategy. First, we assign the literals in $\mu^{\mathbf{A}}$, which also force us to add $\{\neg B_2^+, \neg B_4^+\}$ due to (3.11f) and (3.11h), respectively. Then we add $\{B_6^+\}$ by step (a) on (3.11i), $\{B_2^-\}$ by step (b) on (3.11b), and $\{\neg B_1^+, \neg B_3^-, \neg B_3^+\}$ by step (c) on (3.11a), (3.11c), and (3.11g), respectively. These force us to add $\{B_7^+\}$ and $\{B_5^+\}$ due to (3.11k) and (3.11j), respectively. Finally, we add $\{B_4^-\}$ by step (b) on (3.11d). Overall, this allows the enumeration of only the partial assignment $\mu^{\mathbf{A}}$, with both disjoint and non-disjoint enumeration:

$$\{\neg A_3, \neg A_4, \neg A_7\} \quad //\{\neg B_1^+, \neg B_2^+, B_2^-, \neg B_3^+, \neg B_3^-, \neg B_4^+, B_4^-, B_5^+, B_6^+, B_7^+\}.$$

The shorter $\mu^{\mathbf{A}}$ is w.r.t. a total assignment, the greater the chance that $\mathsf{CNF_{Ts}}$ and $\mathsf{CNF_{PG}}$ will force the production of many extra assignments, and hence the greater the benefit of using NNF+$\mathsf{CNF_{PG}}$ to avoid this. That said, if the enumerator is able to produce short partial assignments $\mu^{\mathbf{A}}$ satisfying the formula, we expect a high benefit from using NNF+$\mathsf{CNF_{PG}}$ instead of $\mathsf{CNF_{Ts}}$ and $\mathsf{CNF_{PG}}$; conversely, if the enumerator produces only total or nearly total assignments, we expect no or very limited benefit, respectively.

Implementation-wise, the strategy to assign the values of $\eta^{\mathbf{B}}$ described above is difficult to implement inside the current enumerators. Therefore, there is no formal guarantee that a generic enumeration procedure always finds the exact $\eta^{\mathbf{B}}$ that prevents the generation of longer assignments. For example, the enumeration procedure of Section 1.3.2 finds a total truth assignment $\eta^{\mathbf{A}} \cup \eta^{\mathbf{B}}$ that satisfies the formula, and then finds $\mu^{\mathbf{A}} \subseteq \eta^{\mathbf{A}}$ that is minimal w.r.t. that specific $\eta^{\mathbf{B}}$ such that $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_{\mathbb{B}} \mathsf{CNF_{PG}}(\mathsf{NNF}(\varphi))$, so that the $\eta^{\mathbf{B}}$ found is not guaranteed to be the one that allows for the most effective minimization of $\mu^{\mathbf{A}}$. Ad hoc enumeration heuristics could be adopted. Nevertheless, in Section 3.3 we show that a very simple heuristic —i.e., assign false values to decision atoms first— guarantees dramatic improvements w.r.t. previous approaches, using two state-of-the-art AllSAT/$\mathcal{T}$-AllSAT enumerators.

Relationship with Dual-Rail Encoding. The idea of using two different variables to label the positive and negative occurrences of a subformula has similarities with the so-called *dual-rail* encoding [33, 172] (see also Section 2.2). Using the dual-rail encoding for enumeration, however, requires ad hoc enumeration algorithms, since the solver must take into account the three-valued semantics of truth assignments over dual-rail atoms, e.g., when adding blocking clauses [83].

Our CNF encoding, instead, can be used in combination with any projected enumeration algorithm that satisfies the properties described in Section 1.3.1. In fact, $\mathsf{NNF} + \mathsf{CNF_{PG}}$ duplicates only the label atoms in $\mathbf{B}$, which are projected away, and thus does not require any modification of the enumeration algorithm. Moreover, it introduces fewer atoms than the dual-rail encoding. Finally, we note that the clauses in the form $(\neg B_i^+ \vee \neg B_i^-)$ are not needed for correctness but only for efficiency, and can in principle be omitted. In the dual-rail encoding, instead, their presence is essential to prevent the illegal assignment $\eta(B_i^+) = \eta(B_i^-) = \top$.

3.3 Experimental Evaluation

In this section, we experimentally evaluate the impact of different CNF-izations on the disjoint and non-disjoint AllSAT and $\mathcal{T}$-AllSAT tasks. In order to compare them on fair ground, we implemented a basic version of each from scratch in PySMT [85], avoiding specific optimizations done by the solvers.

We conducted an extensive collection of experiments on AllSAT and $\mathcal{T}$-AllSAT. We tested the encodings on MathSAT, which supports disjoint and non-disjoint AllSAT and $\mathcal{T}$-AllSAT, and on TabularAllSAT and TabularAllSMT, which support disjoint AllSAT and $\mathcal{T}$-AllSAT, respectively. The choice of the solvers is motivated in Section 3.3.1; the details needed to reproduce the experiments are given in Section 3.3.2, the benchmarks are described in Section 3.3.3, and the results are presented in Section 3.3.4; finally, in Section 3.3.5 we analyze and discuss the application of the encodings to related fields.

3.3.1 An Analysis of Available Solvers

To evaluate the CNF encodings, we need a SAT/SMT enumerator that:

(i) is publicly available;

(ii) takes as input a CNF formula;

(iii) allows *projected* enumeration;

(iv) allows enumerating (disjoint or non-disjoint) *partial* truth assignments;

(v) also, the ability of producing *minimal* assignments is valuable although not necessary.

In the literature, we found the following candidate solvers: for AllSAT, we found RELSAT [10], Grumberg [99], SOLALL [133], Jin [114, 115], clasp [88], PicoSAT [16], Yu [235], BC, NBC, and BDD [219], depbdd [218], Dualiza [155], BASolver [240], AllSATCC [135], HALL [83, 82], TabularAllSAT [206, 207], and decdnnf [125]. For $\mathcal{T}$-AllSAT, the only candidates are MathSAT [48], aZ3 [176], and TabularAllSMT [207].

In Table A.1 of Appendix A.2.1, we report an analysis of features (i)–(v) for each of the above-mentioned solvers. Overall, we observe that only four solvers match all our needs: MathSAT, TabularAllSAT, TabularAllSMT, and Dualiza. MathSAT supports disjoint and non-disjoint AllSAT and $\mathcal{T}$-AllSAT enumeration of minimal partial assignments, using the blocking-clause-based enumeration strategy of [129] described in Section 1.3.2. TabularAllSAT supports disjoint AllSAT enumeration of non-minimal partial assignments, based on chronological backtracking without blocking clauses [206, 207]. TabularAllSMT, its counterpart for $\mathcal{T}$-AllSAT, supports disjoint $\mathcal{T}$-AllSAT. Dualiza supports disjoint and non-disjoint AllSAT enumeration of non-minimal partial assignments, based on dual reasoning. Since Dualiza has been shown empirically to perform poorly [82], we decided to leave it out of our experiments and to focus on MathSAT, TabularAllSAT, and TabularAllSMT only.

3.3.2 Information for the Reproducibility of the Experiments

We ran MathSAT with the option -dpll.allsat_minimize_model=true to enumerate minimal partial assignments. For disjoint and non-disjoint enumeration, we set the options -dpll.allsat_allow_duplicates=false and -dpll.allsat_allow_duplicates=true, respectively. We set the options -preprocessor.simplification=0 and -preprocessor.toplevel_propagation=false to disable several non-validity-preserving preprocessing steps. As discussed in Section 3.1.2 and Section 3.2, we also set the options -dpll.branching_initial_phase=0 and -dpll.branching_cache_phase=2 to split on the false branch first while enabling phase caching. TabularAllSAT was run with default options, which include branching on the false branch first. TabularAllSMT was run with the same options as MathSAT to disable preprocessing steps, since it uses MathSAT for theory reasoning.

All the experiments were run on an Intel Xeon Gold 6238R @ 2.20GHz 28-core machine with 128 GB of RAM, running Ubuntu Linux 20.04. We set a timeout of 3600 s for each instance. Benchmarks, results, and source code are available on Zenodo [144, 145]. An updated version of the source code is available on GitHub.[3]

3.3.3 Benchmarks

We consider five sets of benchmarks of non-CNF formulas coming from different sources, both synthetic and real-world. For AllSAT, we consider three sets of benchmarks, namely synthetic Boolean formulas, formulas encoding properties of ISCAS'85 circuits [30, 103], and formulas encoding combinatorial circuits as And-Inverter Graphs (AIGs) [83].

For $\mathcal{T}$-AllSAT, we consider two sets of benchmarks, namely synthetic SMT($\mathcal{LRA}$) formulas and formulas encoding WMI problems [204].

The Boolean synthetic benchmarks. We generated the Boolean synthetic benchmarks by nesting Boolean operators $\wedge, \vee, \leftrightarrow$ to a depth d. For internal nodes, we chose $\leftrightarrow$ with 0.1 probability, and $\wedge$ and $\vee$ with an equal probability of 0.45. Internal and leaf nodes were negated with 0.5 probability. We generated 100 instances with 20 Boolean atoms and depth $d = 8$, 100 instances with 25 Boolean atoms and depth $d = 8$, and 100 instances with 30 Boolean atoms and depth $d = 6$, for a total of 300 instances.

The ISCAS'85 benchmarks. The ISCAS'85 benchmarks are a set of 10 combinatorial circuits used in test generation, timing analysis, and technology mapping [30]. They have well-defined, high-level structures and functions based on multiplexers, ALUs, decoders, and other common building blocks [103]. We generated random instances as described in [217]. In particular, for each circuit, we constrained 60%, 70%, 80%, 90%, and 100% of the outputs to either 0 or 1, for a total of 250 instances.

The AIG benchmarks. The AIG benchmarks [83] consist of 89 instances, subdivided into 3 groups: 29 instances encoding industrial static timing analysis problems containing up to 13000 input variables, 40 instances from the "arithmetic" and "random_control"

[3] https://github.com/masinag/allsat-cnf

benchmarks of the EPFL suite [4] —combining the multiple outputs with an OR or an XOR operator— containing up to 1200 input variables, and 20 instances consisting of large randomly-generated AIGs containing up to 2800 input variables.

The CNF-izations discussed above can be applied to AIGs as well, as an AIG can be seen as a non-CNF formula involving only $\wedge$ and $\neg$ operators. Moreover, we store formulas as DAGs, so that the same subformula is shared among all its occurrences, and the CNF-izations use the same atom to label all occurrences of the same subformula. Hence, all the resulting CNF formulas have linear size w.r.t. the size of the AIG.

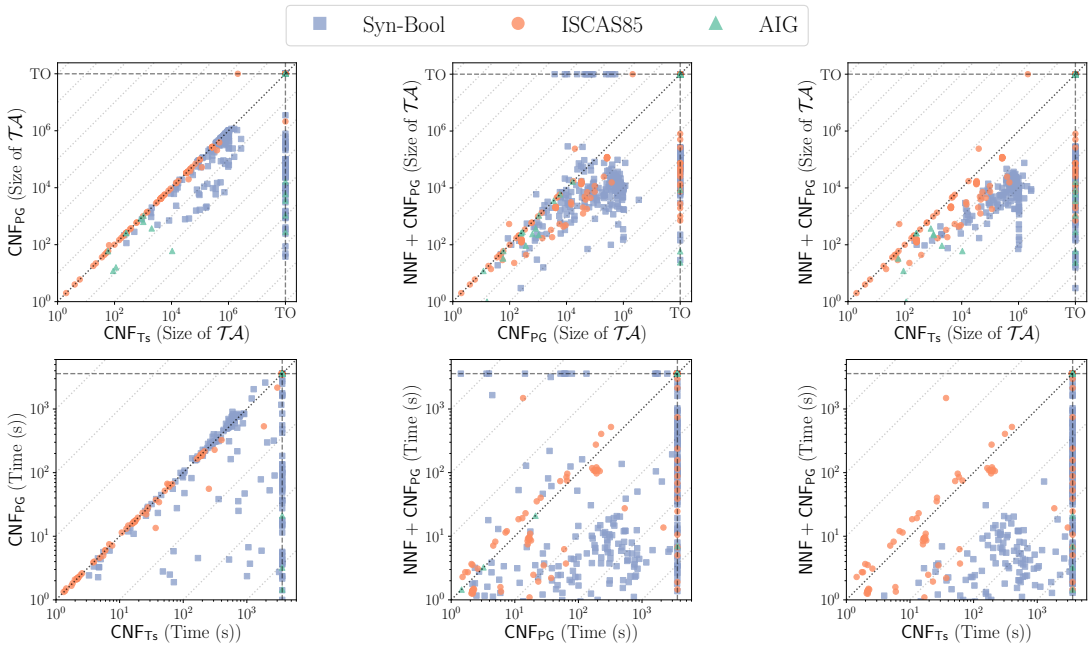
The SMT($\mathcal{LRA}$) synthetic benchmarks. These problems were generated with the same procedure as the Boolean ones, with the difference that atoms are randomly-generated $\mathcal{LRA}$-atoms over a set of R real variables $\{x_1, \ldots, x_R\}$, in the form $(\sum_{i=1}^{R} a_i x_i \leq b)$, where each a_i and b are random real coefficients. We generated 100 instances with depth $d = 5$, 100 instances with depth $d = 6$, and 100 instances with depth $d = 7$, all of them involving $R = 5$ real variables, for a total of 300 instances.

The WMI benchmarks. We generated WMI problems using the procedure in [204]. Specifically, the paper addresses the problem of enumerating all the different paths of the weight function by encoding it into a skeleton formula (see also Chapter 4). Each instance consists of a skeleton formula of a randomly-generated weight function, where the conditions are random formulas over Boolean and $\mathcal{LRA}$-atoms. Since the conditions are typically non-atomic, the resulting formula is not in CNF. Following [204], we fixed the numbers of both Boolean atoms and real variables to 3, and generated 10 instances for each depth value 4, 5, 6, and 7, for a total of 40 instances.

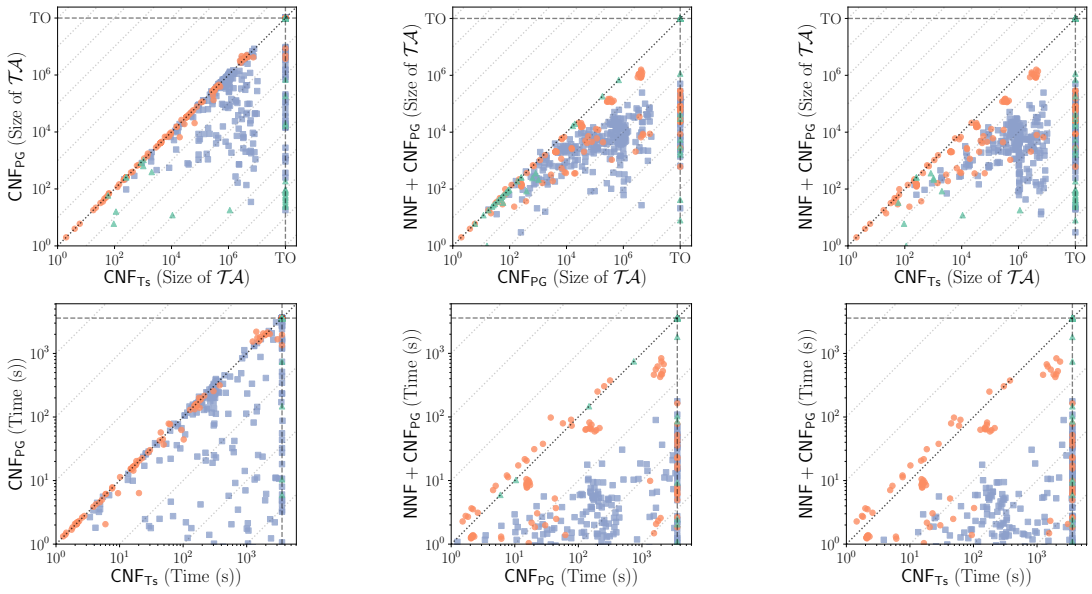
3.3.4 Results

Figures 3.1 and 3.2 show the results for AllSAT with MATHSAT and TABULARALLSAT, respectively. Figures 3.3 and 3.4 show the results for $\mathcal{T}$-AllSAT with MATHSAT and TABULARALLSMT, respectively. For each figure, we report a pair of subfigures comparing the CNF-izations for disjoint and non-disjoint enumeration (when both are supported). Each subfigure reports a set of scatter plots to compare $\mathsf{CNF_{Ts}}$, $\mathsf{CNF_{PG}}$ and $\mathsf{NNF{+}CNF_{PG}}$ in terms of the number of partial truth assignments (size of $\mathcal{TA}$), in the first row, and execution time, in the second row. Each benchmark is represented by a different color and marker. Timeouts are represented by points on the dashed lines and summarized in the tables below the plots. Notice the logarithmic scale of the axes (!).

The Boolean synthetic benchmarks. (Figures 3.1 and 3.2) The plots show that in the disjoint case $\mathsf{CNF_{PG}}$ performs better than $\mathsf{CNF_{Ts}}$, as both MATHSAT (Figure 3.1a) and TABULARALLSAT (Figure 3.2a) enumerate a smaller $\mathcal{TA}$ (first row) in less time (second row) on every instance. $\mathsf{NNF{+}CNF_{PG}}$ yields by far the best results for both solvers, drastically reducing the size of $\mathcal{TA}$ and the execution time by orders of magnitude w.r.t. both $\mathsf{CNF_{Ts}}$ and $\mathsf{CNF_{PG}}$. The advantage of $\mathsf{NNF{+}CNF_{PG}}$ over both $\mathsf{CNF_{Ts}}$ and $\mathsf{CNF_{PG}}$ is even more evident for the non-disjoint case (Figure 3.1b).



(a) Results for disjoint enumeration.

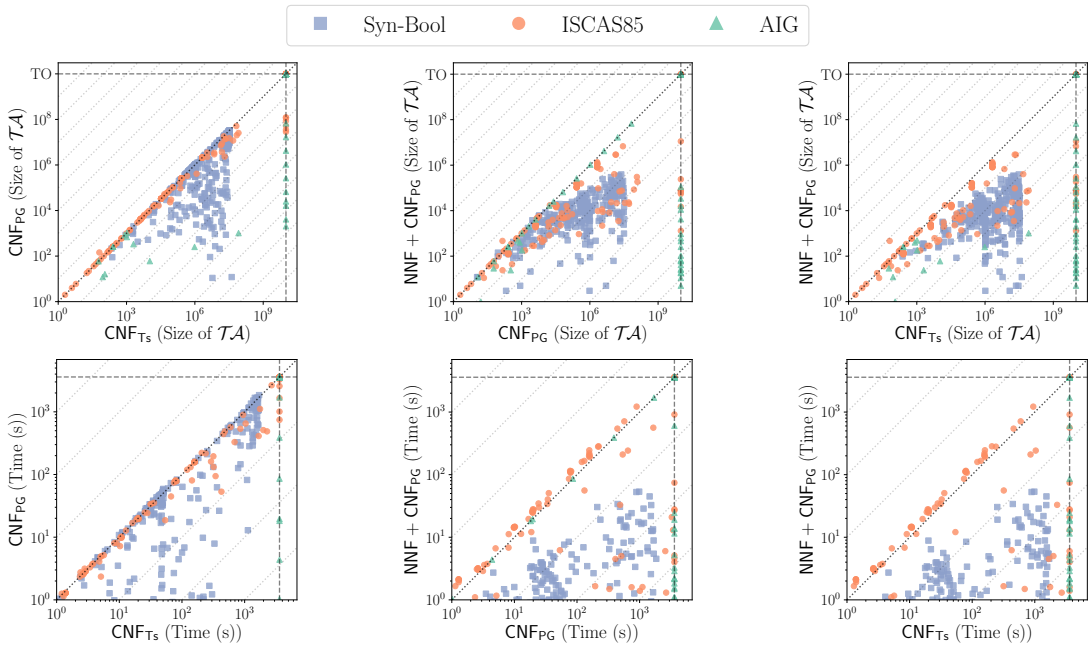


(b) Results for non-disjoint enumeration.

Bench.	Instances	T.O. for disjoint AllSAT			T.O. for non-disjoint AllSAT		
		$\mathsf{CNF_{Ts}}$	$\mathsf{CNF_{PG}}$	$\mathsf{NNF{+}CNF_{PG}}$	$\mathsf{CNF_{Ts}}$	$\mathsf{CNF_{PG}}$	$\mathsf{NNF{+}CNF_{PG}}$
Syn-Bool	300	151	84	**20**	88	48	**0**
ISCAS85	250	47	43	**30**	27	22	**1**
AIG	89	79	73	**71**	78	60	**50**

(c) Number of timeouts.

Figure 3.1: Results on the Boolean benchmarks using MathSAT. Plots in 3.1a and 3.1b compare CNF-izations by $\mathcal{TA}$ size (first row) and execution time (second row). Points on dashed lines represent timeouts, shown in 3.1c. All axes use a logarithmic scale.



(a) Results for disjoint enumeration.

Bench.	Instances	T.O. for disjoint AllSAT		
		$\mathsf{CNF}_{\mathsf{Ts}}$	$\mathsf{CNF}_{\mathsf{PG}}$	NNF+$\mathsf{CNF}_{\mathsf{PG}}$
Syn-Bool	300	**0**	**0**	**0**
ISCAS85	250	17	11	**3**
AIG	89	77	67	**47**

(b) Number of timeouts.

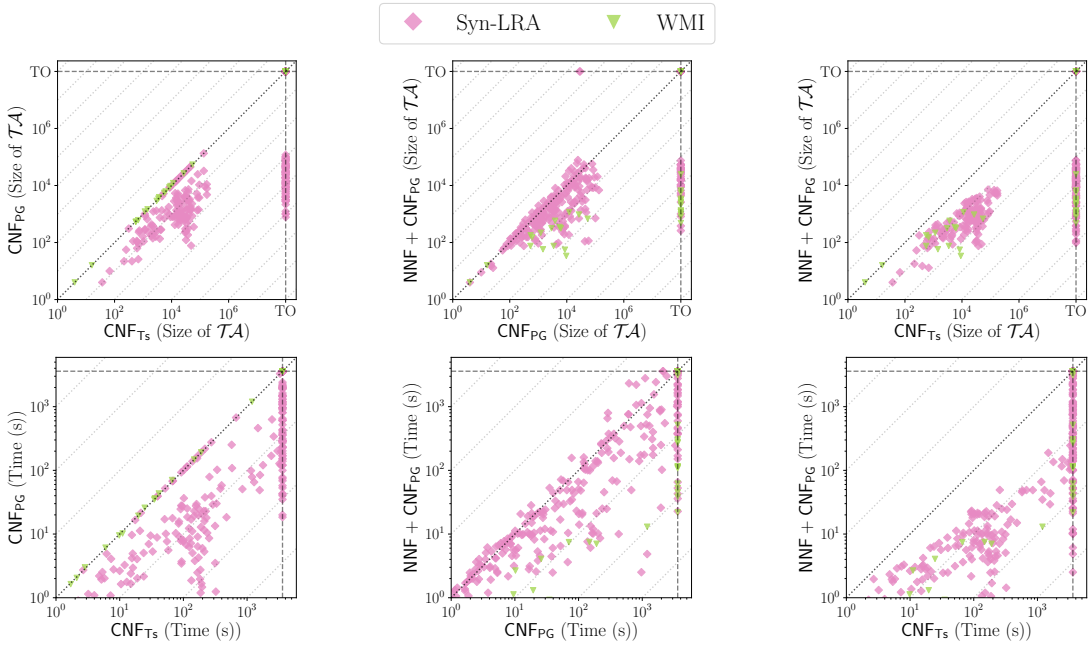
Figure 3.2: Results on the Boolean benchmarks using TabularAllSAT. Plots in 3.2a compare CNF-izations by $\mathcal{TA}$ size (first row) and execution time (second row). Points on dashed lines represent timeouts, shown in 3.2b. All axes use a logarithmic scale.

The ISCAS'85 benchmarks. (Figures 3.1 and 3.2) Here $\mathsf{CNF}_{\mathsf{Ts}}$ and $\mathsf{CNF}_{\mathsf{PG}}$ behave very similarly, both in runtime and in the size of $\mathcal{TA}$. This is because, in circuits, most subformulas occur with double polarity, making the two encodings almost identical.

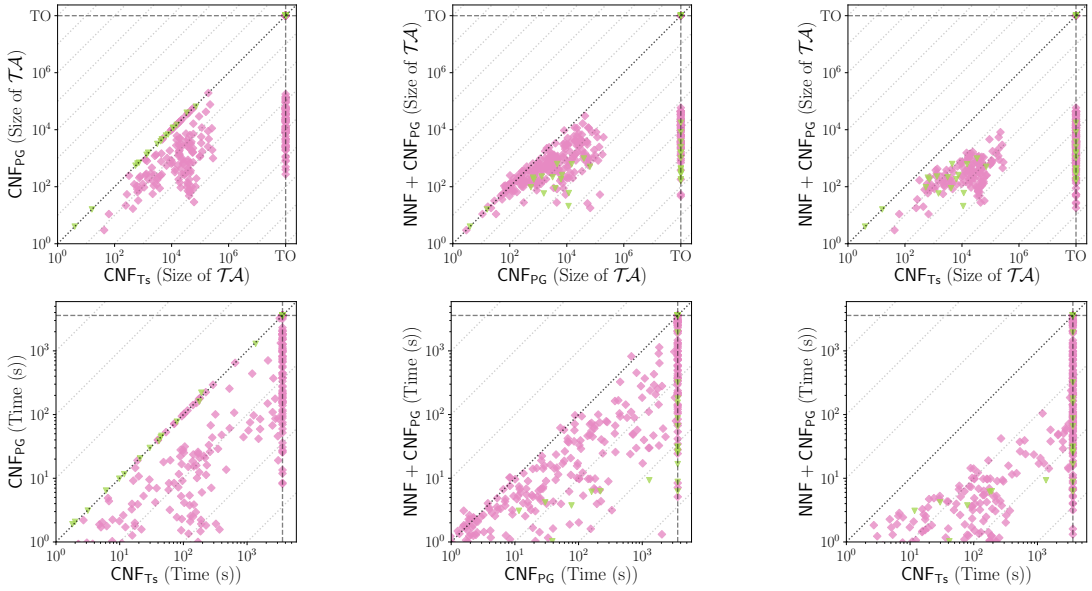
With NNF+$\mathsf{CNF}_{\mathsf{PG}}$, both disjoint and non-disjoint enumeration are much more effective, enumerating a much smaller $\mathcal{TA}$, with only a few outliers. For some instances, NNF+$\mathsf{CNF}_{\mathsf{PG}}$ takes slightly longer; this may be because it can produce a formula up to twice as large, with up to twice as many label atoms as the other encodings, increasing the time needed to find assignments. We remark that enumerating a smaller $\mathcal{TA}$ is still beneficial in many applications, even at the price of a small time overhead.

The AIG benchmarks. (Figures 3.1 and 3.2) This set of benchmarks is by far the most challenging one, with many Boolean atoms connected in a very complex structure. Thus, many instances time out even with the best-performing encoding, NNF+$\mathsf{CNF}_{\mathsf{PG}}$.

Nevertheless, we can still observe that NNF+$\mathsf{CNF}_{\mathsf{PG}}$ is the best-performing encoding for both solvers in terms of both the size of $\mathcal{TA}$ and execution time.



(a) Results for disjoint enumeration.

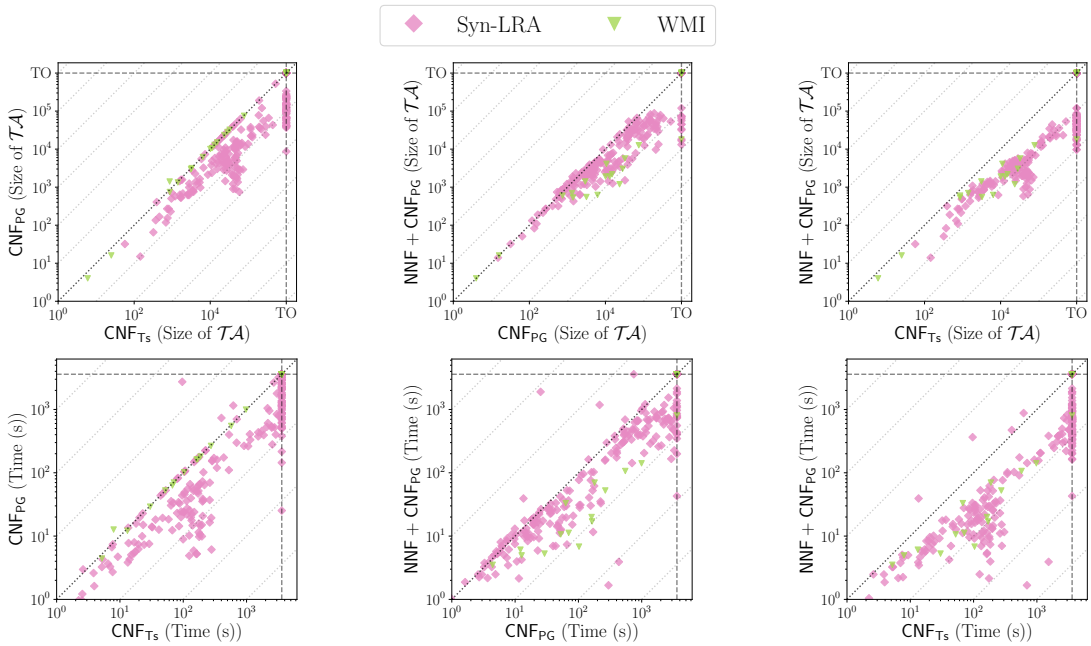


(b) Results for non-disjoint enumeration.

Bench.	Instances	T.O. for disjoint $\mathcal{T}$-AllSAT			T.O. for non-disjoint $\mathcal{T}$-AllSAT		
		$\mathsf{CNF_{Ts}}$	$\mathsf{CNF_{PG}}$	$\mathsf{NNF{+}CNF_{PG}}$	$\mathsf{CNF_{Ts}}$	$\mathsf{CNF_{PG}}$	$\mathsf{NNF{+}CNF_{PG}}$
Syn-LRA	300	155	88	**48**	152	74	**7**
WMI	40	23	23	**9**	23	23	**9**

(c) Number of timeouts.

Figure 3.3: Results on the SMT($\mathcal{LRA}$) benchmarks using MathSAT. Plots in 3.3a, 3.3b compare CNF-izations by $\mathcal{TA}$ size (first row) and execution time (second row). Points on dashed lines represent timeouts, shown in 3.3c. All axes use a logarithmic scale.



(a) Results for disjoint enumeration.

Bench.	Instances	T.O. for disjoint $\mathcal{T}$-AllSAT		
		$\mathsf{CNF_{Ts}}$	$\mathsf{CNF_{PG}}$	$\mathsf{NNF{+}CNF_{PG}}$
Syn-LRA	300	147	92	**73**
WMI	40	24	24	**23**

(b) Number of timeouts.

Figure 3.4: Results on SMT($\mathcal{LRA}$) benchmarks using TabularAllSMT. Plots in 3.4a compare CNF-izations by $\mathcal{TA}$ size (first row) and execution time (second row). Points on dashed lines represent timeouts, shown in 3.4b. All axes use a logarithmic scale.

The SMT($\mathcal{LRA}$) synthetic benchmarks. (Figures 3.3 and 3.4) The plots confirm that the analysis also holds for the $\mathcal{T}$-AllSAT case. The results are in line with those obtained on the Boolean benchmarks, for both the disjoint and non-disjoint cases.

The WMI benchmarks. (Figures 3.3 and 3.4) In these benchmarks, most subformulas occur with double polarity, so $\mathsf{CNF_{Ts}}$ and $\mathsf{CNF_{PG}}$ are almost identical and yield very similar results in both metrics. $\mathsf{NNF{+}CNF_{PG}}$ provides a significant advantage, allowing the enumeration of a much smaller $\mathcal{TA}$. This is crucial in WMI, where each truth assignment requires computing a costly integral [160, 161, 204, 205]. While WMI requires disjoint enumeration, we also report results for the non-disjoint case for completeness.

Overall results. Overall, the plots and tables in Figures 3.1 to 3.4, it is eye-catching that the usage of $\mathsf{NNF{+}CNF_{PG}}$ instead of $\mathsf{CNF_{Ts}}$ or $\mathsf{CNF_{PG}}$ causes a dramatic improvement in the performance of MathSAT, TabularAllSAT, and TabularAllSMT in terms of both CPU time and size of $\mathcal{TA}$, for both disjoint and non-disjoint enumeration, across all AllSAT and $\mathcal{T}$-AllSAT benchmarks.

3.3.5 Discussion: Using Our Encoding with Related Problems

We stress the fact that the proposed $\mathsf{NNF}{+}\mathsf{CNF}_{\mathsf{PG}}$ encoding is conceived and expected to be effective for *enumeration* techniques that generate *partial* assignments (see Section 3.2). Thus, we do not expect benefits in using it for plain SAT or SMT *solving* (see Remark 3.2.1), nor for enumeration techniques that generate *total* or *nearly-total* assignments. To this end, we empirically investigate and discuss related problems where, for this reason, $\mathsf{NNF}{+}\mathsf{CNF}_{\mathsf{PG}}$ is not expected to help (see also Section 2.2).

Plain SAT and SMT solving. To support Remark 3.2.1, in Appendix A.2.2 we compare the different CNF encodings for plain SAT and SMT solving on all the benchmarks in Section 3.3.3. Even though these problems are very small for plain solving, and SAT and SMT solvers deal with them efficiently, we can see that using $\mathsf{NNF}{+}\mathsf{CNF}_{\mathsf{PG}}$ brings no advantage, and solving is uniformly slower than with $\mathsf{CNF}_{\mathsf{PG}}$ or $\mathsf{CNF}_{\mathsf{Ts}}$. This shows that our encoding works specifically for enumeration but not for solving, as expected.

Answer Set Programming. For the reason discussed above, we do not expect our CNF encoding to bring benefits to Answer Set Programming (ASP), because answer sets represent *total* truth assignments: they include atoms assigned to true, while other atoms are implicitly assumed to be assigned to false.[4] Notice that the concept of "minimal" in ASP refers to the minimality of the set of atoms that are true in the answer set, and not to the minimality of the set of atoms that are assigned a truth value, as in SAT and SMT enumeration (see Sections 1.2 and 1.3.2).

(Weighted) #SAT. For (weighted) SAT model counting the $\mathsf{CNF}_{\mathsf{Ts}}$ encoding is currently the most suitable, because it preserves the model count, whereas $\mathsf{CNF}_{\mathsf{PG}}$ —and hence $\mathsf{NNF}{+}\mathsf{CNF}_{\mathsf{PG}}$— does not. To preserve the model count with these encodings and with current model counters, one should perform *projected* #SAT on the original atoms, which can be more expensive than the non-projected case.

Although investigating ad hoc model counting techniques for $\mathsf{NNF}{+}\mathsf{CNF}_{\mathsf{PG}}$-encoded formulas (e.g., by exploiting the polarity of label variables) could be an interesting topic for future research, it would definitely exceed the scope of this discussion.

Notably, the model counter #CLASP [6] is based on the enumeration of minimal partial assignments and would likely benefit from our encoding. Unfortunately, we did not receive an answer from the authors about the availability of the tool.

d-DNNF compilation, enumeration, and counting. We investigated the effect of the different CNF-izations on d-DNNF (projected) compilation, enumeration, and model counting. We tested the tools D4+DECDNNF [125] for projected d-DNNF compilation and enumeration, and D4 [126] for projected d-DNNF-based model counting, on the

[4]We received confirmation from the clasp and clingo developers that these tools do not support partial assignments. See `https://github.com/potassco/clingo/issues/512#issuecomment-2223162682`.

Boolean benchmarks in Section 3.3.3. Each formula was converted into CNF and then compiled into d-DNNF projected on the original atoms **A**.

The results are reported in Appendices A.2.3 and A.2.4 for enumeration and model counting, respectively. We observe that when using use $\mathsf{NNF}{+}\mathsf{CNF}_{\mathsf{PG}}$ instead of $\mathsf{CNF}_{\mathsf{Ts}}$ or $\mathsf{CNF}_{\mathsf{PG}}$, these tools: (a) show no substantial difference in terms of the number and size of the assignments generated; (b) even worsen their runtime performance.

Point (a) is due to the fact that the $\mathrm{D4}_p$ algorithm [127] —implemented by D4 for projected compilation and counting— branches first on important atoms **A**, and only at the end of the branch tries to complete the assignment on **B**. Consequently, the assignments projected on **A** include all or almost all important atoms, thus producing total or nearly total projected assignments on **A**. This is not surprising, since d-DNNF-based tools typically derive their efficiency not from the enumeration of short partial assignments but from the effective decomposition and caching of subproblems.

Point (b) may be due to several reasons: unlike $\mathsf{CNF}_{\mathsf{Ts}}$, $\mathsf{CNF}_{\mathsf{PG}}$ does not allow the assignment of the **A** values to force the deterministic assignment of the **B** values by BCP, causing unnecessary search; also, the NNF transformation up to doubles the number of the non-important atoms **B** to be projected away; furthermore, we conjecture that the duplication of labels B_i^+ and B_i^-, along with distinct encodings φ_i^+ and φ_i^- for $\mathsf{NNF}(\varphi_i)$ and $\mathsf{NNF}(\neg\varphi_i)$, may affect the effectiveness of the caching and partitioning mechanisms inside the D4 d-DNNF compiler.

Although ad hoc d-DNNF compilation strategies for counting and enumeration could in principle be devised to exploit the properties of our $\mathsf{NNF}{+}\mathsf{CNF}_{\mathsf{PG}}$ encoding, investigating such techniques would exceed the scope of this thesis. Finally, we note that d-DNNF compilation techniques for non-CNF formulas have recently been proposed [67].

3.4 Conclusions

We have presented a theoretical and empirical analysis of the impact of different CNF-izations on SAT and SMT enumeration, both disjoint and non-disjoint. We have shown how the most popular transformations conceived for SAT and SMT solving, namely the Tseitin and the Plaisted and Greenbaum CNF-izations, prevent the solver from producing short partial assignments, thus seriously affecting the effectiveness of the enumeration. To overcome this limitation, we have proposed to preprocess the formula by converting it into NNF before applying the Plaisted and Greenbaum transformation. We have shown, both theoretically and empirically, that the latter approach can fully overcome the problem and can drastically reduce both the number of partial assignments and the execution time.

This work opens an interesting research avenue: investigating the role of CNF-ization in neighboring fields such as d-DNNF compilation and model counting, possibly adapting d-DNNF compilers and model counters to exploit different forms of CNF-izations.

Chapter 4

SMT-Enumeration Techniques for Structure-Aware WMI

> **Full Disclosure.** This chapter is based on the paper *"Enhancing SMT-based Weighted Model Integration by Structure Awareness"* [205], co-authored by the Ph.D. candidate with Dr. Giuseppe Spallitta, Dr. Paolo Morettin, Prof. Andrea Passerini, and Prof. Roberto Sebastiani. The main contributor of the paper is Dr. Giuseppe Spallitta. The candidate contributed to the conceptualization of the proposed approach, the implementation, and the experimental evaluation.

We now focus on an application of SMT enumeration, namely probabilistic inference in hybrid domains via Weighted Model Integration (WMI) (Section 1.4). State-of-the-art WMI solvers are based on two main approaches: Knowledge Compilation (KC) and SMT-based Predicate Abstraction (PA) (see Sections 1.4.2 and 2.3). Both approaches have their own strengths and weaknesses.

KC-based solutions [72, 120], although effective in exploiting the structure of the problem, can result in exponentially large circuits when the problem is densely coupled.

SMT-based solutions [160, 161], on the other hand, are more scalable and can handle complex constraints, but they tend to ignore the conditional structure of the weight function, which forces the computation of unnecessarily many integrals.

Contributions. To address these problems, in this chapter we introduce Structure-Aware Enumeration for WMI (SAE4WMI), a novel algorithm for WMI that aims to combine the best of both worlds by introducing weight-structure awareness into SMT-based WMI. The main idea is to build a formula, which we call the *conditional skeleton*, mimicking the conditional structure of the weight function to drive the $\mathcal{T}$-AllSAT enumeration algorithm. An extensive experimental evaluation on synthetic and real-world benchmarks shows substantial advantages over state-of-the-art WMI solvers.

Moreover, we highlight how SMT-based algorithms are *agnostic* to the integration algorithm invoked for each enumerated assignment. To this end, we extend SAE4WMI to

support both *exact numerical integration* (already used in [160, 161]) and *approximate integration*. The advantages of the latter are twofold: (i) it improves the scalability on complex instances when integration is the major bottleneck; (ii) it allows computing WMI for problems with non-polynomial weight functions (e.g., Gaussian densities). We show the relevance of the latter point by presenting a prototypical application to the verification of the fairness of probabilistic programs.

Organization. The rest of the chapter is organized as follows. In Section 4.1, we analyze the WMI approaches based on KC and on PA, identifying some of their weaknesses. In Section 4.2, we address the identified limitations, showing theoretical ideas to make WMI-PA weight-structure aware. We implement these ideas in our novel algorithm, SAE4WMI. Section 4.3 is devoted to an empirical evaluation of SAE4WMI, comparing it with existing approaches on both synthetic and real-world benchmarks. In Section 4.4, we highlight how PA-based approaches are agnostic to the integrator chosen to compute the volume of the polytopes and propose an adaptation of the algorithm to deal with approximate integration. Our conclusions are presented in Section 4.5.

4.1 Analysis of the Limitations of State-of-the-Art WMI Solvers

We first present an analysis of KC and WMI-PA approaches for WMI.

4.1.1 Knowledge Compilation

We start our analysis by noticing a major problem with existing KC approaches for WMI [72, 120]. These approaches can easily blow up in space even with simple weight functions. Let $\mathbf{x}$ be a set of real variables and $\mathbf{A}$ a set of Boolean atoms. Consider

$$w(\mathbf{x}, \mathbf{A}) \stackrel{\text{def}}{=} \prod_{i=1}^{N} (\mathsf{If}\ \psi_i\ \mathsf{Then}\ w_{i1}(\mathbf{x})\ \mathsf{Else}\ w_{i2}(\mathbf{x})) \tag{4.1}$$

where the ψ_i are $\mathcal{LRA}$-conditions on $\mathbf{x}$ and $\mathbf{A}$ and w_{i1}, w_{i2} are generic functions on $\mathbf{x}$.

First, XADDs [121] do not interleave arithmetic and conditional operators; rather, they push all the arithmetic operators below the conditional ones. Thus, with (4.1) the resulting decision diagrams consist of 2^N branches on the ψ_i's, each corresponding to a distinct unconditioned weight function of the form $\prod_{i=1}^{N} w_{ij_i}(\mathbf{x})$ s.t. $j_i \in \{1, 2\}$. See Figure 4.1 for an example with $N = 3$ and $\psi_i \stackrel{\text{def}}{=} A_i$, $i \in \{1, 2, 3\}$.

Second, if decision diagrams are built on the Boolean abstraction of $w(\mathbf{x}, \mathbf{A})$, e.g. [72], they do not eliminate a priori the branches representing $\mathcal{LRA}$-unsatisfiable combinations of ψ_i's, which can be up to exponentially many.

With WMI-PA, instead, the representation of (4.1) does not grow in size, because $\mathsf{FIUC}^{\mathcal{LRA}}$ functions allow for interleaving arithmetic and conditional operators. Also, SMT enumeration does not produce $\mathcal{LRA}$-unsatisfiable assignments on the ψ_i's.

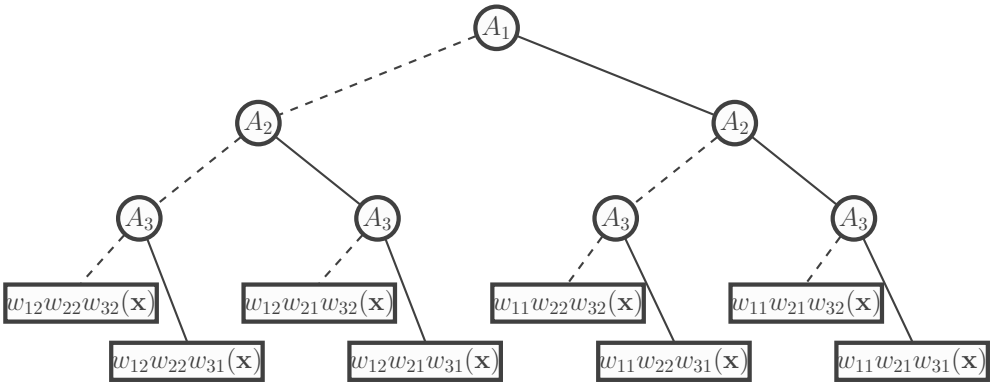


Figure 4.1: Example of exponential growth of XADDs for a weight function as in (4.1) with $N = 3$ and $\psi_i \stackrel{\text{def}}{=} A_i$, $i \in \{1, 2, 3\}$. Decision XADD nodes, i.e., if-then-else conditions, are circles labeled with decision atoms. Solid and dashed edges represent true and false branches, respectively. Square leaf nodes contain $\mathsf{FI}^{\mathcal{LRA}}$ weight functions.

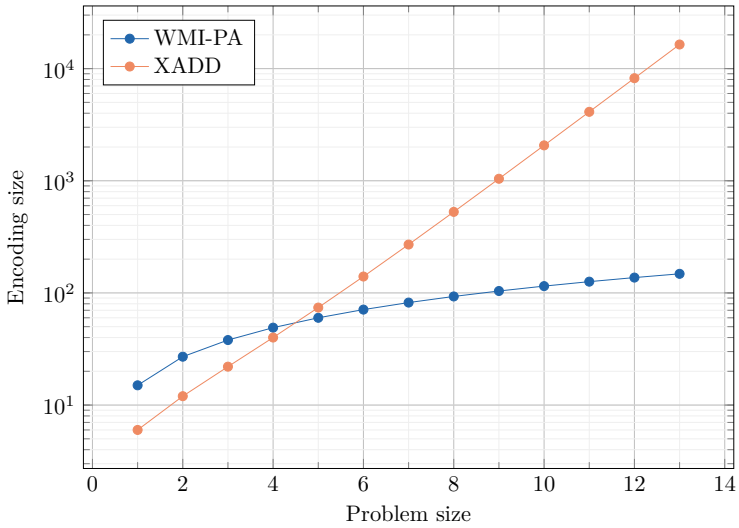


Figure 4.2: Size of XADD diagram (number of nodes) and WMI-PA formula as the problem size increases, in logarithmic scale. Whereas the size of the diagram grows exponentially, WMI-PA encodes the problem into a formula of linear size.

This behavior has been empirically confirmed and is shown in Figure 4.2, where we plot on a logarithmic scale the number of nodes of a KC-based encoding of (4.1) using XADD for problems of increasing size, compared with the size of the encoding used by WMI-PA. This graph clearly shows the exponential blow-up of XADD size, whereas the size of the encoding used by WMI-PA grows linearly. We stress the fact that (4.1) is not an artificial scenario; for example, it occurs in the real-world logistics problems in [161].

Similar considerations can be made for other KC-based approaches, e.g., XSDDs [120].

4.1.2 WMI-PA

We now highlight a major problem with WMI-PA: unlike the KC approaches, *it fails to leverage the conditional structure of the weight function to efficiently enumerate the assignments to its conditions*. We illustrate the issue by means of a simple example.

Example 4.1.1. Let $\varphi \stackrel{\text{def}}{=} \top$, $\chi \stackrel{\text{def}}{=} (0 \leq x_1 \leq 2) \wedge (0 \leq x_2 \leq 3)$, and

$$w(\mathbf{x}, \mathbf{A}) \stackrel{\text{def}}{=} \mathsf{If}\ A_1\ \mathsf{Then}\ \Big[\mathsf{If}\ (x_1 \geq 1)\quad \mathsf{Then}\quad \Big(\mathsf{If}\ (x_2 \geq 1)\ \mathsf{Then}\ x_1^2 x_2\ \mathsf{Else}\ x_1^3 x_2\Big)$$
$$\mathsf{Else}\quad \Big(\mathsf{If}\ (x_2 \geq 2)\ \mathsf{Then}\ x_1 x_2^2\ \mathsf{Else}\ x_1 x_2^3\Big)\Big]$$
$$\mathsf{Else}\ \ (\mathsf{If}\ A_2\ \mathsf{Then}\ 3x_1x_2\ \mathsf{Else}\ 2x_1x_2)\,.$$

To compute $\mathsf{WMI}\Big(\varphi \wedge \chi, w \;\Big|\; \mathbf{x}, \mathbf{A}\Big)$, in principle only six integrals need to be computed, corresponding to the six branches of w.

$$\begin{aligned}
x_1^2 x_2 &\text{ on } (1 \leq x_1 \leq 2) \wedge (1 \leq x_2 \leq 3) (\text{if } A_1 = \top)\\
x_1^3 x_2 &\text{ on } (1 \leq x_1 \leq 2) \wedge (0 \leq x_2 \leq 1) (\text{if } A_1 = \top)\\
x_1 x_2^2 &\text{ on } (0 \leq x_1 \leq 1) \wedge (2 \leq x_2 \leq 3) (\text{if } A_1 = \top)\\
x_1 x_2^3 &\text{ on } (0 \leq x_1 \leq 1) \wedge (0 \leq x_2 \leq 2) (\text{if } A_1 = \top)\\
2x_1x_2 &\text{ on } (0 \leq x_1 \leq 2) \wedge (0 \leq x_2 \leq 3) (\text{if } A_1 = \bot, A_2 = \top)\\
3x_1x_2 &\text{ on } (0 \leq x_1 \leq 2) \wedge (0 \leq x_2 \leq 3) (\text{if } A_1 = \bot, A_2 = \bot)
\end{aligned}$$

However, WMI-PA (Algorithm 3) proceeds as follows. It first labels the conditions in w with fresh Boolean atoms $\{B_1, B_2, B_3\}$, obtaining:

$$\varphi^* \stackrel{\text{def}}{=} \varphi \wedge \chi \wedge (B_1 \leftrightarrow (x_1 \geq 1)) \wedge (B_2 \leftrightarrow (x_2 \geq 1)) \wedge (B_3 \leftrightarrow (x_2 \geq 2)),$$

$$w^*(\mathbf{x}, \mathbf{A} \cup \mathbf{B}) \stackrel{\text{def}}{=} \mathsf{If}\ A_1\ \mathsf{Then}\ \Big[\mathsf{If}\ B_1\quad \mathsf{Then}\quad \Big(\mathsf{If}\ B_2\ \mathsf{Then}\ x_1^2 x_2\ \mathsf{Else}\ x_1^3 x_2\Big)$$
$$\mathsf{Else}\quad \Big(\mathsf{If}\ B_3\ \mathsf{Then}\ x_1 x_2^2\ \mathsf{Else}\ x_1 x_2^3\Big)\Big]$$
$$\mathsf{Else}\ \ (\mathsf{If}\ A_2\ \mathsf{Then}\ 3x_1x_2\ \mathsf{Else}\ 2x_1x_2)\,.$$

Then, by applying $\mathcal{TTA}(\exists \mathbf{x}.\varphi^*)$ we obtain 24 total assignments $M^{\mathbf{A}^*}$ on $\mathbf{A} \cup \mathbf{B}$, i.e., all the truth assignments on $\mathbf{A} \cup \mathbf{B}$ excluding those containing $\{\neg B_2, B_3\}$, which are $\mathcal{LRA}$-unsatisfiable according to $\varphi^*(\mathbf{x}, \mathbf{A} \cup \mathbf{B})$.

As a result, WMI-PA computes 24 integrals instead of 6. In particular, WMI-PA computes some identical integrals multiple times, e.g., the truth assignments $\{A_1, A_2, B_1, B_2, B_3\}$ and $\{A_1, \neg A_2, B_1, B_2, B_3\}$ both correspond to the same integral of $x_1^2 x_2$ over $(1 \leq x_1 \leq 2) \wedge (2 \leq x_2 \leq 3)$.

Also, it splits each integral of $x_1^2 x_2$ and $x_1 x_2^3$ into two parts, based on the irrelevant truth values of B_3 and B_2, respectively, and each integral of $2x_1x_2$ and $3x_1x_2$ into six parts, based on the eight irrelevant truth values of B_1, B_2, B_3 minus the two $\mathcal{LRA}$-unsatisfiable ones containing $\{\neg B_2, B_3\}$.

A key issue with WMI-PA is that the enumeration of $\mathcal{TTA}(\exists \mathbf{x}.\varphi^*)$ in Algorithm 3 line 3 *is not aware of the conditional structure of* w^*, and in particular, it is not aware that often *partial* assignments to the set of conditions in w^* are sufficient to identify a branch of w^*. Instead, it enumerates all the $\mathcal{LRA}$-satisfiable total assignments extending them. (For example, in Example 4.1.1, $\{A_1, B_1, B_2\}$ suffices to identify $x_1^2 x_2$, regardless of the values of A_2 and B_3, but WMI-PA enumerates $\{A_1, A_2, B_1, B_2, B_3\}$, $\{A_1, A_2, B_1, B_2, \neg B_3\}$, $\{A_1, \neg A_2, B_1, B_2, B_3\}$, $\{A_1, \neg A_2, B_1, B_2, \neg B_3\}$.) Enumerating *total* truth assignments on $\mathbf{A} \cup \mathbf{B}$ has two main issues.

First, enumerating total assignments on $\mathbf{B}$ forces the unnecessary partitioning of integrals into subregions. In fact, forcing the enumerator to assign a truth value to *all* the $B_i \in \mathbf{B}$ also forces the solver to assign a truth value to all $\mathcal{LRA}$-conditions $\psi_i \in \mathbf{\Psi} \setminus \mathbf{A}$, even when the conditional structure of w does not need it. Since $\mathbf{\Psi}$ can contain $\mathcal{LRA}$-conditions, a truth assignment on them identifies a subregion of the solution space. The more conditions we assign, the more we split the solution space into subregions, and the more integrals we compute. For example, in Example 4.1.1, the integral of $x_1^2 x_2$ is unnecessarily split into two integrals for $(x_2 \geq 2)$ and $\neg(x_2 \geq 2)$ due to the unnecessary branch on B_3.

Second, enumerating total assignments on $\mathbf{A}$ causes the unnecessary duplication of integrals. In fact, not knowing w^*, the enumerator is always forced to assign *all* original Boolean atoms in $\mathbf{A}$, even when the combination $\mu^{\mathbf{A}} \cup \mu^{\mathbf{B}}$ of a *partial* assignment $\mu^{\mathbf{A}}$ on $\mathbf{A}$ and a total assignment $\mu^{\mathbf{B}}$ on $\mathbf{B}$ suffices to satisfy φ^* and to make w^* $\mathsf{FI}^{\mathcal{LRA}}$, that is, $\mathcal{TTA}(\exists \mathbf{x}.\varphi^*)$ is used instead of $\mathcal{TA}(\exists \mathbf{x}.\varphi^*)$ in (1.3b) and in line 3 of Algorithm 3. For example, in Example 4.1.1, each of the integrals of $x_1^2 x_2$ is computed twice, once for $\{A_1, A_2, \ldots\}$ and once for $\{A_1, \neg A_2, \ldots\}$, due to the unnecessary branching on A_2. Although this problem could in principle be fixed by caching the integrals, this could be expensive in terms of both time and memory.

To cope with these issues, we need to modify WMI-PA to make it aware of the conditional structure of w, so that we can best exploit *partial* truth assignments on $\mathbf{A}$ and on the (atoms of the) conditions in w. (One further issue, dealing with the fact that the conditions in $\mathbf{\Psi}$ may not be atomic, will be addressed in Section 4.2.2.)

4.2 A Structure-Aware WMI Algorithm

To address the issues described in Section 4.1, we aim to combine the best of KC approaches —i.e., weight-structure awareness— with the best of PA-based approaches —i.e., SMT enumeration— by introducing weight-structure awareness into PA-based WMI. In Section 4.2.1, we present the general idea for making WMI-PA weight-structure aware, by introducing and exploiting the notion of *conditional skeleton* for w. A preliminary approach and algorithm (namely "SA-WMI-PA"), based on an implicit enumeration of a skeleton, was proposed in [204]. For the sake of compactness, we do not report it here. In Section 4.2.2, we propose our novel and current best approach and algorithm (namely "Structure-Aware Enumeration for WMI", in short "SAE4WMI"), which explicitly generates the skeleton and is both much simpler and more effective than the approach in [204].

4.2.1 General Idea: Exploiting a Conditional Skeleton of w

We start by introducing the following concept.

Definition 4.2.1 (Conditional skeleton of w, $\mathsf{sk}(w)$). Let φ, χ be as in Section 1.4; let $w(\mathbf{x}, \mathbf{A})$ be $\mathsf{FIUC}^{\mathcal{LRA}}$ on the set of conditions $\mathbf{\Psi}$. We call a *conditional skeleton of* w, written $\mathsf{sk}(w)$, any $\mathcal{LRA}$ formula s.t.:

(a) its atoms are all and only those occurring in the conditions in $\mathbf{\Psi}$;

(b) $\mathsf{sk}(w)$ is $\mathcal{LRA}$-valid, so that $\varphi \wedge \chi$ is equivalent to $\varphi \wedge \chi \wedge \mathsf{sk}(w)$;

(c) every *partial* truth assignment μ to (the atoms occurring in) the conditions $\mathbf{\Psi}$ that satisfies $\mathsf{sk}(w)$ is such that $w|_\mu$ is $\mathsf{FI}^{\mathcal{LRA}}$.

Example 4.2.1. Consider the problem in Example 4.1.1. Then the following formula is a conditional skeleton $\mathsf{sk}(w)$ for $w(\mathbf{x}, \mathbf{A})$:

$$
\begin{array}{lll}
(\ \ A_1 \vee \neg A_1) & & \wedge \ //\textit{split on } A_1 \\
(\neg A_1 \vee \ \ (x_1 \geq 1) \vee \neg(x_1 \geq 1)) & & \wedge \ //\textit{if } A_1 \textit{ split on } (x_1 \geq 1) \\
(\neg A_1 \vee \neg(x_1 \geq 1) \vee \ \ (x_2 \geq 1) \vee \neg(x_2 \geq 1)) & & \wedge \ //\textit{if } A_1, \ \ (x_1 \geq 1) \textit{ split on } (x_2 \geq 1) \\
(\neg A_1 \vee \ \ (x_1 \geq 1) \vee \ \ (x_2 \geq 2) \vee \neg(x_2 \geq 2)) & & \wedge \ //\textit{if } A_1, \neg(x_1 \geq 1) \textit{ split on } (x_2 \geq 2) \\
(\ \ A_1 \vee \ \ A_2 \vee \neg A_2) & & \wedge \ //\textit{if } \neg A_1 \textit{ split on } A_2
\end{array}
$$

For example, the partial assignment $\mu \stackrel{\text{def}}{=} \{A_1, (x_1 \geq 1), (x_2 \geq 1)\}$ propositionally satisfies $\mathsf{sk}(w)$, and is such that, in Example 4.1.1, $w|_\mu = x_1^2 x_2$, which is $\mathsf{FI}^{\mathcal{LRA}}$. Notice that we use the verification semantics for satisfiability by partial assignment (see Remark 1.2.1), so that, e.g., $\{A_1\}$ does not satisfy $\mathsf{sk}(w)$, as the residual is logically equivalent to $\top$, but not syntactically equal to it.

Assume we have produced one such formula $\mathsf{sk}(w)$. Unlike WMI-PA, we do *not* replace the conditions $\mathbf{\Psi}$ in $w(\mathbf{x}, \mathbf{A})$ with fresh Boolean atoms $\mathbf{B}$, and we conjoin to $\varphi \wedge \chi$ the skeleton $\mathsf{sk}(w)$ instead of $\bigwedge_{k=1}^{K}(B_k \leftrightarrow \psi_k)$. The latter formula requires enumerating *total* assignments to $\mathbf{\Psi}$ (i.e., to $\mathbf{B}$), whereas $\mathsf{sk}(w)$ allows us to enumerate *partial* assignments μ to $\mathbf{\Psi}$ that verify condition (c), i.e., assignments for which $w|_\mu$ is $\mathsf{FI}^{\mathcal{LRA}}$. Thus, we can rewrite (1.3b)-(1.3a) as:

$$
\mathsf{WMI}\Big(\varphi, w \ \Big|\ \mathbf{x}, \mathbf{A}\Big) = \sum_{\mu \stackrel{\text{def}}{=} \mu^{\mathbf{A}} \cup \mu^{\mathcal{LRA}} \in \mathcal{TA}(\varphi^{\mathsf{sk}})} 2^{|\mathbf{A} \setminus \mu^{\mathbf{A}}|} \cdot \mathsf{WMI}_{\mathsf{nb}}\Big(\mu^{\mathcal{LRA}}, w|_\mu \ \Big|\ \mathbf{x}\Big) \tag{4.2}
$$

$$
\varphi^{\mathsf{sk}} \stackrel{\text{def}}{=} \varphi \wedge \chi \wedge \mathsf{sk}(w) \tag{4.3}
$$

(Notice that in (4.2) μ is in $\mathcal{TA}(\varphi^{\mathsf{sk}})$, not $\mathcal{TTA}(\varphi^{\mathsf{sk}})$, that is, we enumerate *partial* assignments μ for φ^{sk} and, in particular, *partial* assignments for $\mathbf{\Psi}$.)

Item (c) in Definition 4.2.1 guarantees that $\mathsf{WMI}_{\mathsf{nb}}\Big(\mu^{\mathcal{LRA}}, w|_\mu \ \Big|\ \mathbf{x}\Big)$ in (4.2) can be directly computed, without any further branching on the conditions in $\mathbf{\Psi}$. The

$2^{|\mathbf{A}\setminus\mu^{\mathbf{A}}|}$ factor in (4.2), where $|\mathbf{A}\setminus\mu^{\mathbf{A}}|$ is the number of Boolean atoms in $\mathbf{A}$ that are not assigned by $\mu^{\mathbf{A}}$, accounts for the fact that, if $A_i \in \mathbf{A}$ is not assigned in $\mu^{\mathbf{A}}$, then $\mathsf{WMI}_{\mathsf{nb}}\big(\mu^{\mathcal{LRA}}, w|_\mu \;\big|\; \mathbf{x}\big)$ must be counted twice, because μ represents two assignments $\mu\cup\{A_i\}$ and $\mu\cup\{\neg A_i\}$ which would produce two identical integrals, because their $\mathcal{LRA}$ component and the branch of the weight function they identify are identical. When $|\mathbf{A}\setminus\mu^{\mathbf{A}}| > 0$, this allows us to compute only one integral rather than $2^{|\mathbf{A}\setminus\mu^{\mathbf{A}}|}$ integrals (!).

Notice that the latter is only one of the computational advantages of (4.2)-(4.3) w.r.t. (1.3b)-(1.3a): even more importantly, (4.2)-(4.3) allow for enumerating partial assignments also on the $\mathcal{LRA}$-conditions in $\mathbf{\Psi}\setminus\mathbf{A}$, reducing the number of integrals to compute for each partial assignment by up to a 2^j factor, j being the number of unassigned $\mathcal{LRA}$ atoms in μ. (In this case, no multiplication factor should be considered, because an unassigned $\mathcal{LRA}$ condition ψ_i in μ causes the merging of the two integration domains of $\mu\cup\{\psi_i\}$ and $\mu\cup\{\neg\psi_i\}$.)

Example 4.2.2. Consider the problem in Example 4.1.1 and the corresponding $\mathsf{sk}(w)$ formula in Example 4.2.1. Let $\varphi^{\mathsf{sk}} \stackrel{\text{def}}{=} \varphi\wedge\chi\wedge\mathsf{sk}(w)$, as in (4.3). We show how $\mathcal{TA}(\varphi^{\mathsf{sk}})$ in (4.2) can be enumerated. Assume nondeterministic choices are picked following the order $A_1, A_2, (x_1\geq 1), (x_2\geq 1), (x_2\geq 2)$, assigning positive values first. Then in the first branch the following *total* truth assignment is generated:

$$\{(x_1\geq 0),(x_1\leq 2),(x_2\geq 0),(x_2\leq 3)\}\cup\{A_1,A_2,(x_1\geq 1),(x_2\geq 1),(x_2\geq 2)\}$$

(where $\{(x_1\geq 0),(x_1\leq 2),(x_2\geq 0),(x_2\leq 3)\}$ is assigned deterministically by $\varphi\wedge\chi$), from which the solver extracts the minimal partial assignment satisfying $\mathsf{sk}(w)$:

$$\mu_1 \stackrel{\text{def}}{=} \{(x_1\geq 0),(x_1\leq 2),(x_2\geq 0),(x_2\leq 3)\}\cup\{A_1,(x_1\geq 1),(x_2\geq 1)\}.$$

In the next branch, the following minimal partial assignment is produced:

$$\mu_2 \stackrel{\text{def}}{=} \{(x_1\geq 0),(x_1\leq 2),(x_2\geq 0),(x_2\leq 3)\}\cup\{A_1,(x_1\geq 1),\neg(x_2\geq 1)\}.$$

The assignments μ_1 and μ_2 give opposite truth values to at least one atom. Overall, the algorithm enumerates the following collection of minimal partial assignments:

$$\begin{aligned}
&\{(x_1\geq 0),(x_1\leq 2),(x_2\geq 0),(x_2\leq 3)\}\cup\{\ A_1,\ (x_1\geq 1),\ (x_2\geq 1)\},\\
&\{(x_1\geq 0),(x_1\leq 2),(x_2\geq 0),(x_2\leq 3)\}\cup\{\ A_1,\ (x_1\geq 1),\neg(x_2\geq 1)\},\\
&\{(x_1\geq 0),(x_1\leq 2),(x_2\geq 0),(x_2\leq 3)\}\cup\{\ A_1,\neg(x_1\geq 1),\ (x_2\geq 2)\},\\
&\{(x_1\geq 0),(x_1\leq 2),(x_2\geq 0),(x_2\leq 3)\}\cup\{\ A_1,\neg(x_1\geq 1),\neg(x_2\geq 2)\},\\
&\{(x_1\geq 0),(x_1\leq 2),(x_2\geq 0),(x_2\leq 3)\}\cup\{\neg A_1,\ A_2\},\\
&\{(x_1\geq 0),(x_1\leq 2),(x_2\geq 0),(x_2\leq 3)\}\cup\{\neg A_1,\neg A_2\}
\end{aligned}$$

which correspond to the six integrals of Example 4.1.1. Notice that, according to (4.2), the first four integrals have to be multiplied by 2, because the partial assignment $\{A_1\}$ covers two total assignments $\{A_1,A_2\}$ and $\{A_1,\neg A_2\}$.

Algorithm 5 $\mathsf{Convert_{sk}}(w, \mathsf{conds})$ // conds is a set of literal conditions

1: **if** ($\{w$ is a constant or a polynomial$\}$) **then**
2: **return** $\top$
3: **end if**
4: **if** $(w == (w_1 \bowtie w_2), \bowtie \in \{+, -, \cdot\})$ **then**
5: **return** $\mathsf{Convert_{sk}}(w_1, \mathsf{conds}) \wedge \mathsf{Convert_{sk}}(w_2, \mathsf{conds})$
6: **end if**
7: **if** $(w == g(w_1, \ldots, w_k)$, g unconditioned) **then**
8: **return** $\bigwedge_{i=1}^{k} \mathsf{Convert_{sk}}(w_i, \mathsf{conds})$
9: **end if**
10: **if** $(w == (\mathsf{If}\ \psi\ \mathsf{Then}\ w_1\ \mathsf{Else}\ w_2))$ **then**
11: **if** ψ is a literal **then**
12: $\mathsf{branch} \leftarrow \bigvee_{\psi_i \in \mathsf{conds}} \neg\psi_i \vee \psi \vee \neg\psi$
13: $\mathsf{defs}_1 \leftarrow \mathsf{Convert_{sk}}(w_1, \mathsf{conds} \cup \{\psi\})$
14: $\mathsf{defs}_2 \leftarrow \mathsf{Convert_{sk}}(w_2, \mathsf{conds} \cup \{\neg\psi\})$
15: **return** $\mathsf{branch} \wedge \mathsf{defs}_1 \wedge \mathsf{defs}_2$
16: **else**
17: **let** B be a fresh Boolean atom
18: $\mathsf{branch} \leftarrow [\![\bigvee_{\psi_i \in \mathsf{conds}} \neg\psi_i \vee (B \vee \neg B) \vee \mathsf{CNF_{PG}}(\mathsf{NNF}(B \leftrightarrow \psi))]\!]$
19: $\mathsf{defs}_1 \leftarrow \mathsf{Convert_{sk}}(w_1, \mathsf{conds} \cup \{B\})$
20: $\mathsf{defs}_2 \leftarrow \mathsf{Convert_{sk}}(w_2, \mathsf{conds} \cup \{\neg B\})$
21: **return** $\mathsf{branch} \wedge \mathsf{defs}_1 \wedge \mathsf{defs}_2$
22: **end if**
23: **end if**

4.2.2 Encoding a Conditional Skeleton Explicitly

Notation. If $\phi \stackrel{\text{def}}{=} \bigwedge_i C_i$ is a CNF formula and $C \stackrel{\text{def}}{=} \bigvee_j l_j$ is a clause, then the formula $\bigwedge_i (C \vee C_i)$ is shortened into $[\![C \vee \phi]\!]$.

Encoding. Algorithm 5 shows the procedure for building a formula $\mathsf{sk}(w)$. Intuitively, the output consists of a conjunction of formulas, each stating "*if all the conditions $\psi_1, \ldots, \psi_i$ of the current subbranch in w hold, then split on the next condition ψ_{i+1} in the branch*". For example, for the problem in Example 4.1.1, Algorithm 5 produces as $\mathsf{sk}(w)$ the formula reported in Example 4.2.1, whose satisfying partial assignments can be enumerated as in Example 4.2.2.

The recursive procedure $\mathsf{Convert_{sk}}(w_i, \mathsf{conds})$ takes a subtree w_i of w and the set of literals conds under whose scope w_i occurs, and returns the CNF representation of $\bigwedge_{\psi_i \in \mathsf{conds}} \psi_i \rightarrow \mathsf{sk}(w_i)$ (i.e., it returns $[\![(\bigvee_{\psi_i \in \mathsf{conds}} \neg\psi_i) \vee \mathsf{sk}(w_i)]\!]$). Hence, $\mathsf{Convert_{sk}}(w, \emptyset)$ returns $\mathsf{sk}(w)$. The set conds contains the conditions that need to be true in order to activate the current branch. The recursive encoding of $\mathsf{sk}(w)$ works as follows.

Lines 1–2 handle the base cases. Constants and polynomials are conditionless, so they can be encoded as $\top$. Here conds plays no role, as $(\bigwedge_{\psi_i \in \mathsf{conds}} \psi_i) \rightarrow \top$ reduces to $\top$.

Lines 4–5 handle arithmetic operators $w_1 \bowtie w_2$, with $\bowtie \in \{+, -, \cdot\}$. Since we must

ensure that the conditions in both terms w_1 and w_2 are enumerated, we conjoin the results of recursively applying the conversion procedure to w_1 and w_2.

Similarly, in lines 7–8, for a function $g(w_1, \dots, w_k)$ we must ensure that the conditions in all terms $w_1, \dots, w_k$ are enumerated. Hence, we conjoin the results of recursively applying the conversion procedure to each argument $w_1, \dots, w_k$.

Finally, lines 10–21 handle conditional terms $\mathsf{If}\ \psi\ \mathsf{Then}\ w_1\ \mathsf{Else}\ w_2$. Let us first assume that ψ is a literal (lines 10–15). When all conditions in conds are true, the condition ψ must be split upon. This is captured in line 12 by the valid clause[1]

$$\bigvee_{\psi_i \in \mathsf{conds}} \neg\psi_i \vee \psi \vee \neg\psi.$$

Moreover, when all conds are true and ψ is true, all branches of w_1 must be enumerated. Hence, in line 13 we recursively call the procedure on w_1 after adding ψ to conds. This call returns $[\![\bigvee_{\psi_i \in \mathsf{conds}} \neg\psi_i \vee \neg\psi \vee \mathsf{sk}(w_1)]\!]$. Symmetrically, in line 14 we recursively call the procedure on w_2 after adding $\neg\psi$ to conds, yielding $[\![\bigvee_{\psi_i \in \mathsf{conds}} \neg\psi_i \vee \psi \vee \mathsf{sk}(w_2)]\!]$.

The case where ψ is not a literal (lines 17–21) is discussed in the next paragraph.

Unlike KC approaches, $\mathsf{sk}(w)$ grows linearly in the number of branches of w. For example, with (4.1), $\mathsf{sk}(w)$ is $\bigwedge_{i=1}^{N}(\psi_i \vee \neg\psi_i)$, whose size grows linearly with N.

Dealing with non-literal conditions. In general, conditions in $\mathbf{\Psi}$ are not necessarily single literals. For example, they can be *range conditions* in the form $(l \leq x_i) \wedge (x_i \leq u)$ or more complex conditions. In these cases, $\bigvee_{\psi_i \in \mathsf{conds}} \neg\psi_i \vee \psi \vee \neg\psi$ (line 12) would not be a clause, and hence $\mathsf{sk}(w)$ would not be in CNF.

This would force the SMT enumerator to convert $\mathsf{sk}(w)$ into CNF, typically using the Tseitin CNF-ization [222], by introducing extra label atoms $\mathbf{B}$, which are then projected away during enumeration. In Chapter 3, we discussed how this prevents the enumeration of short partial assignments, which for WMI translates into the computation of more integrals than necessary, as illustrated in the following example.

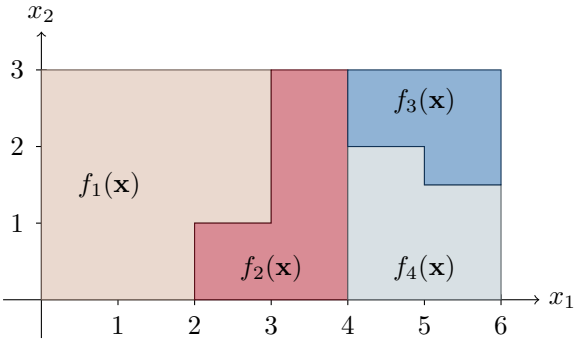
Example 4.2.3. Let $\varphi \stackrel{\text{def}}{=} \top$, $\chi \stackrel{\text{def}}{=} (x_1 \geq 0) \wedge (x_1 \leq 6) \wedge (x_2 \geq 0) \wedge (x_2 \leq 3)$, and

$$w(\mathbf{x}, \mathbf{A}) = \mathsf{If}\ (x_1 \leq 4)\ \ \mathsf{Then}\ [\mathsf{If}\ \psi_1\ \mathsf{Then}\ f_1(\mathbf{x})\ \mathsf{Else}\ f_2(\mathbf{x})],\ \mathsf{Else}\ [\mathsf{If}\ \psi_2\ \mathsf{Then}\ f_3(\mathbf{x})\ \mathsf{Else}\ f_4(\mathbf{x})]$$
$$\psi_1 \stackrel{\text{def}}{=} (x_1 \leq 2) \vee ((x_1 \leq 3) \wedge (x_2 > 1)), \quad \psi_2 \stackrel{\text{def}}{=} (x_2 > 2) \vee ((x_1 > 5) \wedge (x_2 > \tfrac{3}{2}))$$

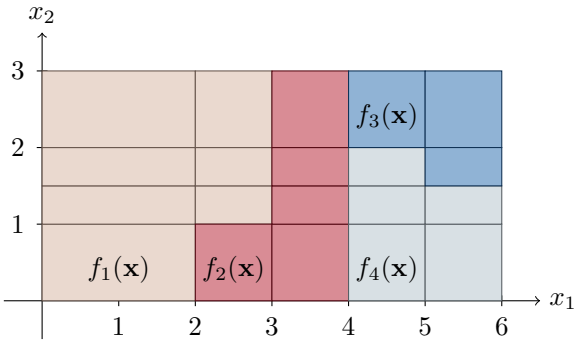
such that the functions $f_i(\mathbf{x})$ are $\mathsf{FI}^{\mathcal{LRA}}$. w is graphically represented in Figure 4.3a.

SA-WMI-PA [204], using either its implicit $\mathsf{sk}(w)$ as in the original algorithm or the above-mentioned non-CNF explicit $\mathsf{sk}(w)$, enumerates 11 convex regions, as shown in Figure 4.3c. Notice that the brown, red and grey areas are each split into three regions, whereas it is possible to split each into only two convex regions.

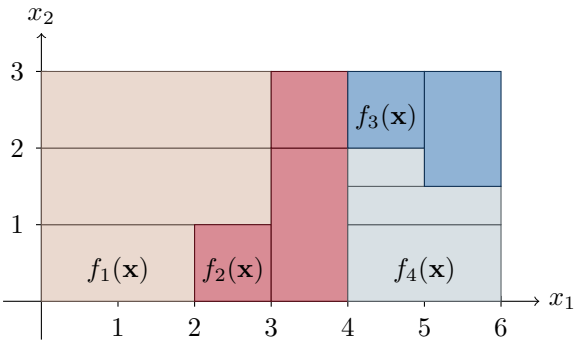
[1] Since the SMT enumerator will detect these tautologies and ignore these clauses, we actually add $\bigvee_{\psi_i \in \mathsf{conds}} \neg\psi_i \vee (B \leftrightarrow \psi)$, where B is a fresh Boolean atom which is projected away during enumeration.



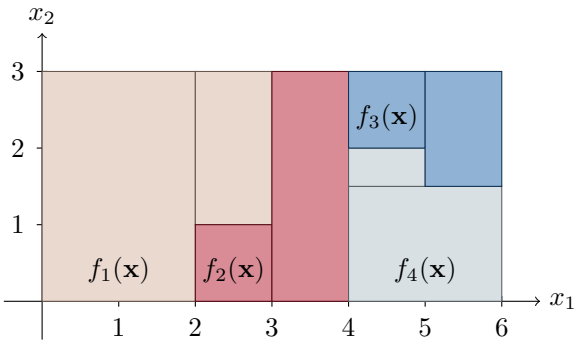
(a) w and its 4 non-convex $\mathsf{FI}^{\mathcal{LRA}}$ branches.



(b) WMI-PA computes 20 distinct integrals.



(c) SA-WMI-PA computes 11 distinct integrals.



(d) SAE4WMI computes 8 distinct integrals.

Figure 4.3: Graphical representation of the effectiveness of different SMT-based WMI solvers on Example 4.2.3. SAE4WMI divides the integration domain into the minimum number of convex subdomains, each with a $\mathsf{FI}^{\mathcal{LRA}}$ weight.

To cope with this problem, we use the $\mathsf{NNF}{+}\mathsf{CNF}_{\mathsf{PG}}$ encoding presented in Section 3.2, but make the "labeling definitions" local to the conditions in conds, so that when the branch is not "selected", the definitions are simplified to $\top$ and do not force the assignment of the atoms in the definition. Thus, in Algorithm 5, lines 17-21, we substitute each non-literal condition ψ in branch with a fresh Boolean atom B and add

$$[\![\bigvee_{\psi_i \in \mathsf{conds}} \neg\psi_i \vee (B \vee \neg B) \vee \mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(B \leftrightarrow \psi))]\!]$$

(We recall that all conditions ψ_i in conds are literals by construction, see Algorithm 5.)

In Figure 4.3d, we show that using the "local" CNF version of $\mathsf{sk}(w)$ we enumerate 8 regions, instead of the 11 ones in Figure 4.3c. Here, 8 is also the minimum number of convex regions into which we can split the problem, since the 4 colored regions in Figure 4.3a are non-convex, so the integral over each region cannot be computed directly.

We stress the importance of the locality of labeling clauses in the enumeration of truth assignments: the SMT enumerator needs to branch on B (hence, on ψ) *only in those branches in which all* conds *are true*, that is, only in those branches where a truth value for ψ is needed to define a branch of w. Notice that, if $\mathsf{CNF}_{\mathsf{PG}}$ introduces some labeling definitions, the latter are also active only when all conditions in conds are true.

Algorithm 6 $\mathsf{SAE4WMI}(\varphi, w, \mathbf{x}, \mathbf{A})$

1: $\mathsf{sk}(w) \leftarrow \mathsf{Convert}_{\mathsf{sk}}(w, \emptyset)$
2: $\varphi^{\mathsf{sk}} \leftarrow \varphi \wedge \chi \wedge \mathsf{sk}(w)$
3: $M^{\mathbf{A}} \leftarrow \mathcal{TA}(\exists \mathbf{x}.\varphi^{\mathsf{sk}})$
4: $wmi \leftarrow 0$
5: **for** $\mu^{\mathbf{A}} \in M^{\mathbf{A}}$ **do**
6: $\quad k \leftarrow |\mathbf{A} \setminus \mu^{\mathbf{A}}|$
7: $\quad \mathsf{Simplify}(\varphi^{\mathsf{sk}}|_{\mu^{\mathbf{A}}})$
8: $\quad$ **if** $\varphi^{\mathsf{sk}}|_{\mu^{\mathbf{A}}}$ is a cube **then**
9: $\quad\quad wmi \leftarrow wmi + 2^k \cdot \mathsf{WMI}_{\mathsf{nb}}(\varphi^{\mathsf{sk}}|_{\mu^{\mathbf{A}}}, w|_{\mu^{\mathbf{A}}}|\mathbf{x})$
10: $\quad$ **else**
11: $\quad\quad \mathcal{M} \leftarrow \mathcal{TA}(\varphi^{\mathsf{sk}}|_{\mu^{\mathbf{A}}})$
12: $\quad\quad$ **for** $(\mu \stackrel{\text{def}}{=} \mu^{\mathbf{A}'} \wedge \mu^{\mathcal{LRA}}) \in \mathcal{M}$ **do**
13: $\quad\quad\quad k' \leftarrow k - |\mu^{\mathbf{A}'}|$
14: $\quad\quad\quad wmi \leftarrow wmi + 2^{k'} \cdot \mathsf{WMI}_{\mathsf{nb}}(\mu^{\mathcal{LRA}}, w|_{\mu^{\mathbf{A}} \wedge \mu}|\mathbf{x})$
15: $\quad\quad$ **end for**
16: $\quad$ **end if**
17: **end for**
18: **return** wmi

4.2.3 The SAE4WMI Procedure

We devise a new algorithm, SAE4WMI, which improves over SA-WMI-PA [204] by using the explicit skeleton of Section 4.2.2, and further improves the enumeration, as described below. The procedure is outlined in Algorithm 6. (To simplify the narration, here we assume that no new Boolean atom $\mathbf{B}$ is introduced, see Section 4.2.2; otherwise, $\mathcal{TA}(\exists \mathbf{x}.\varphi^{\mathsf{sk}})$ and $\mathcal{TA}(\varphi^{\mathsf{sk}}|_{\mu^{\mathbf{A}}})$ are replaced with $\mathcal{TA}(\exists \mathbf{x}.\exists \mathbf{B}.\varphi^{\mathsf{sk}})$ and $\mathcal{TA}(\exists \mathbf{B}.\varphi^{\mathsf{sk}}|_{\mu^{\mathbf{A}}})$, respectively.)

In lines 1–3, after producing $\mathsf{sk}(w)$ by Algorithm 5, we use projected $\mathcal{T}$-AllSAT (see Section 1.3.1) to enumerate a set $M^{\mathbf{A}}$ of partial truth assignments $\mu^{\mathbf{A}}$ over $\mathbf{A}$. This set is defined as $M^{\mathbf{A}} \stackrel{\text{def}}{=} \mathcal{TA}(\exists \mathbf{x}.\varphi^{\mathsf{sk}})$, where $\varphi^{\mathsf{sk}} \stackrel{\text{def}}{=} \varphi \wedge \chi \wedge \mathsf{sk}(w)$.

Next, in lines 5–16, we consider each $\mu^{\mathbf{A}} \in M^{\mathbf{A}}$ and compute and simplify the residual formula $\varphi^{\mathsf{sk}}|_{\mu^{\mathbf{A}}}$. As in WMI-PA and SA-WMI-PA, if $\varphi^{\mathsf{sk}}|_{\mu^{\mathbf{A}}}$ is already a conjunction of literals, its integral can be computed directly and returned after multiplying it by a factor 2^k, where k is the number of Boolean atoms left unassigned by $\mu^{\mathbf{A}}$.[2]

If $\varphi^{\mathsf{sk}}|_{\mu^{\mathbf{A}}}$ is not a conjunction of literals, the algorithm proceeds by enumerating a set M of partial truth assignments $\mu = \mu^{\mathbf{A}'} \cup \mu^{\mathcal{LRA}}$ over the remaining atoms of the formula, such that $M \stackrel{\text{def}}{=} \mathcal{TA}(\varphi^{\mathsf{sk}}|_{\mu^{\mathbf{A}}})$. Since $\mu^{\mathbf{A}}$ is partial, the residual formula $\varphi^{\mathsf{sk}}|_{\mu^{\mathbf{A}}}$ may still contain Boolean atoms, as shown in Example 4.2.4; consequently, μ may include a non-empty Boolean component $\mu^{\mathbf{A}'}$. The algorithm then computes the integral of the $\mathsf{FI}^{\mathcal{LRA}}$ function $w|_{\mu^{\mathbf{A}} \cup \mu}$ over the convex region $\mu^{\mathcal{LRA}}$. This integral is multiplied by $2^{k'}$, where $k' \stackrel{\text{def}}{=} k - |\mu^{\mathbf{A}'}|$ is the number of Boolean atoms not assigned by $\mu^{\mathbf{A}} \cup \mu$.

[2]Notice that, in this case, the conjunction of literals $\varphi^{\mathsf{sk}}|_{\mu^{\mathbf{A}}}$ does not contain Boolean literals, since these would have already been assigned by $\mu^{\mathbf{A}}$.

As with WMI-PA and SA-WMI-PA, in the implementation (lines 3–5 and 10–12) the sets $M^{\mathbf{A}}$ and M are not generated explicitly; rather, their elements are generated, integrated, and then dropped one by one, so as to avoid space blow-up (see Remark 1.4.1).

We remark that producing shorter partial assignments over the Boolean atoms has a positive effect regardless of the structure of w. In fact, as noted in Section 4.1.2, enumerating (nearly) total assignments on $\mathbf{A}$ can lead to computing identical integrals of the same $\mathsf{FI}^{\mathcal{LRA}}$ function over the same polytope. By finding shorter partial assignments, SAE4WMI reduces the number of duplicate integrals, as shown in the following example.

Example 4.2.4. Let $\chi \stackrel{\text{def}}{=} (x \geq 0) \wedge (x \leq 4)$, $w(\mathbf{x}, \mathbf{A}) = 1.0$, and

$$\varphi = (A_1 \vee A_2 \vee A_3) \wedge (\neg A_1 \vee (x \geq 1)) \wedge (\neg A_2 \vee (x \geq 2)) \wedge (\neg A_3 \vee (x \leq 3)).$$

Since $\mathbf{\Psi} = \emptyset$, both $\bigwedge_i B_i \leftrightarrow \psi_i$ and $\mathsf{sk}(w)$ reduce to $\top$, and $\varphi^* = \varphi^{\mathsf{sk}} \stackrel{\text{def}}{=} \varphi \wedge \chi$.
With WMI-PA, $\mathcal{TTA}(\exists \mathbf{x}.\varphi^*)$ directly finds 7 total assignments ($\mu^{\mathbf{A}} \wedge \mu^{\mathbf{A}}_{residual} \wedge \mu^{\mathcal{LRA}}$):

$\mu^{\mathbf{A}}$	$\mu^{\mathbf{A}}_{residual}$	$\mu^{\mathcal{LRA}}$
$\{ A_2,$	$A_1, \quad A_3,$	$(x \geq 2), (x \leq 3)\}$
$\{ A_2,$	$A_1, \neg A_3,$	$(x \geq 2), (x \leq 4)\}$
$\{ A_2,$	$\neg A_1, \quad A_3,$	$(x \geq 2), (x \leq 3)\}$
$\{ A_2,$	$\neg A_1, \neg A_3,$	$(x \geq 2), (x \leq 4)\}$
$\{ A_1, \neg A_2,$	$A_3,$	$(x \geq 1), (x \leq 3)\}$
$\{ A_1, \neg A_2,$	$\neg A_3,$	$(x \geq 1), (x \leq 4)\}$
$\{\neg A_1, \neg A_2, \quad A_3,$		$(x \geq 0), (x \leq 3)\}$

(4.4)

With both SA-WMI-PA and SAE4WMI, suppose $\mathcal{TA}(\exists \mathbf{x}.\varphi^{\mathsf{sk}})$ first finds $\mu^{\mathbf{A}} \stackrel{\text{def}}{=} \{A_2\}$, so that $\varphi^{\mathsf{sk}}|_{\mu^{\mathbf{A}}} \stackrel{\text{def}}{=} (x \geq 0) \wedge (x \leq 4) \wedge (\neg A_1 \vee (x \geq 1)) \wedge (x \geq 2) \wedge (\neg A_3 \vee (x \leq 3))$.

With SA-WMI-PA, $\mathcal{TTA}(\exists \mathbf{x}.\varphi^{\mathsf{sk}}|_{\mu^{\mathbf{A}}})$ enumerates all four total residual assignments on A_1, A_3,[a] duplicating the integrals on $x \in [2, 3]$ and $x \in [2, 4]$ by uselessly case-splitting on A_1 and $\neg A_1$, as shown in the first four rows of (4.4).

With SAE4WMI, $\mathcal{TA}(\varphi^{\mathsf{sk}}|_{\mu^{\mathbf{A}}})$ avoids this case split and enumerates only the 2 partial assignments in the following list ($\mu \stackrel{\text{def}}{=} \mu^{\mathbf{A}'} \wedge \mu^{\mathcal{LRA}}$):[b]

$\mu^{\mathbf{A}}$	$\mu \stackrel{\text{def}}{=} \mu^{\mathbf{A}'} \wedge \mu^{\mathcal{LRA}}$
$\{ A_2,$	$A_3, \quad (x \geq 2), (x \leq 3)\}$
$\{ A_2,$	$\neg A_3, \quad (x \geq 2), (x \leq 4)\}$
$\{ A_1, \neg A_2,$	$\neg A_3, \quad (x \geq 1), (x \leq 4)\}$
$\{\neg A_1, \neg A_2, \quad A_3,$	$(x \geq 0), (x \leq 3)\}$
$\{ A_1, \neg A_2, \quad A_3,$	$(x \geq 1), (x \leq 3)\}$

(4.5)

The two corresponding integrals for the first two assignments in (4.5) are multiplied, respectively, by $2^{3-|\{A_2\}|-|\{A_3\}|} = 2$ and $2^{3-|\{A_2\}|-|\{\neg A_3\}|} = 2$.

[a] Here, $\exists \mathbf{x}.\varphi^{\mathsf{sk}}|_{\mu^{\mathbf{A}}}$ corresponds to a valid Boolean formula on $\{A_1, A_3\}$.
[b] The truth value of A_1 is irrelevant here because $(x \geq 1)$ is forced to be true by $(x \geq 2)$.

The benefit becomes more evident as the number of Boolean atoms increases, since if we avoid computing the $\mathcal{TTA}(\exists \mathbf{x}.\varphi^{\mathsf{sk}}|_{\mu^{\mathbf{A}}})$ over the residual Boolean atoms, then we are potentially largely cutting the number of integrals.

4.3 Experimental Evaluation

In the following experiments, we evaluate SAE4WMI against the current state of the art in WMI, using the same setting as [204]. To this end, we compare the proposed algorithm SAE4WMI and its previous version SA-WMI-PA [204] with the previous SMT-based approach WMI-PA [161] and the state-of-the-art KC-based approaches: XADD[3] [121], XSDD[3] and FXSDD[3] [120]. We evaluate the performance of these solvers on both synthetic and real-world-inspired benchmarks in terms of execution time and, for SMT-based approaches, the number of integrals computed.

4.3.1 Information for the Reproducibility of the Experiments

WMI-PA, SA-WMI-PA, and SAE4WMI use MathSAT [48] for SMT enumeration. As in Section 3.3, to compute $\mathcal{TA}(\ldots)$, we set -dpll.allsat_allow_duplicates=false for disjoint enumeration, -dpll.allsat_minimize_model=true for partial enumeration, and disable non-validity-preserving preprocessing steps via -preprocessor.simplification=0 and -preprocessor.toplevel_propagation=false. For $\mathcal{TTA}(\ldots)$, the second option is set to false.

Integration over convex polytopes is handled by LattE [63] in the SMT-based approaches, whereas KC-based solvers use PSI Algebra for both integration and inconsistency checking [90]. This difference is not penalizing the KC-based approaches, which were shown to achieve worse results when using LattE [120].

All experiments are performed on a 28-core machine with an Intel Xeon Gold 6238R @ 2.20 GHz, 128 GB of RAM, and Ubuntu Linux 24.04. The code of WMI-PA, SA-WMI-PA, and SAE4WMI,[4][5]as well as the benchmarks are available on GitHub.[6]

We set a timeout of 3600 s for the synthetic benchmarks and 1200 s for the DET benchmarks.

4.3.2 Benchmarks

Synthetic benchmarks. Random formulas are obtained using the generator introduced by [161]. In contrast to other recent works [121, 120], these synthetic benchmarks do not contain strong structural regularities, offering a more neutral perspective on how the different techniques are expected to perform on average. We fix the numbers of Boolean and real variables at 3 while varying the depth of the weights in $\{4, 5, 6, 7\}$, for a total of 40 instances.

[3] https://github.com/weighted-model-integration/pywmi

[4] https://github.com/unitn-sml/wmpy/releases/tag/wmipa

[5] https://github.com/unitn-sml/wmpy for an up-to-date version of SAE4WMI

[6] https://github.com/unitn-sml/wmi-benchmarks

Density Estimation Trees benchmarks. We then consider the problem of computing arbitrary algebraic queries over *Density Estimation Trees* (DETs) [182]. DETs are the density estimation equivalent of decision or regression trees, encoding piecewise constant distributions over mixed continuous/discrete domains. Having only univariate conditions in the internal nodes, DETs support efficient density estimation and marginal inference over single variables. Answering algebraic queries over multiple variables, like $\Pr(X \leq Y)$, requires marginalizing over an oblique constraint, which is reduced to:

$$\Pr(X \leq Y) = \frac{\mathsf{WMI}((X \leq Y) \wedge \chi_{DET}, w_{DET})}{\mathsf{WMI}(\chi_{DET}, w_{DET})}. \tag{4.6}$$

This type of query commonly arises when applying WMI-based inference to probabilistic formal verification tasks, when properties involve multiple continuous variables. The support χ_{DET} is obtained by conjoining the lower and upper bounds of each continuous variable, while w_{DET} encodes the density estimate by means of nested if-then-elses and non-negative constant leaves.

Consistently with [204], we train DETs on a selection of hybrid datasets from the UCI repository [74]. The datasets and resulting models are summarized in Appendix B.1. Following the approach of [159], discrete numerical features are relaxed into continuous variables, while n-ary categorical features are one-hot encoded with n binary variables.

DETs are trained on each dataset using the standard greedy procedure [182] with bounds on the number of instances for each leaf set to $[100, 200]$. We investigate the effect of algebraic dependencies by generating increasingly complex queries involving a number of variables $k = \max(1, \lfloor H \cdot |\mathbf{x}| \rfloor)$, where $H \in [0, 1]$ is the ratio of continuous variables appearing in the inequality. For each value of H, 5 random inequalities involving k variables are generated. This yields a total of 82 WMI domains, with 5 queries each.

4.3.3 Results

We report our findings using cactus plots. Each tick on the x-axis corresponds to a single instance, and results on the y-axis are sorted independently for each method. Steeper slopes mean lower efficiency.

Synthetic benchmarks. (Figure 4.4) Consistently with [160, 161], in this setting SMT-based solvers perform better than KC-based solvers (Figure 4.4 left), which suffer from the complex algebraic dependencies between continuous variables. SAE4WMI drastically improves over WMI-PA, in terms of time and number of computed integrals (Figure 4.4 right). As expected, the advantage of structure awareness given by the introduction of the skeleton is eye-catching.

Additionally, SAE4WMI offers an advantage over SA-WMI-PA, although the order of magnitude remains quite the same. Notice that in this setting, the number of Boolean atoms is limited to 3, so the revised enumeration technique of Section 4.2.3 for the Boolean component of the formula does not show its full potential. The benefits of the novel algorithm SAE4WMI with respect to SA-WMI-PA (and WMI-PA) will be evident in the next experiment, where more complex logical structures are involved.

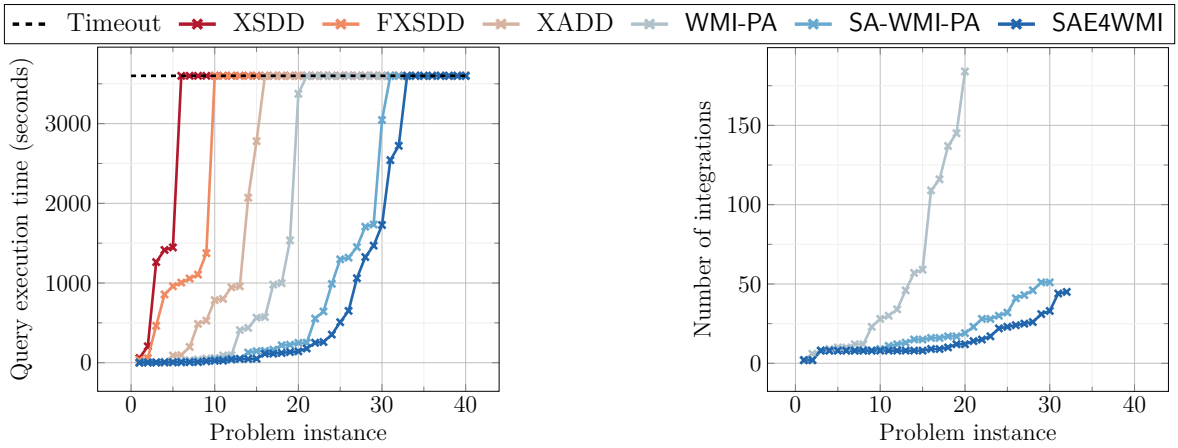
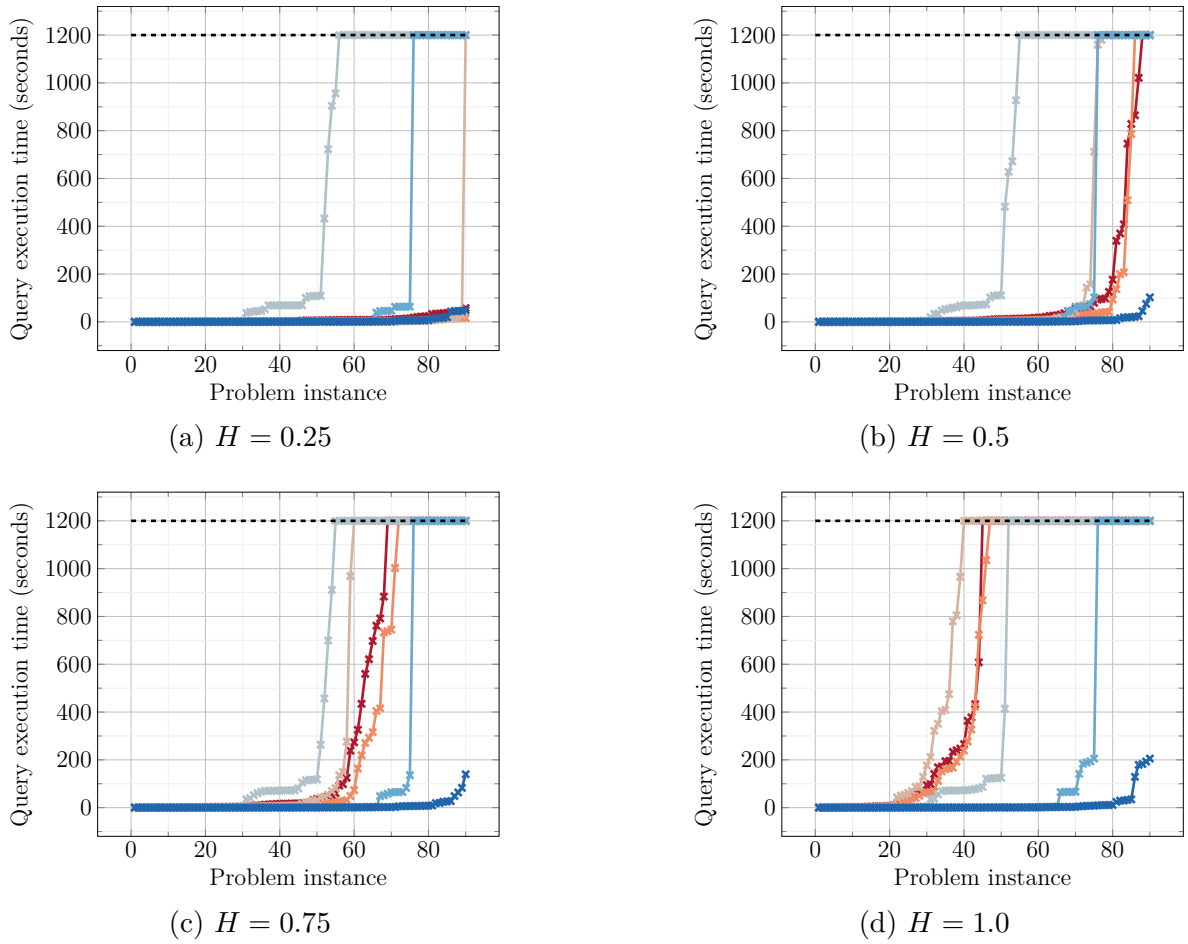


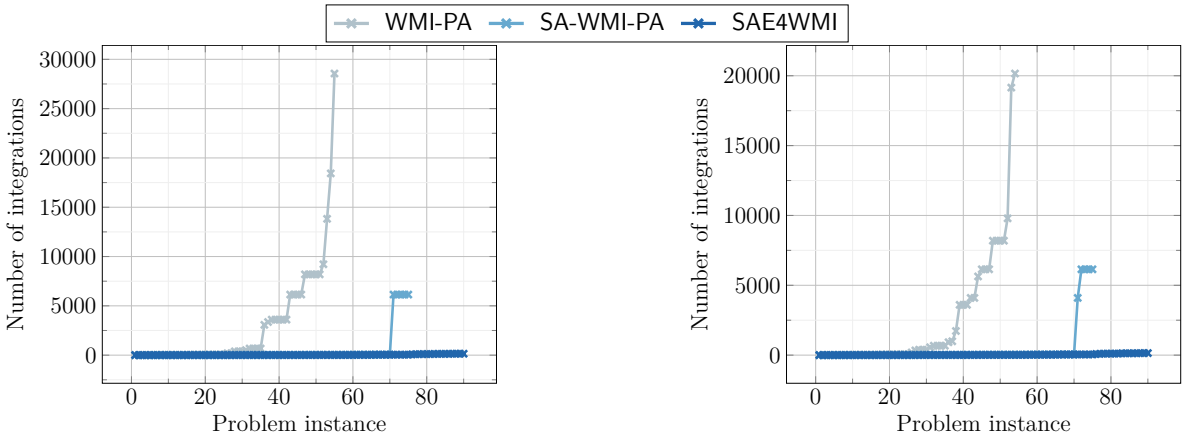
Figure 4.4: Cactus plots of the synthetic experiments reporting execution time (left) for all methods and the number of integrals (right) for WMI-PA, SA-WMI-PA, and SAE4WMI.



(a) $H = 0.25$ (b) $H = 0.5$

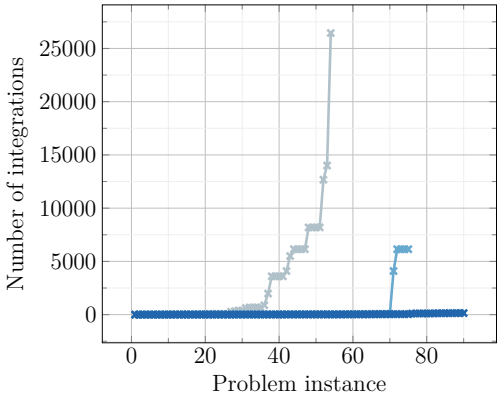
(c) $H = 0.75$ (d) $H = 1.0$

Figure 4.5: Cactus plots representing average query execution times in seconds on a set of DET problems with $H \in \{0.25, 0.5, 0.75, 1\}$.

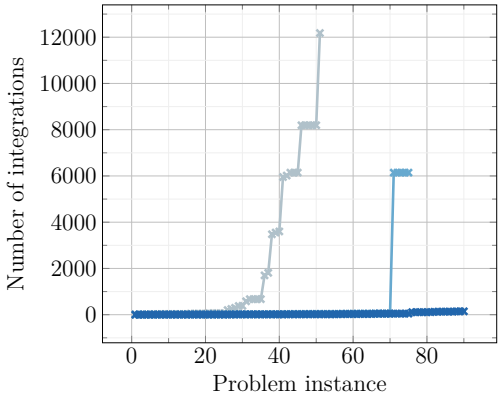


(a) $H = 0.25$

(b) $H = 0.5$



(c) $H = 0.75$



(d) $H = 1.0$

Figure 4.6: Cactus plots representing the number of computed integrals on a set of DET problems with $H \in \{0.25, 0.5, 0.75, 1\}$.

Density Estimation Trees benchmarks. (Figures 4.5 and 4.6) Figure 4.5 depicts the runtime of the algorithms for $H \in \{0.25, 0.5, 0.75, 1\}$. Specifically, for each WMI domain, we show the *average* time for the 5 queries. Since internal nodes of w do not have arithmetic operations, DETs are better suited to KC-based solvers than the synthetic benchmarks are. However, while in the simplest inference cases ($H \leq 0.5$) substantial factorization is possible, when the coupling between variables increases this advantage is outweighed by the combinatorial reasoning capabilities of SMT-based approaches.

The effectiveness of SAE4WMI w.r.t. both SA-WMI-PA and WMI-PA is particularly evident on these problems. Analyzing the number of integrals computed (Figure 4.6), we notice a drastic reduction for SAE4WMI w.r.t. other SMT-based solvers. This is expected: the number of Boolean atoms is typically very high, so that we are able to drastically reduce the number of partial assignments over the Boolean component w.r.t. SA-WMI-PA thanks to the revised enumeration strategy. Crucially, in contrast with the recent SA-WMI-PA, the novel algorithm is able to compete or even outperform KC-based solvers even when the oblique queries involve a low number of variables.

4.4 Extending SAE4WMI with Different Integration Backends

A key feature of SMT-based solvers like WMI-PA, SA-WMI-PA and SAE4WMI is that they are *agnostic to the integration backend*. We implemented a modular interface, allowing the use of different integration backends, including approximate ones. This has multiple advantages. First, one can use specialized integrators that target specific weight families, such as piecewise-constant functions, or leverage structural properties such as factorizability. Second, approximate integrators can be used on high-dimensional problems where exact integration of $\mathsf{FIUC}^{\mathcal{LRA}}$ functions is infeasible. Third, it broadens the application of WMI beyond $\mathsf{FIUC}^{\mathcal{LRA}}$ functions to any function whose integral over convex polytopes can be approximated, including multivariate Gaussian distributions and black-box functions.

We explored the practical benefits of this approach with a variant of SAE4WMI using Monte Carlo (MC) approximate integration [167]:

$$\int_{\mu^{\mathcal{LRA}}} w(\mathbf{x})\ \mathrm{d}\mathbf{x} \approx \overbrace{\int_{\mu^{\mathcal{LRA}}} 1\ \mathrm{d}\mathbf{x}}^{\mathsf{Vol}(\mu^{\mathcal{LRA}})} \cdot \mathbb{E}_{\mathbf{x}\sim\mathcal{U}(\mu^{\mathcal{LRA}})}[w(\mathbf{x})] \tag{4.7}$$

In practice, we use the VolEsti [39] library for approximating $\mathsf{Vol}(\mu^{\mathcal{LRA}})$ and for sampling w. In contrast to fully approximate WMI algorithms [13, 1], in our case the combinatorial reasoning is exact. This feature can be crucial in contexts where it is acceptable to compute approximate probabilities but the hard constraints cannot be violated. Considering the difficulties of sampling in combinatorial spaces, this approach is also preferable when discrete reasoning is feasible but the integration is not.

Results on the synthetic benchmarks. We evaluated the resulting solver, named SAE4WMI(VolEsti), on the synthetic benchmarks. We compare it against SAE4WMI using exact integration as in Section 4.3.3, here referred to as SAE4WMI(LattE). Figure 4.7 shows the runtime and the approximation error of SAE4WMI(VolEsti). By increasing the number of samples N used to estimate $\mathbb{E}_{\mathbf{x}\sim\mathcal{U}(\mu^{\mathcal{LRA}})}[w(x)]$, we can trade the approximation error against execution time. From Figure 4.7 (top-left) we notice that SAE4WMI(VolEsti) with $N \in \{10^2, 10^3, 10^4, 10^5\}$ is drastically faster than SAE4WMI(LattE). By zooming into these plots, in Figure 4.7 (bottom-left), we notice that the execution time of SAE4WMI(VolEsti) grows very slowly with N. In fact, sampling in 3 dimensions is very cheap, so that the times required for SMT enumeration and for computing the integration volumes dominate the overall execution time. In Figure 4.7 (right) we notice that the relative error decreases significantly as we increase N, allowing us to obtain very accurate results with a very limited computational cost.

We additionally equipped SAE4WMI with an exact *symbolic* integrator based on an XADD. Whereas symbolic integration was shown to be preferable combined with KC-based algorithms (Figure 3 in [120]), numerical integration performs substantially better with our approach (Figure 4.7, top-left).

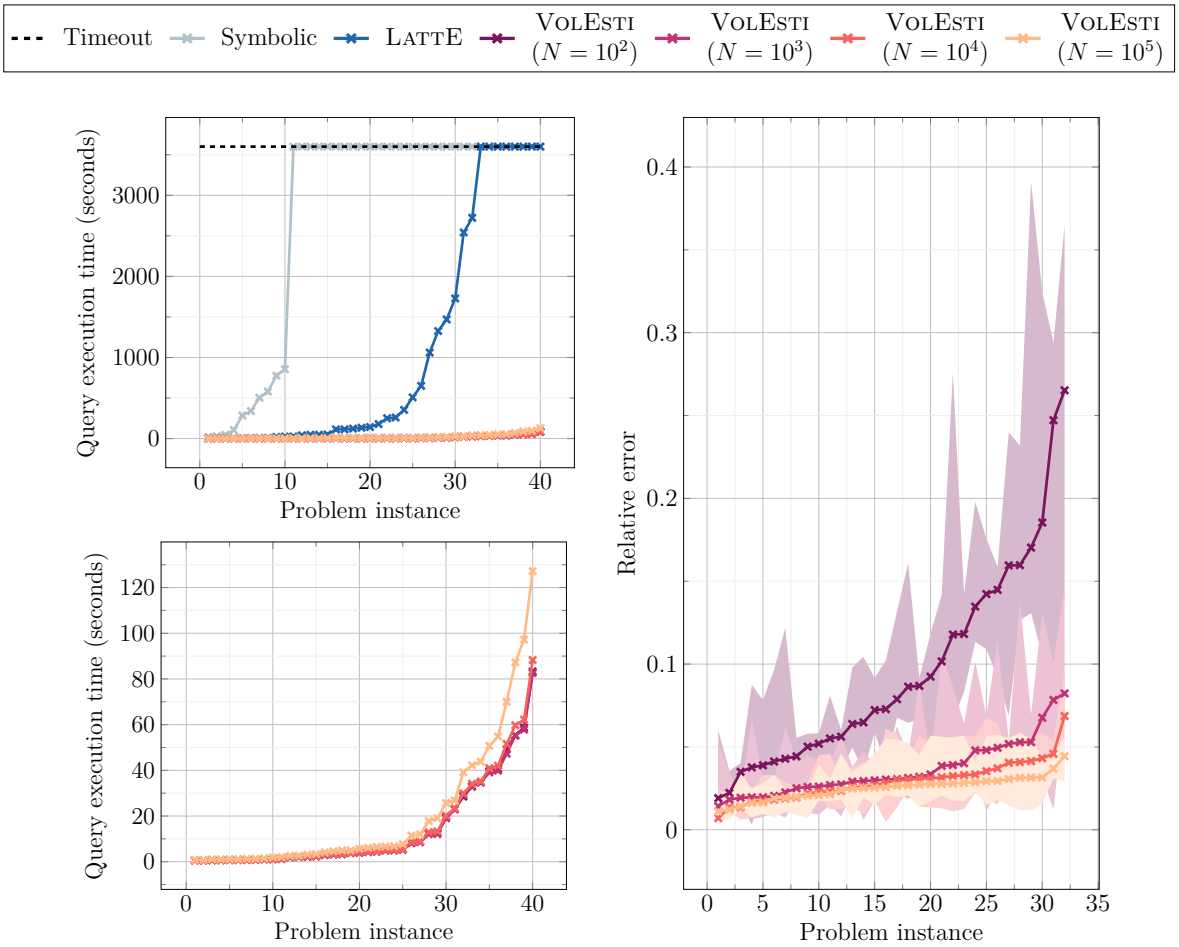


Figure 4.7: Cactus plots comparing SAE4WMI using different integration backends on the synthetic benchmarks. For SAE4WMI(VolEsti), we plot the median and color the range between the first and third quartiles of the outcomes obtained over 10 runs with different random seeds. Top-left: execution time of SAE4WMI with different integration backends. Bottom-left: zoom on the previous plot, showing only the results for SAE4WMI(VolEsti). Right: relative error of SAE4WMI(VolEsti) w.r.t. SAE4WMI(LattE) (where the latter does not time out).

Probabilistic Program Verification. We additionally showcase a prototypical application with Gaussian weights that are now supported by the approximate integration backend, borrowing the setting and examples from FairSquare [2]. In their setting, a (deterministic) *decision program* dec has to decide whether to hire a candidate or not, according to some features. The input of dec is distributed according to a Gaussian probabilistic program, the *population model* pop, which additionally models sensitive features that are not accessed by dec, e.g., whether an individual belongs to a minority group. The goal is to quantify the demographic parity of dec under the distribution pop:

$$\frac{\Pr(\mathsf{hire} \mid \mathsf{minority})}{\Pr(\mathsf{hire} \mid \neg\mathsf{minority})} \tag{4.8}$$

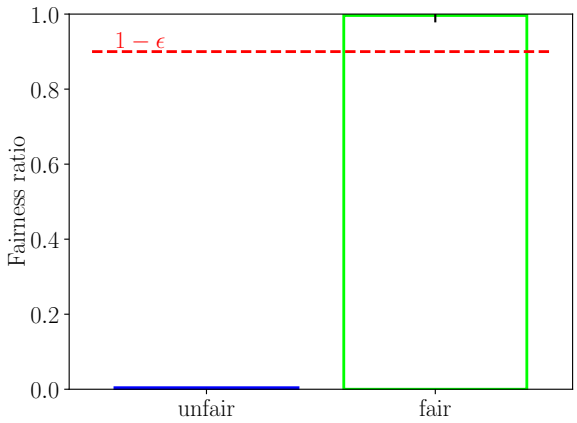


Figure 4.8: The ratio in (4.8) computed for the initial (unfair) dec and the modified (fair) one. Average and standard deviation over 5 runs are reported, confirming the results reported in the FairSquare paper with high confidence.

We report our results on the same introductory examples used in the FairSquare paper, whose detailed description is reported in Appendix B.2.

In contrast to our approach, FairSquare only certifies whether the ratio in (4.8) is greater than a user-defined threshold $1 - \varepsilon$. With SAE4WMI (VolEsti), we could indeed observe that the initial dec does not satisfy the fairness criterion with $\varepsilon = 0.1$. This problem was addressed by making dec less sensitive to a feature that was directly influenced by the sensitive condition. This second version of the program is fair with high probability, as reported in Figure 4.8.

Current limitations. A current limitation of this approach is that VolEsti does not directly offer statistical guarantees on the relative error. In problems like volume approximation [38] and sampling [37], practical statistical bounds on the error were obtained using diagnostics like the empirical sample size (ESS) or the potential scale reduction factor (PSRF) [91]. The adoption of similar mechanisms in our MC integrator is a fundamental step toward the application of WMI in formal verification.

4.5 Conclusions

Starting from the identification of the limitations of existing WMI solvers, in this chapter we have proposed a novel algorithm, SAE4WMI, that combines SMT enumeration with weight-structure awareness, thus unleashing the full potential of SMT-based technology.

An extensive experimental evaluation shows the advantage of SAE4WMI over current state-of-the-art WMI solvers on both synthetic and real-world problems. These include inference tasks over constrained density functions learned from data, highlighting the potential of the approach as a flexible tool for probabilistic inference in hybrid domains.

Whereas the improvements described in this chapter drastically reduce the number of integrals required to perform WMI, the integration itself can be a bottleneck for

scalability. To deal with this issue, we have shown how SMT-based approaches can seamlessly incorporate multiple integration backends, including symbolic, numerical, and approximate ones, allowing one to choose the most appropriate strategy for the problem at hand. In particular, using approximate MC integration techniques could drastically improve performance at the cost of a limited loss of precision.

We have further showcased the potential utility of our approach on a prototypical application aimed at verifying the fairness of probabilistic programs, as a first step toward the application of WMI technology to the probabilistic verification of hybrid systems and machine learning models.

Chapter 5

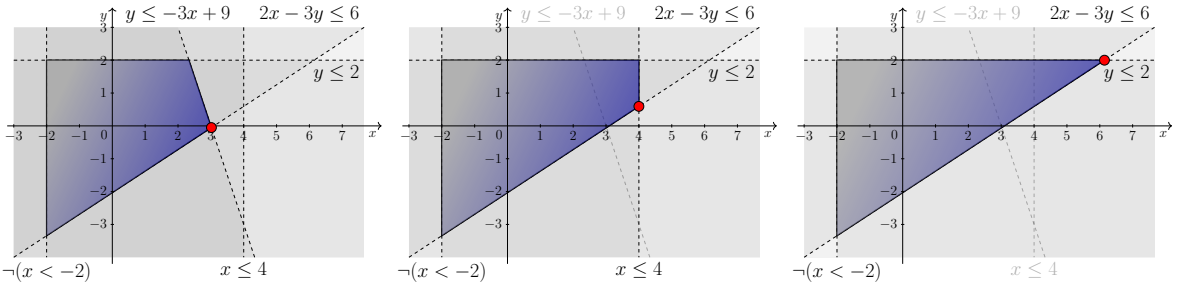
SMT-Enumeration Techniques for Optimization Modulo Theories

> **Full Disclosure.** This chapter is based on the paper *"Exploiting Partial-Assignment Enumeration in Optimization Modulo Theories"* [148], co-authored by the Ph.D. candidate with Prof. Roberto Sebastiani. The candidate is the main contributor. His contributions include the conceptualization, the core implementation, and the execution of the experimental evaluation.

We now turn our attention to an SMT-enumeration-related problem, i.e., Optimization Modulo Theories (OMT). In lazy OMT solving (Section 1.5.2), an SMT solver *enumerates $\mathcal{T}$-satisfiable truth assignments* that propositionally satisfy a given formula, adding increasingly tighter bounds on the cost function until an optimum solution is found.

Contributions. In this chapter, we study the applicability of enumeration-based techniques to OMT solving, focusing on partial-truth-assignment reductions to improve the effectiveness and efficiency of the search. First, we note that the $\mathcal{T}$-optimizer is typically invoked on total truth assignments, and we spotlight how this can be suboptimal. Second, we propose ways to exploit partial truth assignments in OMT solving, tailoring existing techniques to the OMT setting. We discuss the general idea for an arbitrary theory, and describe an implementation for the specific case of $\mathcal{LRA}$ and $\mathcal{LIRA}$, possibly combined with other theories. We implemented these strategies in the OMT solver OptiMathSAT [198]. An empirical evaluation over OMT($\mathcal{LRA}$), OMT($\mathcal{LIRA}$), and OMT($\mathcal{LRA} \cup \mathcal{AR}$) benchmarks shows that they improve both the efficiency of optimal solving and the quality of obtained solutions for anytime solving.

Organization. The chapter is organized as follows. In Section 5.1, we analyze the role of total and partial truth assignments in OMT solving. In Section 5.2, we propose two strategies to exploit partial truth assignments for OMT, and experimentally evaluate them in Section 5.3. In Section 5.4, we draw conclusions and discuss future work.



(a) Total assignment μ (5.1). (b) Partial assignment μ' (5.2). (c) Partial assignment μ'' (5.3).

Figure 5.1: Graphical representation of Example 5.1.1. For each step, the half-planes representing the literals in the truth assignment are delimited by dashed lines and colored in gray. The intersection of these constraints is colored in blue, with a gradient that follows the value of cost (the lower the value of cost, the darker the color), and the red dot represents the optimum model found within this region.

5.1 An Analysis of Enumeration in OMT

In the following, to simplify the notation and presentation, we refer to a single theory $\mathcal{T}$, but the results can be straightforwardly extended to combinations of theories. Moreover, we refer to the minimization problem, but the results can be easily adapted to maximization problems.

As described in Section 1.5.2, a basic OMT-solving schema involves the interaction of a combinatorial component and a theory-specific optimization component. In the combinatorial component, an SMT solver enumerates $\mathcal{T}$-satisfiable truth assignments that propositionally satisfy the problem formula φ conjoined with increasingly tighter bounds on the cost of the optimum solution. In the theory-specific component, a $\mathcal{T}$-minimizer finds a $\mathcal{T}$-model of minimum cost within the constraints imposed by the given truth assignment. This model is then used to tighten the upper bound for the cost of the optimum model and continue the search, until the formula is found unsatisfiable.

Since the enumeration is based on the CDCL($\mathcal{T}$) schema [143], these truth assignments are typically *total*, i.e., they assign a truth value to each atom of the formula. However, we point out that total truth assignments can often over-constrain the search space for the optimum model, whereas relying on *partial* truth assignments can be much more effective. Intuitively, *by removing from the current satisfying truth assignment $\mathcal{T}$-literals that are not strictly necessary for the propositional satisfaction of the formula, we enlarge the area within which the optimum model is searched, thus increasing the chances of finding a better optimum model.* This means that the solver can add a tighter upper bound to the cost of the global optimum, potentially reducing the number of search iterations needed to find it, and consequently the overall solving time. Moreover, this improvement can be crucial for anytime OMT solving, as it allows the solver to converge faster to better solutions within the given time limit.

We illustrate this idea in the following example.

Example 5.1.1. Consider the OMT($\mathcal{LRA}$) problem where $\mathsf{cost} \stackrel{\text{def}}{=} -2x$ and φ is the formula (1.1) in Example 1.2.1, which we recall here for the reader's convenience:

$$\varphi \stackrel{\text{def}}{=} \Big((2x - 3y \leq 6) \vee (x \leq 4) \Big) \wedge \Big((y \leq 2) \vee (y \leq -3x + 9) \vee (x < -2) \Big). \quad (1.1)$$

Consider the following scenario, which is graphically represented in Figure 5.1. Consider the $\mathcal{LRA}$-satisfiable total truth assignment that propositionally satisfies φ:

$$\mu \stackrel{\text{def}}{=} \{(2x - 3y \leq 6), (y \leq 2), \neg(x < -2), (y \leq -3x + 9), (x \leq 4)\}. \quad (5.1)$$

The optimum model of μ is $\{x \mapsto 3, y \mapsto 0\}$ with $\mathsf{cost} = -6$ (Figure 5.1a). We notice that some literals in μ are not strictly necessary for propositionally satisfying φ. In fact, we only need one true literal in each clause to propositionally satisfy the formula. Not every drop is equally effective, though. For instance, if we drop $(x \leq 4)$, $(y \leq 2)$, or $\neg(x < -2)$, then we get a truth assignment with the same optimum as μ, since these literals do not "oppose" the optimization of cost in μ. Dropping at least one of the other two literals, instead, leads to a truth assignment with a better optimum model. For instance, if we drop $(y \leq -3x + 9)$, we get:

$$\mu' \stackrel{\text{def}}{=} \mu \setminus \{(y \leq -3x + 9)\} = \{(2x - 3y \leq 6), (y \leq 2), \neg(x < -2), (x \leq 4)\}. \quad (5.2)$$

The optimum model of μ' is $\{x \mapsto 4, y \mapsto 2/3\}$ with $\mathsf{cost} = -8$ (Figure 5.1b). Now, two literals, $(x \leq 4)$ and $(2x - 3y \leq 6)$, oppose the optimization of cost, and either of them can be safely dropped. Assume that we drop $(x \leq 4)$, then we get:

$$\mu'' \stackrel{\text{def}}{=} \mu' \setminus \{(x \leq 4)\} = \{(2x - 3y \leq 6), (y \leq 2), \neg(x < -2)\}. \quad (5.3)$$

with optimum model $\{x \mapsto 6, y \mapsto 2\}$ and $\mathsf{cost} = -12$ (Figure 5.1c). Now, the only literals opposing the optimization of cost are $(2x - 3y \leq 6)$ and $(y \leq 2)$, but neither can be dropped. Hence, no further improvement can be obtained.

In general, partial truth assignments have an optimum model that is necessarily better than or equal to that of the total truth assignments extending them. Since multiple partial truth assignments can be obtained from a total one, the choice of which literals to drop can be crucial to improve the quality of the optimum model found.

5.2 Exploiting Partial Truth Assignments in OMT

Algorithm 7 shows the general schema of our approach. The algorithm is a variant of the OMT linear-search schema [195, 196] (Section 1.5.2). Its main difference is the call to OmtMinimizeAssignment (line 7), which reduces the truth assignment fed to the $\mathcal{T}$-minimizer, while preserving propositional satisfaction. Depending on its implementation, this reduction can be more or less effective in improving the search for the global optimum. We describe two possible implementations in Sections 5.2.1 and 5.2.2.

Algorithm 7 LinearSearchOmtWithPartialAssignments(φ, cost)
Input: φ: $\mathcal{T}$-formula; cost: objective $\mathcal{T}$-term
Output: SAT/UNSAT, optimum model $\mathcal{M}$

```
 1: M ← ∅                                              {Best model found so far}
 2: ub ← ∞                                             {Current upper bound}
 3: res ← SAT                                          {Status of the search}
 4: while res = SAT do
 5:    ⟨res, η⟩ ← SMT.IncrementalSolve(φ ∧ (cost < ub))
 6:    if res = SAT then
 7:       μ ← OmtMinimizeAssignment(φ, η, cost)
 8:       M ← T-Solver.MinimizeModel(μ, cost)
 9:       ub ← M(cost)
10:    end if
11: end while
12: if M = ∅ then
13:    return ⟨UNSAT, ∅⟩
14: else
15:    return ⟨SAT, M⟩
16: end if
```

5.2.1 Basic Assignment Reduction

The first approach is to reduce the truth assignment using Algorithm 2 in Section 1.3.2, i.e., iterating over all the literals in the current truth assignment η, and dropping them one by one, if possible. A straightforward improvement is to try to drop only $\mathcal{T}$-literals, since they are the ones that, if dropped, can potentially enlarge the area within which the optimum $\mathcal{T}$-model is searched. In the case of theory combination, we drop only literals from the theory $\mathcal{T}$ of the "$\leq$" symbol w.r.t. which we minimize. Possibly, heuristics can be used to choose an appropriate dropping order; in our implementation, we used the default strategy that follows the appearance order of the atoms in the formula. This procedure is simple and general, and comes with a limited overhead, as each truth assignment is scanned only once to find the literals to drop, and the $\mathcal{T}$-minimizer is called only once for each candidate assignment.

This approach, however, might not be very effective in practice, as it "blindly" removes literals from the truth assignment without taking into account the properties of the OMT search strategy. In particular, the search space may be enlarged in a direction that does not improve the objective, bringing no benefit to the search. Accurately choosing which literals to drop is crucial because the removal of a literal may prevent the removal of other literals that could potentially be more relevant for the optimization.

5.2.2 Guided Assignment Reduction

We propose an ad hoc assignment-reduction technique for OMT solving, which is outlined in Algorithm 8. Suppose that, after the $\mathcal{T}$-minimizer has found a minimum model

Algorithm 8 OmtMinimizeAssignmentGuided(φ, η, cost)
Input: φ: $\mathcal{T}$-formula; η: $\mathcal{T}$-SAT total truth assignment s.t. $\eta \models \varphi$; cost: obj.$\mathcal{T}$-term
Output: μ: reduced truth assignment $\mu \subseteq \eta$ satisfying φ

```
1:  μ ← η
2:  M ← T-Solver.MinimizeModelApprox(μ, cost)
3:  ℓ ← T-Solver.ProposeLiteralToDrop()
4:  while ℓ ≠ ⊥ do
5:    if μ \ {ℓ} satisfies all clauses in φ then
6:      μ ← μ \ {ℓ}
7:      M ← T-Solver.MinimizeModelApprox(μ, cost)
8:    end if
9:    ℓ ← T-Solver.ProposeLiteralToDrop()
10: end while
11: return μ
```

within the current truth assignment μ (line 2), it also returns one or more literals that limit the current minimum (line 3). These literals are part of some (possibly minimal) $\mu' \subseteq \mu$ such that $\mu' \cup \{\mathsf{cost} < \mathcal{M}(\mathsf{cost})\}$ is $\mathcal{T}$-unsatisfiable. Intuitively, the removal of any literal $\ell \in \mu'$ is very likely to lead to a better optimum model, provided that $\mu \setminus \{\ell\}$ still propositionally satisfies φ (line 5).

We can then iteratively drop these literals and re-run the $\mathcal{T}$-minimizer, until no more literals can be dropped (lines 4–9). Notice that we call $\mathcal{T}$-Solver.MinimizeModelApprox instead of $\mathcal{T}$-Solver.MinimizeModel, suggesting that a relaxed optimization algorithm may also be used. The next paragraph describes how to exploit this for $\mathcal{LIRA}$.

In Algorithm 8, we only drop one literal before every optimization call; nevertheless, in principle, more literals could be dropped. Experimentally, we found that dropping more than one literal was not beneficial. The reason is that, generally, when a literal is removed, most of the other literals in μ' do not limit the current minimum any longer. Hence, their removal not only does not lead to a better optimum model, but also can prevent the removal of other more relevant literals.

On proposing literals to drop. For an arbitrary theory $\mathcal{T}$, a generic implementation for $\mathcal{T}$-Solver.ProposeLiteralToDrop (Algorithm 8, lines 3 and 9) could be as follows. Once a minimum model $\mathcal{M}$ for μ is found, invoke the $\mathcal{T}$-solver on $\mu \wedge (\mathsf{cost} < \mathcal{M}(\mathsf{cost}))$ and return a (possibly-minimal) conflict set. For some theories, ad-hoc (possibly heuristic) procedures can be employed. We describe two such techniques for $\mathcal{LRA}$ and $\mathcal{LIRA}$.

As we recalled in Section 1.5, a $\mathcal{LRA}$-minimizer can be implemented as a variant of the Simplex method [75, 195], in which an optimum model is always found at a vertex of the polytope defined by the conjunction of $\mathcal{LRA}$-literals on which it is invoked. Thus, in this case, the candidate literals to be dropped are those that form such a vertex. This information can be easily obtained from the Simplex tableau [75].

For $\mathcal{LIRA}$, directly using a $\mathcal{LIRA}$-minimizer and then identifying the limiting literals presents some challenges:

(a) a single call to the branch-and-bound-based $\mathcal{LIRA}$-minimizer is exponential in the worst case; hence, calling it multiple times (as in Algorithm 8) is not feasible;

(b) extracting the limiting literals from a $\mathcal{LIRA}$-minimizer is not straightforward.

We thus propose proceeding as follows. First, in $\mathcal{T}$-Solver.MinimizeModelApprox an $\mathcal{LRA}$-minimizer is invoked on the relaxation of μ. Then, $\mathcal{T}$-Solver.ProposeLiteralToDrop can be implemented as for $\mathcal{LRA}$. The intuition here is that reasoning on the relaxation of μ is much easier and cheaper —especially if incremental calls are used— and still allows for enlarging the search area in a favorable direction for the $\mathcal{LIRA}$-minimizer. We remark that, after the assignment reduction, the complete $\mathcal{LIRA}$-minimizer is called in Algorithm 7, line 8.

Finally, we suggest that, if the $\mathcal{T}$-minimizer is able to find an optimum model $\mathcal{M}$, then we can use these limiting literals $\ell_1, \ldots, \ell_n$ also to learn a theory lemma $(\neg(\mathsf{cost} < \mathsf{ub}) \vee \neg\ell_1 \vee \ldots \vee \neg\ell_n)$ that blocks truth assignments that we know are not better than the current upper bound ub —thus preventing useless calls to the $\mathcal{T}$-solver. The idea is that, to find a model with $\mathsf{cost} < \mathsf{ub}$, we need to assign at least one of these literals to false. This is the case of $\mathcal{LRA}$, but not of $\mathcal{LIRA}$ if the truncated minimization method is used (see Section 1.5.2).

5.3 Experimental Evaluation

We implemented the above algorithms in the OMT solver OptiMathSAT [198], which is built on top of the MathSAT5 SMT solver [48]. We evaluated the proposed strategies on a set of OMT($\mathcal{LRA}$), OMT($\mathcal{LIRA}$), and OMT($\mathcal{LRA} \cup \mathcal{AR}$) benchmarks from different sources in terms of both the solving time for optimum solving and the quality of the solutions found within the given timeout for anytime solving.

5.3.1 Information for the Reproducibility of the Experiments

All experiments were run on a 28-core machine with an Intel Xeon Gold 6238R @ 2.20 GHz, 128 GB of RAM, and Ubuntu Linux 22.04. The timeout was set to 1200 s. The tool, benchmarks, and results are publicly available.[1]

5.3.2 Benchmarks

We evaluated the proposed strategies on two classes of OMT($\mathcal{LRA}$) benchmarks: OMT-encoded optimal temporal planning [173, 174] and strip-packing problems [195, 196]. We also modified the strip-packing benchmarks to use OMT($\mathcal{LIRA}$) and OMT($\mathcal{LRA} \cup \mathcal{AR}$) encodings to evaluate the effectiveness of our strategies in these theories.

[1] https://optimathsat.disi.unitn.it/resources/optimathsat-partial-assignments.tar.gz

Optimal Temporal Planning. In [173, 174], the authors proposed a way to encode optimal temporal planning problems into a sequence of OMT($\mathcal{LRA}$) problems. Each problem encodes a bounded version of the problem up to a fixed horizon h, with additional abstract actions that represent an over-approximation of the plans beyond the bound. The objective is to minimize the makespan, i.e., the total time taken to reach the goal. If the optimum plan is found without using the abstract actions, then the plan is optimum for the original problem. Otherwise, h is increased, and the process is repeated. We generated problems using the TEMPEST tool,[2] considering the industrial domains Majsp (80 instances), MajspSimplified (80 instances), and Painter (30 instances) [174] with increasing $h \in \{5, 10, 15, 20, 25, 30, 35, 40\}$, for a total of 1520 instances.

Strip-packing. The strip-packing problem (SP) requires arranging N rectangles with widths W_i and heights H_i into a strip of fixed height H and unlimited length. The goal is to minimize the length of the used part of the strip, ensuring all rectangles are placed without overlap or rotation. An OMT($\mathcal{LRA}$) encoding for SP was proposed in [196]. Following [196], we sampled H_i uniformly in $(0, 1]$, W_i in $(1, 2]$, and set $H = \sqrt{N}/2$.

We also generated OMT($\mathcal{LIRA}$) SP problems by randomly choosing, with equal probability, whether to encode the coordinates of each rectangle with integer or rational variables. Finally, we generated OMT($\mathcal{LRA} \cup \mathcal{AR}$) encodings for SP by replacing the variables x_i in the OMT($\mathcal{LRA}$) encoding with a $\mathcal{LRA} \cup \mathcal{AR}$ term $\mathit{read}(x, i + \mathit{offset})$, where x is an array mapping from rationals to rationals, i is a constant indicating the index of the rectangle, and *offset* is a fresh rational variable.

For each of these encodings, we generated 25 random SP problems for each value of $N \in \{25, 50, 75, 100\}$, for a total of 100 instances per encoding.

5.3.3 Results

Figures 5.2 to 5.5 show the results on temporal planning and SP benchmarks for the different theories. For each benchmark set, we report a set of scatter plots.

The rows show different metrics, namely the solving time in seconds (time (s)), the upper bound (u.b.) —i.e., the optimum value when the solver terminated within the time limit, or the value of the best solution found within the timeout otherwise— and the number of iterations (# iter) taken to reach the upper bound (see Algorithm 7).

The columns compare reduction strategies: the left and center columns compare basic and guided reductions against the plain algorithm with no reductions, while the right column compares the two reductions directly.

OMT($\mathcal{LRA}$) — Optimal temporal planning. (Figure 5.2) In these benchmarks, with no truth-assignment reduction, OPTIMATHSAT reported 246 timeouts out of 1520 problems, 211 with the basic reduction, and 212 with the guided reduction.

From the plots (first row, left and center columns), we can see that applying either reduction almost uniformly improves the solving time with few exceptions, making optimal solving up to twice as fast as with no reduction.

[2] `https://github.com/fbk-pso/tempest`

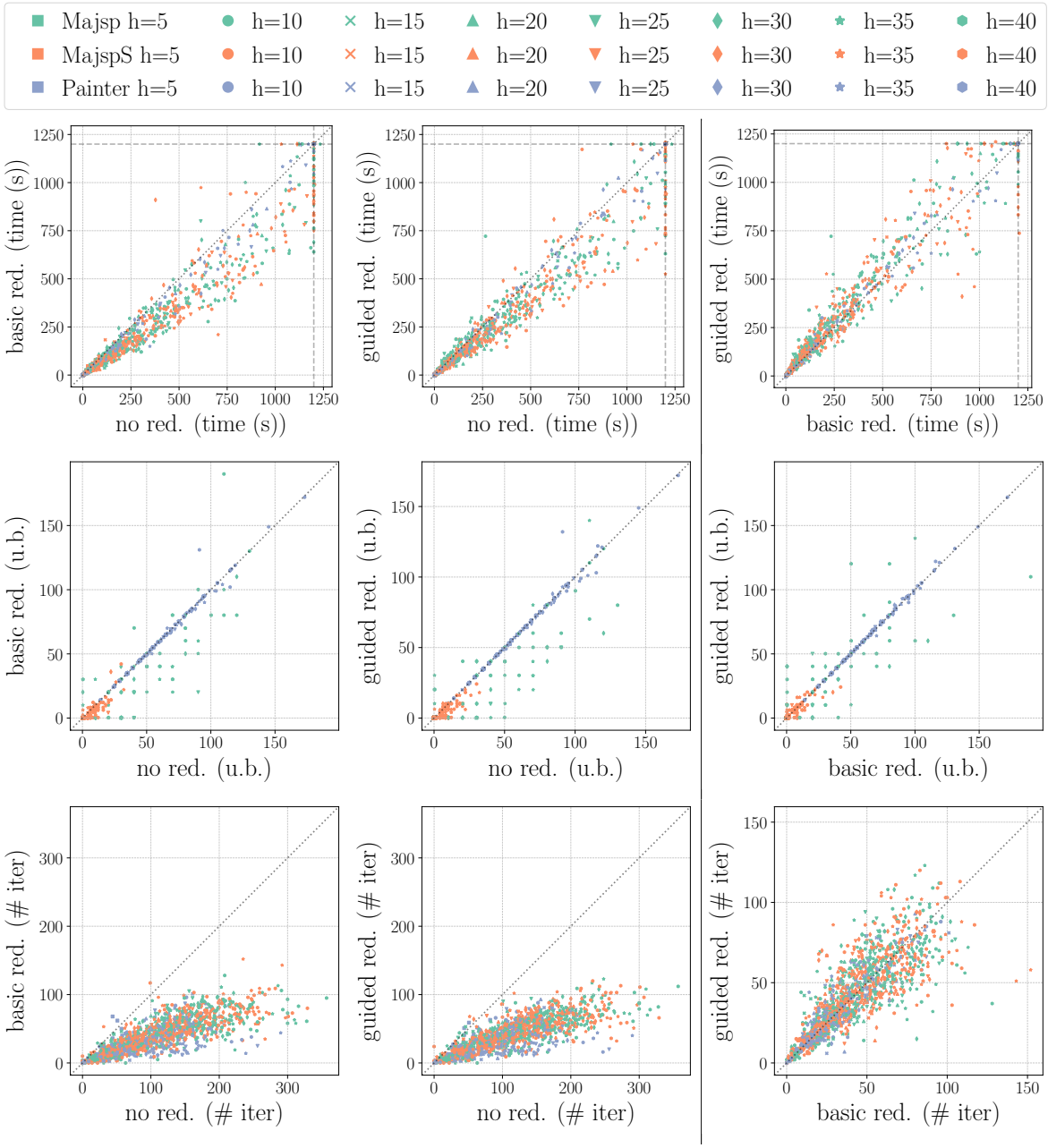


Figure 5.2: Results on OMT($\mathcal{LRA}$)-encoded optimal temporal planning problems.

Moreover, we observe that reducing truth assignments is also very effective for anytime solving (second row, left and center columns). Notice that when the solver terminated within the timeout with both strategies, the corresponding points lie on the bisector, whereas when at least one strategy times out, the points are generally below the bisector. Indeed, this shows that, for anytime solving, both the basic and the guided reductions allow the solver to find a much better upper bound than with no reduction.

Finally, we can see that both strategies are particularly effective in reducing the number of iterations needed to either find the optimum or to reach the best upper bound

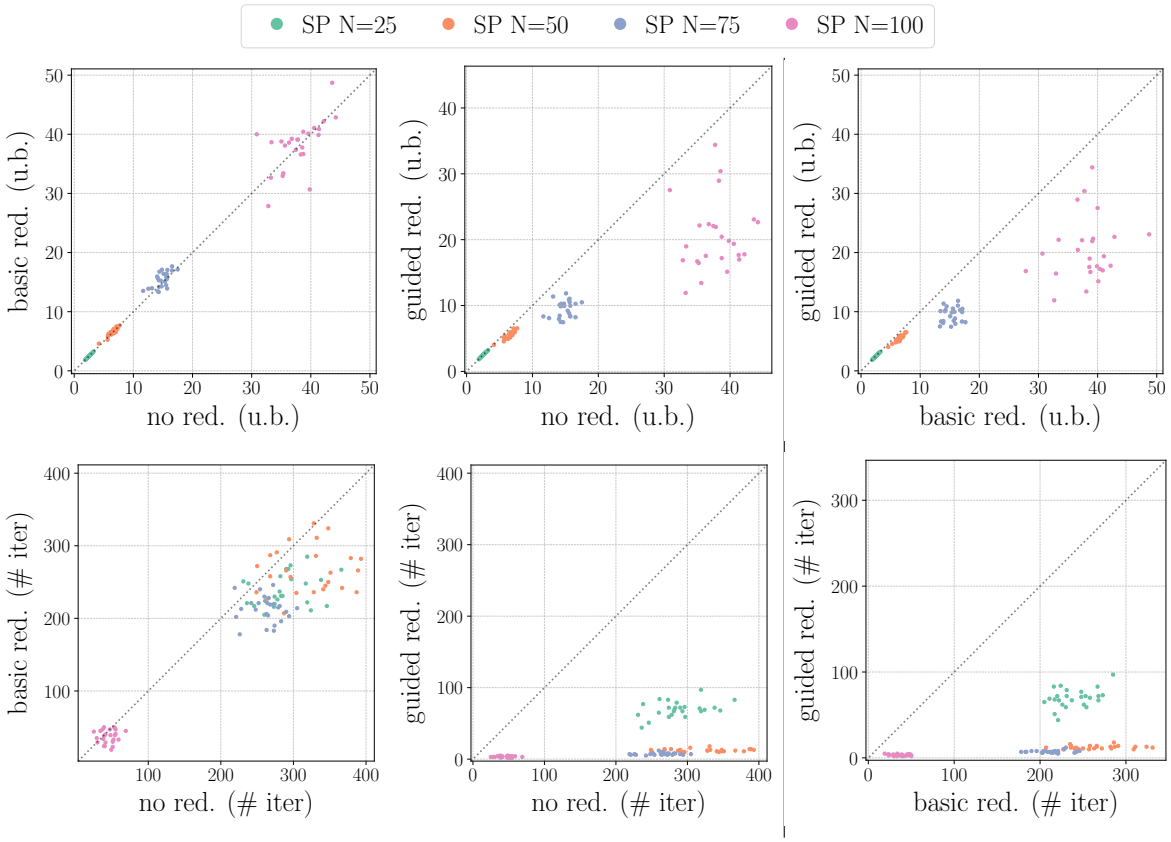


Figure 5.3: Results on OMT($\mathcal{LRA}$)-encoded strip-packing problems.

within the timeout (third row, left and center). Reducing the number of iterations is not an advantage in itself, but it is a good indicator of the effectiveness of the reduction strategies.

Overall, in these benchmarks there is no clear winner between the two reduction strategies (right column), but it is evident that applying either form of truth-assignment reduction can be beneficial in OMT, both for optimal and anytime solving.

OMT($\mathcal{LRA}$) — Strip-packing. (Figure 5.3) Since no instance in this benchmark set terminated within the timeout, we here omit the time plots.

Here, the basic reduction strategy is ineffective because the upper-bound value does not improve over the no-reduction strategy (first row, left column). Also, the number of iterations only slightly decreases (second row, left column), suggesting that here blindly removing atoms from the truth assignment does not help much in finding better solutions.

On the other hand, the guided reduction strategy is much more effective because it allows the solver to find a much better upper bound within the timeout (first row, center and right columns), and the number of iterations is drastically reduced (second row, center and right columns).

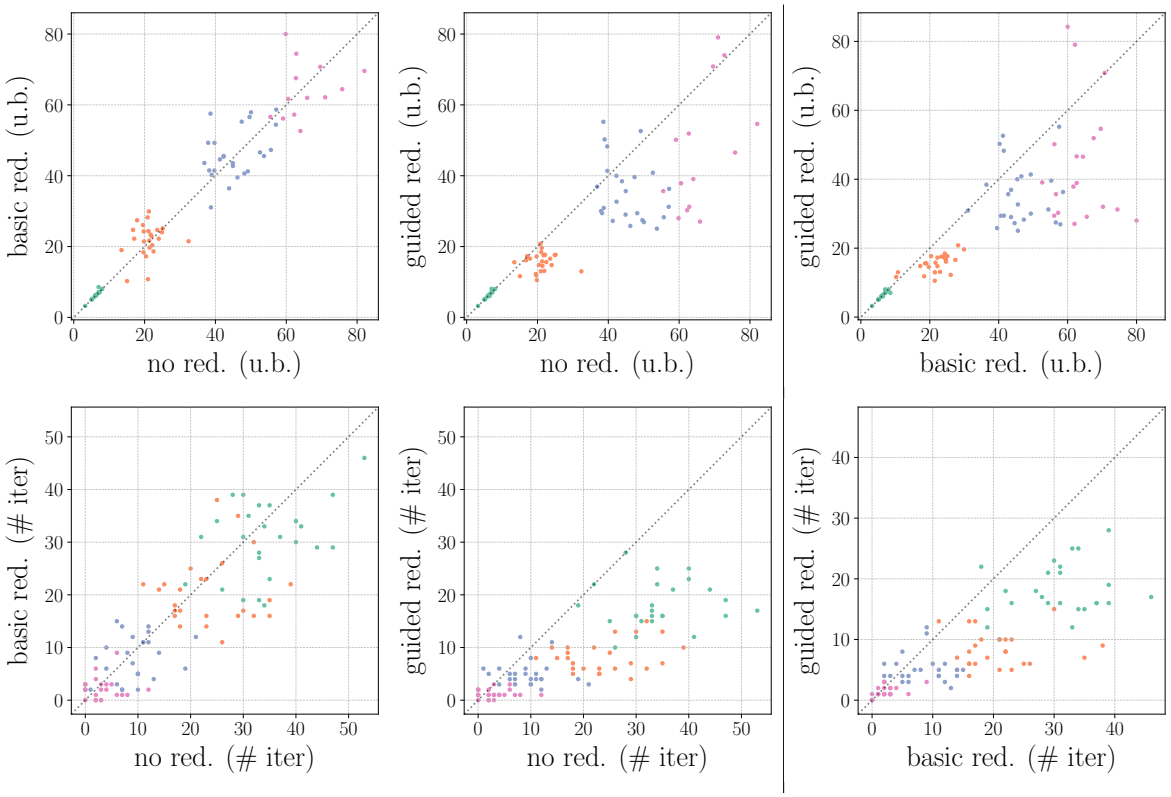


Figure 5.4: Results on OMT($\mathcal{LIRA}$)-encoded strip-packing problems.

OMT($\mathcal{LIRA}$) — Strip-packing. (Figure 5.4) For $\mathcal{LIRA}$, we observe a trend similar to that for $\mathcal{LRA}$. In fact, the basic reduction strategy is not very effective for improving the upper bound (first row, left column), and the number of iterations is only slightly reduced (second row, left column).

The guided reduction, instead, allows the solver to find a much better upper bound within the timeout (first row, center and right columns). Notice that, since these problems are much harder than the OMT($\mathcal{LRA}$) ones, the number of iterations completed within the timeout is much smaller, so the upper bounds found are also larger.

OMT($\mathcal{LRA} \cup \mathcal{AR}$) — Strip-packing. (Figure 5.5) Similarly, we observe that reduction also works for combinations of theories, such as $\mathcal{LRA} \cup \mathcal{AR}$. The results are similar to those for $\mathcal{LRA}$. Remarkably, here the advantage of the guided reduction is even more evident.

Discussion. The results show that applying either form of truth-assignment reduction can be beneficial in OMT, both for optimal and anytime solving, and that accurately selecting which atoms to remove from the truth assignment can make a significant difference in finding better solutions in fewer iterations. We observe, however, that a much smaller number of iterations, i.e. of truth assignments enumerated, does not always translate directly into a shorter solving time. This can be due to several reasons.

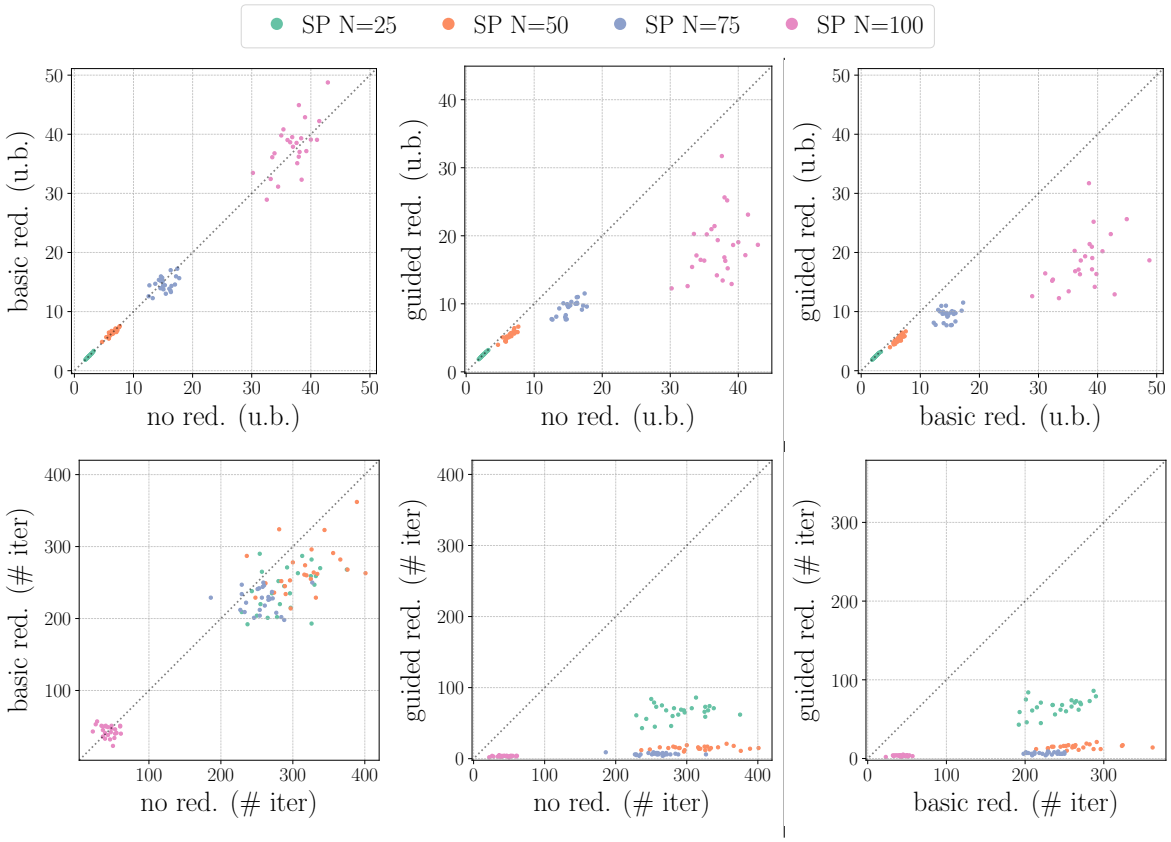


Figure 5.5: Results on OMT($\mathcal{LRA} \cup \mathcal{AR}$)-encoded strip-packing problems.

First, we remark that the global efficiency of OMT depends on several factors, including the order in which truth assignments are enumerated, and different literal selection heuristics may alter this order. Also, the removal of some literals from a truth assignment can prevent the removal of others in subsequent iterations. The effects of these factors are quite unpredictable.

Second, we observe that, in these problems, the number of truth assignments enumerated is typically limited to a few hundred. In fact, in OMT the bounds on the objective already enable very effective pruning of the search space. This also explains why the CNF encoding proposed in Chapter 3 does not seem beneficial for OMT. In fact, the proposed encoding can be up to twice as large in terms of clauses and fresh label atoms w.r.t. the standard Tseitin encoding performed by OptiMathSAT. This increases the cost of each iteration, and the reduction in the number of enumerated truth assignments is insufficient to compensate for it.

Moreover, in OMT, pruning is typically done by theory reasoning, and most of it has to be done anyway, regardless of the number of truth assignments enumerated. Performing this pruning in a single iteration or across many iterations may not have as much effect on the solving time because the efficient incrementality of SMT solvers can greatly reduce the cost of consecutive iterations.

5.4 Conclusions and Future Work

In this chapter, we have investigated the role of truth assignment enumeration in OMT solving and proposed ways to exploit partial truth assignments to improve the efficiency and effectiveness of the search. In particular, we have proposed a truth assignment reduction strategy that takes advantage of the properties of the optimization problem to accurately choose the atoms to remove from the truth assignment.

We have implemented the proposed strategies in the OptiMathSAT solver and evaluated them on a set of OMT($\mathcal{LRA}$), OMT($\mathcal{LIRA}$), and OMT($\mathcal{LRA} \cup \mathcal{AR}$) benchmarks. Our experimental results show that the proposed strategies can significantly improve the performance of the solver, uniformly reducing the overall solving time for optimal solving, and finding much better solutions for anytime solving for all the analyzed theories.

Other truth assignment reduction strategies, such as entailment-based methods [82], have shown significant benefits for SAT enumeration. Their extension to OMT problems could be a promising direction for future work.

Chapter 6

SMT Enumeration for MAP Inference with Non-Convex Constraints

> **Full Disclosure.** This chapter is based on the paper "The Theory and Practice of MAP Inference over Non-Convex Constraints" [123], co-authored by the Ph.D. candidate with Ph.D. student Leander Kurscheidt, Prof. Antonio Vergari, and Prof. Roberto Sebastiani. The main contributors of the paper are Leander Kurscheidt and the candidate. The candidate's contributions include the conceptualization, implementation, and evaluation of the PaMap algorithm, reported in detail in this chapter. The paper also includes another contribution, the MpMap algorithm (Section 6.2), which was mainly conceptualized, implemented, and evaluated by Leander Kurscheidt. Moreover, the data imputation experiments (Section 6.3.3) were mainly designed and performed by Leander Kurscheidt. We also report a brief description of these contributions for completeness, and refer the reader to the original paper [123] for a detailed discussion.

Continuing within the realm of SMT-enumeration techniques for optimization, in this chapter we focus on constrained Maximum-A-Posteriori inference under $\mathcal{LRA}$-constraints (MAP($\mathcal{LRA}$)). As defined in Section 1.6, MAP($\mathcal{LRA}$) requires maximizing a density over the feasible region of a $\mathcal{LRA}$-formula. Since such densities can be highly nonlinear and are often not available in explicit form, these problems are typically intractable for OMT solvers, even when equipped with $\mathcal{NRA}$-reasoning capabilities [198, 113] (see Section 2.5). This motivates the need for new algorithms that can compute MAP($\mathcal{LRA}$) efficiently and scale to real-world instances.

Contributions. First, since the general problem is intractable to solve exactly, we investigate how to approximate constrained MAP over $\mathcal{LRA}$-formulas and a wide class of nonlinear, non-log-concave distributions. To this end, we design PaMap, a general scheme that exploits SMT enumeration to decompose global optimization into a series of MAP inference problems over convex polytopes. These subproblems can be efficiently

solved by standard optimizers, and can provide approximation guarantees for (piecewise) polynomial densities [179, 130].

Second, we briefly summarize some theoretical advances tracing a non-trivial fragment of tractable MAP($\mathcal{LRA}$) problems. Specifically, when distributions are represented as tree-factorized piecewise (exponentiated) polynomials, optimization can be solved exactly via an efficient message-passing algorithm. This algorithm, named MpMap, is inspired by message-passing schemes for WMI [238, 237]. However, performing MAP inference poses different challenges and yields different complexity results.

Lastly, we evaluate PaMap over real-world benchmarks, demonstrating that our optimizer outperforms state-of-the-art OMT solvers [198, 113] and other soft-constrained methods, both in terms of approximation quality and runtime.

As such, we set a first milestone in tackling this challenging constrained optimization problem from both a theoretical and practical perspective.

Organization. The remainder of this chapter is structured as follows. In Section 6.1, we present the technical details of PaMap. Section 6.2 briefly introduces the MpMap message-passing algorithm for tree-structured problems. In Section 6.3 we experimentally evaluate PaMap. In Section 6.4 we summarize our findings and draw our conclusions.

6.1 PAMAP: Scaling Local Convex Optimizers via SMT Enumeration

Unlike Chapter 5, we focus here on the maximization problem. Optimizing a real-valued function under $\mathcal{LRA}$-constraints is typically formalized and solved as an OMT problem (see Section 1.5). As discussed in Section 2.5, however, this approach is currently viable only for a limited class of problems, where:

(a) the constraints are expressed as $\mathcal{LRA}$- or $\mathcal{NRA}$-formulas, and

(b) the density is a $\mathcal{LRA}$- or $\mathcal{NRA}$-term, i.e., a (piecewise) polynomial.

Moreover, if either the constraints or the density are in $\mathcal{NRA}$, OMT solvers employ either incomplete algorithms that are not guaranteed to terminate [18], or complete algorithms with double-exponential worst-case complexity in the number of variables [113], making them impractical for real-world problems.

We address these limitations by designing a general and practical algorithm for approximating MAP($\mathcal{LRA}$) that supports a wide class of densities beyond polynomials, including black-box densities that are not available in closed form. This is made possible by decoupling the enumeration of convex polytopes from the optimization step, which are instead tightly coupled in standard OMT solving (see Section 1.5.2). In fact, though this coupling is highly effective, e.g., for OMT($\mathcal{LRA}$), the introduction of terms in the form $(\mathsf{cost} > \mathsf{lb})$ in the OMT loop (see Section 1.5.2) makes it inevitable to reason on $\mathcal{NRA}$-formulas, even when the original formula is in $\mathcal{LRA}$ and only the objective is a polynomial. By contrast, our approach always reasons on $\mathcal{LRA}$-formulas.

6.1.1 The General Scheme

Algorithm 9 PaMap(φ, p)
Input: φ: $\mathcal{LRA}$-formula; p: density function
Output: lb: max. density; $\mathcal{M}$: the model that achieves it

```
 1: lb ← −∞                                          {current lower bound}
 2: M ← ∅                                            {current best point}
 3: cpenum ← CPEnum(φ, p)
 4: cpopt ← CPOpt()
 5: while cpenum.HasNext() do
 6:    μ ← cpenum.Next()
 7:    lb_μ, M_μ ← cpopt.Optimize(μ, p)
 8:    if lb_μ > lb then
 9:       lb, M ← lb_μ, M_μ
10:       cpenum.UpdateLowerBound(lb)
11:    end if
12: end while
13: return lb, M
```

The algorithm, named PaMap, is inspired in name and spirit by WMI-PA and its subsequent improvements for WMI computation [160, 161, 204, 205] (see also Section 1.4.2 and Chapter 4). It decomposes the solution space of a $\mathcal{LRA}$-formula into a partition of convex polytopes, over which constrained optimization can be performed efficiently, and sometimes with guarantees, for many density classes (see Section 2.5).

The procedure, outlined in Algorithm 9, builds on two key components: an enumerator cpenum that partitions the feasible region of φ into convex polytopes, and a constrained optimizer cpopt to maximize the density p over each convex polytope. PaMap maintains the current best solution lb (lines 1-2), and iteratively considers convex polytopes μ in the partition (lines 5-12). For each μ, it invokes cpopt to find the maximum of p restricted to μ (line 7), updating the best solution accordingly (lines 8-11). Importantly, unlike exact WMI computation, finding the maximum of p does not necessarily require enumerating *all* convex polytopes. Therefore, after updating the best solution, the enumerator is informed of the new lower bound lb (line 10), allowing it to *prune* polytopes that are known not to contain better solutions. As PaMap is a family of optimizers, we discuss next how different choices for the base optimizer cpopt and polytope enumeration cpenum can impact performance.

6.1.2 Optimization over Convex Polytopes

As discussed in Section 2.5, optimization over a single convex polytope is a well-understood problem, so that for cpopt we can leverage off-the-shelf constrained optimizers. Notice that many of them handle arbitrary functions with the only requirement of being able to evaluate them at given points, which allows us to handle densities that are not available in closed form, as we show in Section 6.3 for the case of VAEs.

Feasibility guarantees. Numerical optimizers rely on floating-point arithmetic, and as such the returned point may be slightly infeasible, i.e., it may not satisfy all the constraints of the polytope. Nevertheless, such points can be projected onto the polytope if needed. We remark that this is possible thanks to the fact that we handle constraints and optimization separately, as projection onto a *convex polytope* is a well-known problem [86].

Approximation guarantees. In general, since the combinatorial reasoning is performed exactly, PAMAP inherits any approximation guarantee from the constrained optimizer used on the enumerated polytopes. For local optimizers, a common requirement is a convex objective, which translates into a concave (log-)density. If the negative (log-)density is (log-)-strongly convex, we can bound the approximation error of the arg max of the density (see Section 9.1 of [28]). In contrast, some global optimizers, such as sum-of-squares (SoS) optimization for polynomial densities [130], can guarantee exact recovery under suitable conditions but typically exhibit exponential scaling in time or apply only to very restricted function classes. For general densities, approximation guarantees are rare, though in practice numerical optimizers can often find good solutions efficiently, as we show in Section 6.3.

6.1.3 Enumeration of Convex Polytopes

Partitioning the feasible region of an $\mathcal{LRA}$-formula into convex polytopes can be reduced to $\mathcal{T}$-AllSAT (see Section 1.3), as each enumerated truth assignment corresponds to a conjunction of linear (in)equalities, which defines a non-empty convex polytope. As such, we can leverage off-the-shelf SMT enumeration tools and techniques, such as those in [48, 207], to implement cpenum.

Most importantly, as observed also in Chapters 3, 4 and 5, enumerating *partial* truth assignments can lead to a significant reduction in the number of enumerated polytopes, thus improving the efficiency of PAMAP. We can exploit the techniques presented in Chapter 3 to drastically reduce the number of enumerated polytopes for the case of non-CNF formulas.

Moreover, if the density p is defined as a piecewise function, some constrained optimizers may not be able to handle it directly (e.g., [130]). In such cases, we can use the structure-aware enumeration procedure presented in Section 4.2 to ensure that each enumerated polytope lies entirely within a single piece of p.

Pruning via upper bounds. To prune polytopes that will not contribute to improving the maximum, we can sometimes compute an upper bound on p for a given polytope, or more in general, for a region of the search space, and if this bound is lower than the current best value lb, that region can be safely discarded.

Although tight upper bounds are nontrivial and often impossible to compute in general, we can sometimes exploit the specific shape of p to derive efficient bounding techniques. In Appendix C.3, we provide an example of how to compute such upper bounds for the case of PAL densities [124], which we use in our experiments.

6.1.4 Practical Limitations and Scalability

The primary practical limitation of PaMap is the potentially exponential growth in the number of convex polytopes w.r.t. the number of linear (in)equalities in the formula. As shown in Section 6.3, however, this worst-case behavior is often not observed in practice, especially when effective pruning strategies are employed. We also note that PaMap can consume polytopes as they are enumerated, without the need to store the entire set in memory. Hence, memory usage depends only on the implementations of the convex-polytope optimizer and the convex-polytope enumerator. Concerning the optimizer, several numerical optimizers (e.g., [24, 229, 55, 76]) run in polynomial space. Similarly, $\mathcal{T}$-AllSAT can be done by using polynomial space [207].

6.2 A Scalable Message-Passing Algorithm for Constrained MAP

To address the potential scalability issues of PaMap, we also identify a non-trivial fragment of MAP($\mathcal{LRA}$) problems that can be solved *exactly* and *tractably* via a message-passing scheme, which we call MpMap.

A tree-shaped MAP($\mathcal{LRA}$). This approach is inspired by the message-passing approach for WMI computation [238] (see also Section 2.3). As in the WMI case, we target problems that satisfy the following conditions:

(a) the $\mathcal{LRA}$-formula φ is in CNF and each clause contains at most two $\mathcal{LRA}$-variables, i.e., $\varphi \stackrel{\text{def}}{=} \bigwedge_{i,j} \varphi_{ij} \wedge \bigwedge_i \varphi_i$ with φ_{ij} [resp. φ_i] being the restriction of φ to the clauses over the variables x_i, x_j [resp. x_i];

(b) the probability density function p factorizes into a product of univariate and bivariate factors, i.e., $p(\mathbf{x}) \stackrel{\text{def}}{=} \prod_{i,j} p_{ij}(x_i, x_j) \prod_i p_i(x_i)$;

(c) the *global graph* obtained by connecting variables that appear in the same clause or factor is a tree.

Tractable MAP conditions. Beyond enforcing this tree-shaped structure, establishing exact tractability requires identifying a set of Tractable MAP Conditions (TMC) that a family of densities must satisfy to ensure tractability of MAP($\mathcal{LRA}$). The class of densities that allows tractable WMI is not suitable for tractable MAP($\mathcal{LRA}$). We refer the reader to [123] for a detailed discussion. For MAP inference to be tractable, the function family must satisfy the following conditions:

(i) it is closed under products;

(ii) it supports tractable pointwise maximum computation;

(iii) it allows for the tractable computation of a symbolic supremum under linear bounds.

Two prominent function families satisfying these requirements are Piecewise Polynomials ($\mathbf{\Omega}^{\mathsf{PP}}$) and Piecewise Exponentiated Polynomials ($\mathbf{\Omega}^{\mathsf{PEP}}$).

Indeed, if the global graph is tree-shaped and has a bounded diameter and the density satisfies the TMC, MAP($\mathcal{LRA}$) can be solved exactly in polynomial time.

The MpMap algorithm. The tractability of MAP($\mathcal{LRA}$) under the TMC is proven by construction via the MpMap algorithm, which we briefly summarize here. We refer the reader to [123] for a detailed description of the algorithm and its complexity, and for an experimental evaluation assessing its scalability on synthetic, high-dimensional benchmarks.

MpMap roots the tree-shaped global graph at an arbitrary variable, and passes messages from the leaves towards the root, processing one variable at a time. The tree structure guarantees that, once the value of a parent variable is fixed, the subproblems induced by its children are independent and can therefore be optimized separately.

The message from a child variable x_i to its parent x_j is a univariate function $m_{i\to j}(x_j)$. Intuitively, for each value of x_j, this function represents the optimal value obtainable over the variables in the subtree rooted at x_i, taking into account clauses and density factors involving those variables and x_j.

To compute such a message, MpMap combines univariate density factors and clauses involving x_i, bivariate ones involving x_i and x_j, and the messages already received from the children of x_i. It then eliminates x_i by solving this symbolic maximization problem, obtaining a function that depends only on x_j.

Once all messages have reached the root, MpMap maximizes the resulting univariate function to obtain the optimal MAP value. Similarly, we can compute the $\arg\max$.

The TMC ensure that messages remain tractable throughout the upward pass: after combining local components with incoming messages and maximizing over the current variable under the relevant clauses, the resulting message can still be represented in the same function family.

6.3 Experimental Evaluation

We now empirically assess how well PaMap trades off solution quality and runtime on real-world problems.

We have implemented PaMap in Python, building on existing libraries. For PaMap, we used MathSAT with the encoding presented in Chapter 3 for SMT enumeration; for numerical constrained optimization over convex polytopes, we used SciPy's optimization routines; for Lasserre's method, we implemented the moment hierarchy on top of the SumOfSquares.py package.[1] For PCAdam (see Section 6.3.1), we used PyTorch's Adam optimizer for parallel unconstrained optimization.

The code for reproducing the experiments is publicly available.[2]

[1] `https://github.com/yuanchenyang/SumOfSquares.py`

[2] `https://github.com/april-tools/constrained-map`

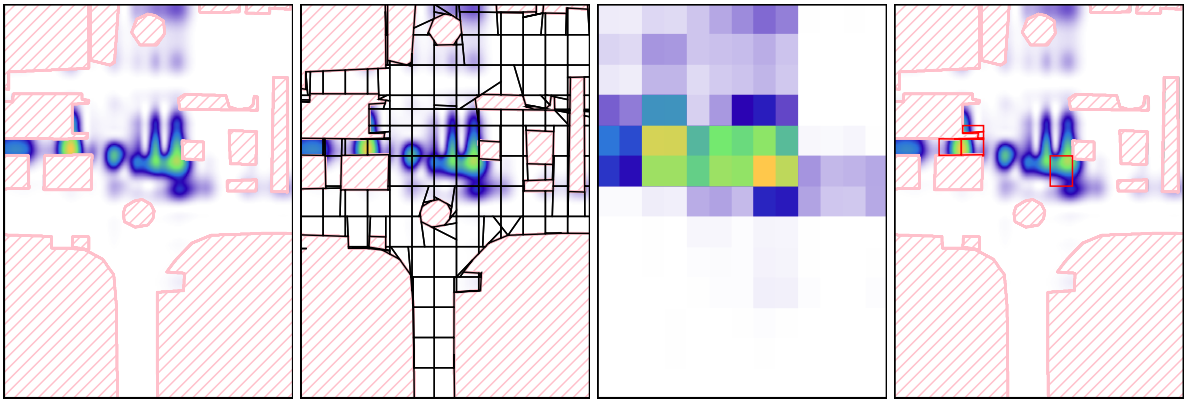


Figure 6.1: Left: constrained density for an agent's next position. Center-left: 257 convex polytopes considered by PaMap without upper-bound-based pruning. Center-right: piecewise-constant upper bound. Right: only 9 convex polytopes considered by PaMap with pruning. More examples are shown in Figure C.1 in Appendix C.3.

6.3.1 Baselines and Comparison

As a first baseline, we compare against the classical Adam optimizer [118], maximizing p without considering the constraints φ. This approach has two limitations: first, it may return infeasible solutions that do not satisfy φ; second, it is prone to getting stuck in local optima, especially in high-dimensional, non-convex landscapes.

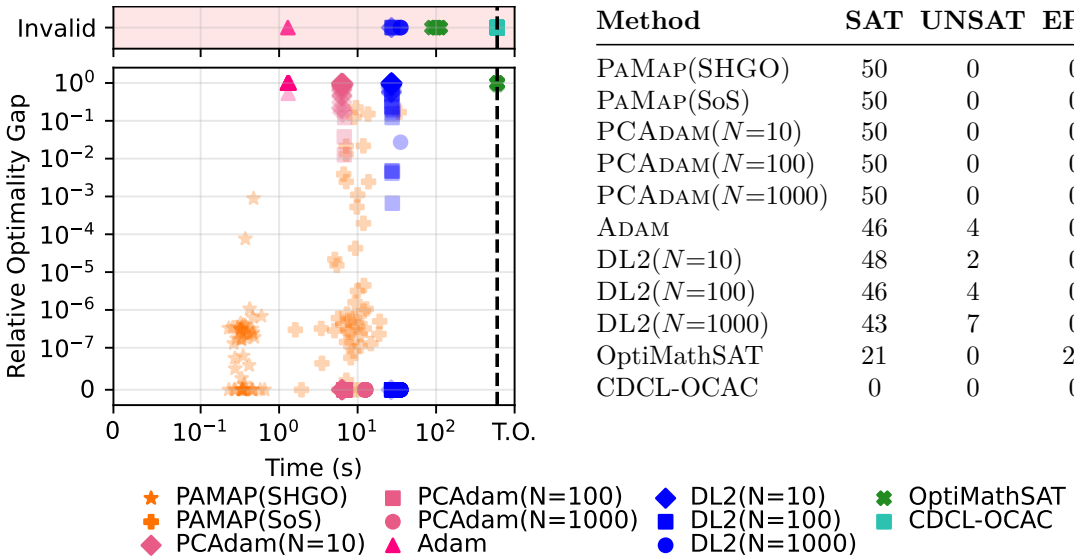
Hence, as a side contribution, we introduce a more competitive baseline PCAdam, a *particle-based, constraint-aware* version of Adam. PCAdam maintains a set of N particles (i.e., candidate solutions) that are iteratively updated using Adam. Crucially, at each iteration, we update the best result found so far among all particles that *satisfy* the constraints φ. Appendix C.1 provides further details.

We also compare against DL2 [80], where we optimize the soft-constrained objective using Adam with multiple random restarts.

Finally, when possible, we compare against OMT solvers for nonlinear real arithmetic, namely OptiMathSAT [198] and CDCL-OCAC [113].

6.3.2 Trajectory prediction on the Stanford Drone Dataset

Benchmark. As a first real-world application, we consider the Stanford Drone Dataset (SDD) [185], a collection of drone-captured trajectories of multiple agents. The agents can only move in walkable areas, modeled as $\mathcal{LRA}$-formulas over the 2D space [124]. Following [124], we learned a predictive density for an agent's future position, conditioned on its past trajectory and the scene layout. We then generated 50 test instances by sampling different agent trajectories. In Figure 6.1 (left) we show an example of such a density. The figure also shows an execution of PaMap on this instance, demonstrating how the computation of upper bounds is crucial for pruning the vast majority of the polytopes and thus improving efficiency. In Appendix C.3 we show further examples (Figure C.1) and provide details on upper-bound computation.



Method	SAT	UNSAT	ERR	T.O.
PaMap(SHGO)	50	0	0	0
PaMap(SoS)	50	0	0	0
PCAdam(N=10)	50	0	0	0
PCAdam(N=100)	50	0	0	0
PCAdam(N=1000)	50	0	0	0
Adam	46	4	0	0
DL2(N=10)	48	2	0	0
DL2(N=100)	46	4	0	0
DL2(N=1000)	43	7	0	0
OptiMathSAT	21	0	29	21
CDCL-OCAC	0	0	0	50

Figure 6.2: Left: Scatter plot of runtime vs. relative optimality gap. Right: Summary table reporting (SAT) instances solved with a point satisfying φ, (UNSAT) instances where the returned point does not satisfy φ, (ERR) instances terminated with an error, and (T.O.) number of timeouts (600s). Note that OptiMathSAT is anytime and returns the best solution found before timeout.

Setup. We run PaMap using two different optimizers: a numerical optimizer (SHGO) and an optimizer based on the SoS-Moment hierarchy (see Section 2.5). We compare it against Adam, PCAdam, and DL2 with different numbers of particles N, OptiMathSAT run in anytime mode, and CDCL-OCAC (see Appendix C.3).

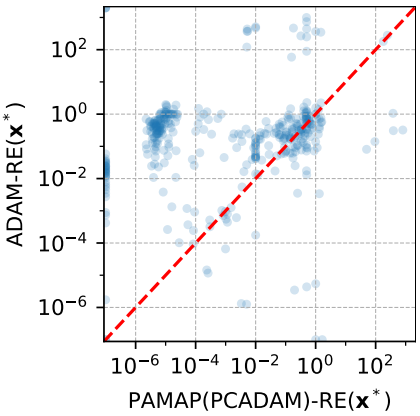
We compare the solvers' performance in terms of runtime and solution quality. We measure the latter by the relative optimality gap, defined as $\mathrm{OG}(\mathsf{lb}) \stackrel{\text{def}}{=} \max(0, v^{gt} - \mathsf{lb})/v^{gt}$, where lb is the value found and v^{gt} is the best known value, computed via a grid search.

Results. Results are shown in Figure 6.2. From the plot, we see that PaMap(SHGO) achieves the best trade-off between runtime and solution quality. PCAdam can be competitive in terms of solution quality, but only if enough particles are used, leading to significantly higher runtimes. PaMap(SoS), here used with a relaxation order of 7, is significantly slower than the SHGO variant. Adam and DL2 are not reliable at finding good, feasible solutions. OMT solvers struggle in this setting: CDCL-OCAC solved no instances, while OptiMathSAT found only poor-quality solutions or reported errors.

6.3.3 Data imputation of a VAE

Benchmark. As a second real-world application, we apply PaMap to data imputation with a density encoded in a VAE [119].

In this experiment, we use the *House-Sales* tabular dataset with the provided constraints from [210]. We train an unconstrained VAE on the training dataset and



Method	PaMap	Adam	Adam	DL2	DL2
Particles	10	10	100	10	100
$\mathrm{RE}(\mathbf{x}^{\mathcal{M}^\star})$	**0.22**	33.12	25.85	12.22	16.78

Figure 6.3: Left: Scatter plot comparing the relative errors of PaMap(PCAdam) and a particle-enhanced version of Adam. Right: Table comparing average relative error for PaMap(PCAdam), particle-enhanced Adam, and DL2.

mask random features in the test dataset. Constraints here are in the form of $\mathcal{LRA}$-formulas over features of a house, such as square meters, number of rooms, and cost.

To evaluate the benefit of constrained imputation, masking is restricted to variables for which constraints are available. For each of 400 test samples, every eligible feature is masked independently with probability 10%, and we compute a MAP estimate over the masked variables subject to the constraints. As we cannot compute the density induced by a VAE exactly, we approximate it via Monte Carlo samples from the latent prior $\mathbf{z}$:

$$p(\mathbf{x}_m \mid \mathbf{x}_o) \approx 1/N \sum_{i=1}^{N} p_{\mathsf{dec}}(\mathbf{x}_m, \mathbf{x}_o \mid \mathbf{z}_i), \quad \mathbf{z}_i \sim \mathcal{N}(0, I) \tag{6.1}$$

where $\mathbf{x} \stackrel{\text{def}}{=} \mathbf{x}_m \cup \mathbf{x}_o$ is a partition of the variables into observed and missing variables, p_{dec} is the decoder network, which outputs the parameters of an isotropic Gaussian distribution, and $\mathbf{z}_i$ is a sample from the latent prior distribution.

Setup. For PaMap, we need a convex-polytope optimizer that can handle a stochastic objective as in (6.1). Since the numerical optimizers we used in the trajectory experiment are not designed for this, we use our PCAdam as the convex-polytope optimizer within PaMap. We denote this configuration as PaMap(PCAdam).

We compare against a particle-based version of Adam with $N \in \{10, 100\}$. We also compare against DL2 [80]. Unfortunately, OMT solvers do not support these densities, preventing a direct comparison in this setting.

We measure the relative error via $\mathrm{RE}(\mathbf{x}^{\mathcal{M}}) \stackrel{\text{def}}{=} \mathrm{avg}(|\mathbf{x}^{\mathcal{M}} - \mathbf{x}^{gt}|/(|\mathbf{x}^{gt}| + 1))$, where $\mathbf{x}^{\mathcal{M}}$ is the vector of values assigned by the model $\mathcal{M}$ returned by the optimizer, and $\mathbf{x}^{gt}$ is the corresponding ground-truth vector.

Results. Results are shown in Figure 6.3. Despite the stochastic objective, PaMap substantially improves imputation accuracy in terms of relative error, even when the

competing methods use more particles.

This shows that MAP inference under $\mathcal{LRA}$-constraints is both practical and effective for real-world tabular data imputation. Further details, including timings, are provided in [123].

6.4 Conclusions

In this chapter, we traced the theoretical and practical foundations of MAP inference in the presence of non-convex $\mathcal{LRA}$-constraints and non-log-concave densities.

To this end, we introduced a new solver, PaMap, that substantially advances the state of the art for this challenging inference task. PaMap is a practical and modular framework combining SMT-based convex-polytope enumeration with constrained optimization over each such polytope.

In our experiments, we showed not only that our solver can achieve a better trade-off between solution quality and execution time, but also that it can be used to perform constrained inference in conjunction with black-box deep generative models.

Chapter 7

A Theory-Agnostic Approach to Theory-Lemma Enumeration in SMT

> **Full Disclosure.** This chapter is based partially on the papers "Canonical Decision Diagrams Modulo Theories" [152] and "Beyond Eager Encodings: A Theory-Agnostic Approach to Theory-Lemma Enumeration in SMT" [50], co-authored by the Ph.D. candidate with the Master's students Massimo Michelutti and Emanuele Civini, Dr. Giuseppe Spallitta, and Prof. Roberto Sebastiani.
> Massimo Michelutti implemented the baseline lemma-enumeration strategy [152].
> Emanuele Civini implemented and evaluated the strategies based on cube-and-conquer, projection, and partitioning [50].
> The Ph.D. candidate conceptualized the enumeration strategies with contributions from Prof. Roberto Sebastiani. Moreover, he supervised the implementation of [152, 50], and implemented and evaluated the latest cube-and-conquer lemma-enumeration strategy, which at the time of submission is still unpublished.

We now turn to a different application of SMT enumeration. This chapter studies the problem of enumerating a complete set of theory lemmas that rule out all $\mathcal{T}$-inconsistent truth assignments propositionally satisfying a given $\mathcal{T}$-formula. Devising efficient and theory-agnostic methods will serve as an enabling technology for the Knowledge Compilation (KC) Modulo Theories framework presented in Chapter 8.

Indeed, lifting Boolean-reasoning techniques to the SMT level most often requires producing $\mathcal{T}$-lemmas ruling out truth assignments that propositionally satisfy the formula but are $\mathcal{T}$-inconsistent. These lemmas prevent a Boolean-reasoning engine from (re)visiting $\mathcal{T}$-inconsistent regions of the search space.

Whereas for standard SMT and OMT solving these lemmas are generated lazily on the fly, in some applications, such as KC Modulo Theories, it is necessary to enumerate a complete set of $\mathcal{T}$-lemmas upfront. Besides eager encodings (e.g., [228, 34, 212, 211]), which are theory-specific and not tailored to a specific formula, in the literature there seems to be no theory-agnostic way to achieve this task (see Section 2.6).

Contributions. In this chapter, we address this problem explicitly, and present efficient and theory-agnostic methods for enumerating such a set of $\mathcal{T}$-lemmas. Starting from a $\mathcal{T}$-AllSAT-based baseline, we introduce three orthogonal improvements that drastically enhance scalability.

First, we propose a cube-and-conquer approach that partitions the Boolean search space into independent residual subproblems, which is massively parallelizable.

Second, we prove that projecting the enumeration on the $\mathcal{T}$-atoms still yields a complete set of $\mathcal{T}$-lemmas, reducing the dimension of the search.

Third, we show that when the $\mathcal{T}$-atoms of φ can be partitioned into symbol-disjoint components, $\mathcal{T}$-lemmas can be enumerated independently within each component. These improvements are theory agnostic and can be combined into a single framework.

An experimental evaluation on synthetic problems [152] and on temporal [224] and numeric planning problems [190] demonstrates that these techniques drastically enhance efficiency, enabling the approach to scale to substantially more complex instances.

Organization. The rest of this chapter is structured as follows. In Section 7.1 we introduce the basic definitions and notation used in this chapter. In Section 7.2 we present our theory-agnostic techniques for enumerating a complete set of $\mathcal{T}$-lemmas and evaluate them in Section 7.3. Finally, in Section 7.4 we draw our conclusions.

7.1 Basic Definitions

Definition 7.1.1 ($H_{\boldsymbol{\alpha}}(\varphi)$,$P_{\boldsymbol{\alpha}}(\varphi)$). Given a set $\boldsymbol{\alpha}$ of atoms and a $\mathcal{T}$-formula $\varphi[\boldsymbol{\alpha}]$, $H_{\boldsymbol{\alpha}}(\varphi) \stackrel{\text{def}}{=} \{\eta_i\}_i$ and $P_{\boldsymbol{\alpha}}(\varphi) \stackrel{\text{def}}{=} \{\rho_j\}_j$ denote, respectively, the set of all $\mathcal{T}$-satisfiable and the set of all $\mathcal{T}$-unsatisfiable total truth assignments on $\boldsymbol{\alpha}$ propositionally satisfying φ, s.t.:

$$\varphi \equiv_{\mathbb{B}} \bigvee_{\eta_i \in H_{\boldsymbol{\alpha}}(\varphi)} \eta_i \vee \bigvee_{\rho_j \in P_{\boldsymbol{\alpha}}(\varphi)} \rho_j. \tag{7.1}$$

Consequently, $H_{\boldsymbol{\alpha}}(\top)$ and $P_{\boldsymbol{\alpha}}(\top)$ are, respectively, the sets of all $\mathcal{T}$-satisfiable and $\mathcal{T}$-unsatisfiable total truth assignments that one can build on the atoms in $\boldsymbol{\alpha}$.[1]

In Appendix E.2, Proposition E.2.1 summarizes basic properties of $H_{\boldsymbol{\alpha}}(\varphi)$ and $P_{\boldsymbol{\alpha}}(\varphi)$.

Example 7.1.1 ($H_{\boldsymbol{\alpha}}(\varphi)$,$P_{\boldsymbol{\alpha}}(\varphi)$). Let $\boldsymbol{\alpha} = \{\alpha_1, \alpha_2\} \stackrel{\text{def}}{=} \{(x \leq 0), (x = 1)\}$, and let

$$\varphi_1 \stackrel{\text{def}}{=} (x \leq 0) \vee (x = 1), \quad \varphi_2 \stackrel{\text{def}}{=} \neg(x \leq 0) \leftrightarrow (x = 1).$$

We have that $\varphi_1 \not\equiv_{\mathbb{B}} \varphi_2$ and $\varphi_1 \equiv_{\mathcal{T}} \varphi_2$. Then:

$$H_{\boldsymbol{\alpha}}(\varphi_1) = H_{\boldsymbol{\alpha}}(\varphi_2) = \{\eta_1, \eta_2\} = \{(x \leq 0) \wedge \neg(x = 1), \neg(x \leq 0) \wedge (x = 1)\},$$
$$P_{\boldsymbol{\alpha}}(\varphi_1) = \{(x \leq 0) \wedge (x = 1)\}, \quad P_{\boldsymbol{\alpha}}(\varphi_2) = \emptyset.$$

[1] Notice that "H" and "P" are the capitalized versions of the Greek letters η and ρ respectively.

Definition 7.1.2 (Ruling-out). We say that a set $\{C_1, \ldots, C_K\}$ of clauses *rules out* a set $\{\rho_1, \ldots, \rho_M\}$ of $\mathcal{T}$-unsatisfiable total truth assignments on $\boldsymbol{\alpha}$ if and only if, for every ρ_j there exists a C_l such that $\rho_j \models_{\mathbb{B}} \neg C_l$, that is, if and only if

$$\bigvee_{j=1}^{M} \rho_j \wedge \bigwedge_{l=1}^{K} C_l \equiv_{\mathbb{B}} \bot.$$

Given $\boldsymbol{\alpha}$ and some $\mathcal{T}$-formula $\varphi[\boldsymbol{\alpha}]$, we denote by $Cl_{\boldsymbol{\alpha}}(\varphi)$ a function that returns a set of $\mathcal{T}$-lemmas $\{C_1, \ldots, C_K\}$ which rules out $P_{\boldsymbol{\alpha}}(\varphi)$. To this end, $Cl_{\boldsymbol{\alpha}}(\top)$ returns a set of $\mathcal{T}$-lemmas ruling out all $\mathcal{T}$-inconsistent truth assignments on $\boldsymbol{\alpha}$.

Example 7.1.2 ($Cl_{\boldsymbol{\alpha}}(\varphi)$). Consider φ_1 as in Example 7.1.1. Then

$$Cl_{\boldsymbol{\alpha}}(\varphi_1) = \{\neg(x \leq 0) \vee \neg(x = 1)\}$$

is a set of $\mathcal{T}$-lemmas ruling out the only truth assignment in $P_{\boldsymbol{\alpha}}(\varphi_1)$.

Remark 7.1.1. Typically, $\mathcal{T}$-lemmas are *much smaller* than partial or total assignments, and thus than the blocking clauses negating them (see Section 1.3.2), as $\mathcal{T}$-lemmas negate *conflicting cores* of $\mathcal{T}$-atoms. Also, whereas a $\mathcal{T}$-consistent partial assignment only implies that *one* of its total extensions is $\mathcal{T}$-consistent, a $\mathcal{T}$-inconsistent one implies that *all* its total extensions are $\mathcal{T}$-inconsistent. Thus, whereas a blocking clause usually rules out only *few* elements of $H_{\boldsymbol{\alpha}}(\varphi)$, a $\mathcal{T}$-lemma rules out *many* elements of $P_{\boldsymbol{\alpha}}(\varphi)$. Hence, unless $H_{\boldsymbol{\alpha}}(\varphi)$ is drastically smaller than $P_{\boldsymbol{\alpha}}(\varphi)$, most of $\mathcal{T}$-AllSAT search is typically spent ruling out assignments in $H_{\boldsymbol{\alpha}}(\varphi)$.

7.2 Theory-Lemma Enumeration Techniques

We present four theory-agnostic methods for enumerating a set $Cl_{\boldsymbol{\alpha}}(\varphi)$ of $\mathcal{T}$-lemmas to rule out $P_{\boldsymbol{\alpha}}(\varphi)$ (see Definition 7.1.2). In Section 7.2.1 we describe a simple method which we will use as a baseline. This method relies on basic, total $\mathcal{T}$-AllSAT enumeration (see Section 1.3.1), which is often intractable on real-world instances due to the combinatorial explosion of total assignments. To address this issue, we introduce three orthogonal improvements that drastically enhance scalability.

First, in Section 7.2.2 we describe a cube-and-conquer strategy that can be naturally parallelized. Second, in Section 7.2.3 we prove that projecting the enumeration onto the $\mathcal{T}$-atoms guarantees finding a complete set of $\mathcal{T}$-lemmas, thereby enabling more efficient enumeration. Finally, in Section 7.2.4 we prove that when the atoms of φ can be partitioned into disjoint sets, $\mathcal{T}$-lemmas can be enumerated for each partition separately. This results in a dramatic improvement in the efficiency of enumeration. Notably, these three improvements are orthogonal and can be combined. For the sake of compactness, all the proofs [51] are deferred to Appendix D.1.

7.2.1 Enumerating $\mathcal{T}$-lemmas via $\mathcal{T}$-AllSAT

Algorithm 10 LemmaEnumerator-$\mathcal{T}$-AllSAT($\varphi[\boldsymbol{\alpha}]$, $\boldsymbol{\alpha'}$)
Input: $\varphi[\boldsymbol{\alpha}]$: a $\mathcal{T}$-formula; $\boldsymbol{\alpha'}$: set of atoms to project on (default: $\boldsymbol{\alpha}$)
Output: $Cl_{\boldsymbol{\alpha}}$: a set of $\mathcal{T}$-lemmas

1: _, $Cl_{\boldsymbol{\alpha}} \leftarrow$ Projected-$\mathcal{T}$-AllSAT$_{\mathsf{total}}(\varphi[\boldsymbol{\alpha}], \boldsymbol{\alpha'})$ {Keep only lemmas}
2: **return** $Cl_{\boldsymbol{\alpha}}$

The baseline approach for $\mathcal{T}$-lemma enumeration is outlined in Algorithm 10 and consists of performing a call to *total*, *non-projected* $\mathcal{T}$-AllSAT on $\varphi[\boldsymbol{\alpha}]$, using the procedure in Algorithm 1 with $\boldsymbol{\alpha'} \stackrel{\text{def}}{=} \boldsymbol{\alpha}$ and no minimization. (In Algorithm 10 we also consider the case with $\boldsymbol{\alpha'} \subset \boldsymbol{\alpha}$ because it will be used in Sections 7.2.3 and 7.2.4.)

The $\mathcal{T}$-lemmas learned by the SMT solver during such an enumeration rule out all $\mathcal{T}$-inconsistent truth assignments in $P_{\boldsymbol{\alpha}}(\varphi)$. In fact, the algorithm terminates when $\varphi \wedge \bigwedge_{\eta_i \in H_{\boldsymbol{\alpha}}(\varphi)} \neg\eta_i$ is $\mathcal{T}$-unsatisfiable, i.e., when the solver has found a set $Cl_{\boldsymbol{\alpha}}$ of $\mathcal{T}$-lemmas s.t.

$$\varphi \wedge \bigwedge_{\eta_i \in H_{\boldsymbol{\alpha}}(\varphi)} \neg\eta_i \wedge \bigwedge_{C \in Cl_{\boldsymbol{\alpha}}} C \equiv_{\mathbb{B}} \bot.$$

This guarantees that $Cl_{\boldsymbol{\alpha}}$ rules out all $\mathcal{T}$-unsatisfiable truth assignments in $P_{\boldsymbol{\alpha}}(\varphi)$.

It is important to note that we must enumerate *total* truth assignments. Indeed, enumerating *partial* truth assignments may produce a set of $\mathcal{T}$-lemmas that does not rule out $P_{\boldsymbol{\alpha}}(\varphi)$ completely. This is because the definition of partial $\mathcal{T}$-AllSAT (see Section 1.3.1) only requires that, for each partial assignment, all its total extensions *propositionally* satisfy φ, but only one extension needs to be $\mathcal{T}$-consistent. The following example illustrates why this can lead to missing $\mathcal{T}$-lemmas.

Example 7.2.1 (Partial $\mathcal{T}$-AllSAT may miss $\mathcal{T}$-lemmas). Let $\boldsymbol{\alpha} \stackrel{\text{def}}{=} \{(x = 0), (x = 1)\}$ and $\varphi \stackrel{\text{def}}{=} (x = 0) \vee (x = 1)$.

Consider a disjoint, partial $\mathcal{T}$-AllSAT enumeration as performed by Algorithm 1 with $\boldsymbol{\alpha'} = \boldsymbol{\alpha}$ and minimization enabled. A possible run may proceed as follows.

The SMT solver sets $\neg(x = 0)$, so that unit propagation forces $(x = 1)$. The resulting total assignment $\eta_1 \stackrel{\text{def}}{=} \{\neg(x = 0), (x = 1)\}$ is $\mathcal{T}$-consistent (line 5). Minimization then drops $\neg(x = 0)$ and returns the partial assignment $\mu_1 \stackrel{\text{def}}{=} \{(x = 1)\}$ (line 6). Notice that no $\mathcal{T}$-conflict is exposed, and no $\mathcal{T}$-lemma is generated. The enumerator blocks the returned cube by adding $\neg(x = 1)$ (line 9).

On the next iteration, $\neg(x = 1)$ is enforced, and unit propagation forces $(x = 0)$. The resulting total assignment $\eta_2 \stackrel{\text{def}}{=} \{(x = 0), \neg(x = 1)\}$ is again $\mathcal{T}$-consistent (line 5). Minimization (line 6) cannot reduce η_2 because of the blocking clause $\neg(x = 1)$, and η_2 is then blocked by adding $\neg(x = 0) \vee (x = 1)$ (line 9), terminating the search.

Along this run, no $\mathcal{T}$-conflict is encountered, hence no $\mathcal{T}$-lemma is generated, since the SMT solver never explored the $\mathcal{T}$-inconsistent total assignment $\{(x = 0), (x = 1)\}$ which belongs to $P_{\boldsymbol{\alpha}}(\varphi)$.

Remark 7.2.1. For the case $\boldsymbol{\alpha'} \subset \boldsymbol{\alpha}$, the algorithm, in general, does not find a complete set of $\mathcal{T}$-lemmas ruling out $P_{\boldsymbol{\alpha}}(\varphi)$. In Sections 7.2.3 and 7.2.4, however, we will show some special cases for $\boldsymbol{\alpha'}$ where the projected version is used without affecting completeness.

7.2.2 Cube and Conquer (and Parallelize)

Algorithm 11 LemmaEnumeratorC&C($\varphi[\boldsymbol{\alpha}]$, $\boldsymbol{\alpha'}$)
Input: $\varphi[\boldsymbol{\alpha}]$: a $\mathcal{T}$-formula; $\boldsymbol{\alpha'}$: set of atoms to project on (default: $\boldsymbol{\alpha}$)
Output: $Cl_{\boldsymbol{\alpha}}$: a set of $\mathcal{T}$-lemmas

1: $M_{\boldsymbol{\alpha'}p}, Cl_{\boldsymbol{\alpha}p} \leftarrow \mathsf{EnumerateCubes}(\varphi[\boldsymbol{\alpha}], \boldsymbol{\alpha'})$ {Enumerate $\mathcal{T}$-sat cubes on $\boldsymbol{\alpha'}$}
2: $Cl_{\boldsymbol{\alpha}} \leftarrow Cl_{\boldsymbol{\alpha}p}$
3: **for** $\mu_i \in M_{\boldsymbol{\alpha'}p}$ **do** {Parallel loop}
4: $_, Cl_{\boldsymbol{\alpha}t} \leftarrow \mathsf{Projected\text{-}}\mathcal{T}\text{-}\mathsf{AllSAT}_{\mathsf{total}}((\varphi \wedge \mu_i)[\boldsymbol{\alpha}], \boldsymbol{\alpha'})$
5: $Cl_{\boldsymbol{\alpha}} \leftarrow Cl_{\boldsymbol{\alpha}} \cup Cl_{\boldsymbol{\alpha}t}$
6: **end for**
7: **return** $Cl_{\boldsymbol{\alpha}}$

Although enumerating partial assignments is not sufficient to find a set of $\mathcal{T}$-lemmas that rules out $P_{\boldsymbol{\alpha}}(\varphi)$, as seen in Example 7.2.1, a set of *disjoint, $\mathcal{T}$-satisfiable* partial assignments can provide a useful decomposition of the Boolean search space into a family of residual, $\mathcal{T}$-satisfiable subproblems. To exploit this decomposition, in Algorithm 11 we outline a cube-and-conquer approach for $\mathcal{T}$-lemma enumeration, inspired by the cube-and-conquer approaches for SAT [105] and, recently, for model counting [233]. We first consider the case where $\boldsymbol{\alpha'} = \boldsymbol{\alpha}$.

First (line 1), the algorithm invokes some routine to enumerate $M_{\boldsymbol{\alpha}p} = \{\mu_1, \ldots, \mu_k\}$, a set of pairwise-disjoint, $\mathcal{T}$-satisfiable cubes on (a subset of) $\boldsymbol{\alpha}$ such that every $\eta_j \in H_{\boldsymbol{\alpha}}(\varphi)$ extends some μ_i. We also collect the $\mathcal{T}$-lemmas $Cl_{\boldsymbol{\alpha}p}$ learned during this enumeration. In the next paragraphs, we discuss two possible implementations of this procedure.

Then, for each $\mu_i \in M_{\boldsymbol{\alpha}p}$ (lines 3-6), the algorithm performs a *total* $\mathcal{T}$-AllSAT enumeration on $\varphi \wedge \mu_i$. Possibly, in line 4, we could also conjoin the previously learned $\mathcal{T}$-lemmas $Cl_{\boldsymbol{\alpha}p}$ to prevent the SMT solver from rediscovering them in the residual subproblems. (We noticed, however, that in practice this can be counterproductive, as it may slow down the solver by adding many clauses to the residual formulas.) After each iteration, we save the lemmas learned by total $\mathcal{T}$-AllSAT (line 5).

Overall, instead of one large total $\mathcal{T}$-AllSAT run, we perform a cheaper decomposition step and many smaller total $\mathcal{T}$-AllSAT runs on restricted residual formulas, whose combined cost is typically lower. Also, the residual enumerations can be parallelized by assigning each μ_i to a process and finally merging the lemmas produced by the processes.

In our architecture, processes execute in isolation, and distinct processes may independently find identical or redundant lemmas. In Section 7.3, we show that in practice the number of lemmas found is comparable to that of the $\mathcal{T}$-AllSAT baseline.

Algorithm 12 EnumerateCubesByProjectedEnumeration($\varphi[\boldsymbol{\alpha}]$, $\boldsymbol{\alpha'}$, m)
Input: $\varphi[\boldsymbol{\alpha}]$: a $\mathcal{T}$-formula; $\boldsymbol{\alpha'}$: set of atoms to project on; m: min. number of cubes
Output: $M_{\boldsymbol{\alpha'}p}$: a partition of $\mathcal{T}$-satisfiable cubes on $\boldsymbol{\alpha'}$; $Cl_{\boldsymbol{\alpha}p}$: a set of $\mathcal{T}$-lemmas

1: $score \leftarrow \mathsf{ComputeAtomScore}(\boldsymbol{\alpha'})$ {Centrality score for each atom}
2: $M_{\boldsymbol{\alpha'}p} \leftarrow \{\top\}$ {Partition of cubes on $\boldsymbol{\alpha'}$}
3: $Cl_{\boldsymbol{\alpha}p} \leftarrow \emptyset$ {$\mathcal{T}$-lemmas learned}
4: $\boldsymbol{\alpha'}_{\mathsf{rem}} \leftarrow \boldsymbol{\alpha'}$ {Atoms not yet projected on}
5: $k \leftarrow \mathsf{InitialBatchSize}(\boldsymbol{\alpha'}, m)$ {Number of atoms to project on}
6: **while** $|M_{\boldsymbol{\alpha'}p}| < m$ and $\boldsymbol{\alpha'}_{\mathsf{rem}} \neq \emptyset$ **do**
7: $\quad\boldsymbol{\alpha'}_{\mathsf{next}} \leftarrow \mathsf{PickAtoms}(\boldsymbol{\alpha'}_{\mathsf{rem}}, k, score)$ {Pick the next atoms to project on}
8: $\quad\boldsymbol{\alpha'}_{\mathsf{rem}} \leftarrow \boldsymbol{\alpha'}_{\mathsf{rem}} \setminus \boldsymbol{\alpha'}_{\mathsf{next}}$
9: $\quad M_{\boldsymbol{\alpha'}p}, Cl'_{\boldsymbol{\alpha}p} \leftarrow \mathsf{ExtendCubes}(M_{\boldsymbol{\alpha'}p}, \boldsymbol{\alpha'}_{\mathsf{next}})$ {Extend cubes over $\boldsymbol{\alpha'}_{\mathsf{next}}$}
10: $\quad Cl_{\boldsymbol{\alpha}p} \leftarrow Cl_{\boldsymbol{\alpha}p} \cup Cl'_{\boldsymbol{\alpha}p}$
11: $\quad k \leftarrow \mathsf{NextBatchSize}(\boldsymbol{\alpha'}, \boldsymbol{\alpha'}_{\mathsf{rem}}, M_{\boldsymbol{\alpha'}p}, m)$
12: **end while**
13: **return** $M_{\boldsymbol{\alpha'}p}, Cl_{\boldsymbol{\alpha}p}$

The correctness of the approach follows from the fact that, overall, all the total truth assignments are explored, so that for each $\rho \in P_{\boldsymbol{\alpha}}(\varphi)$ a $\mathcal{T}$-lemma is eventually generated that rules it out. For the case $\boldsymbol{\alpha'} \subset \boldsymbol{\alpha}$, Remark 7.2.1 applies for Algorithm 11 as well.

We now discuss two possible implementations for EnumerateCubes, called in line 1.

Cubes by partial $\mathcal{T}$-AllSAT. A simple implementation consists of performing a *partial* $\mathcal{T}$-AllSAT enumeration on φ, i.e., $\mathcal{T}\text{-}\mathsf{AllSAT}_{\mathsf{partial}}(\varphi[\boldsymbol{\alpha}], \boldsymbol{\alpha'})$ (see Algorithm 1). This, however, offers no control over the number of generated cubes: generating too few cubes limits parallelization, while generating too many cubes creates a sequential bottleneck. This unpredictability motivates a more controlled approach.

Cubes by projected $\mathcal{T}$-AllSAT. Traditional cube-and-conquer approaches for SAT [105] and SMT [107] are implemented within a solver, leveraging efficient data structures for atom-selection heuristics, such as lookaheads. In Algorithm 12 we outline a different approach that uses a $\mathcal{T}$-AllSAT solver as a black box. The algorithm takes an additional parameter m indicating the minimum number of cubes to enumerate.

The algorithm first computes a centrality score for each atom $\alpha \in \boldsymbol{\alpha'}$ (line 1). While various heuristics are possible, we use a simple variable-frequency score:

$$score(\alpha) \stackrel{\text{def}}{=} \sum_{x \in Vars(\alpha)} freq(x)$$

where $Vars(\alpha)$ is the set of variables in α (α itself for $\mathbb{B}$-atoms), and $freq(x)$ counts the occurrences of variable x across all atoms in $\boldsymbol{\alpha'}$.

Then, the algorithm initializes a partition of cubes $M_{\boldsymbol{\alpha'}p}$ on $\boldsymbol{\alpha'}$ with a single cube $\top$ (line 2). Assuming an ideal case, the function InitialBatchSize initially sets $k =$

$\max(1, \lceil \log_2(m) \rceil)$ as the number of atoms to project on (line 5). In each iteration, the algorithm picks a batch of k atoms $\boldsymbol{\alpha'}_{\mathsf{next}}$ from the remaining set $\boldsymbol{\alpha'}_{\mathsf{rem}}$ based on their scores (line 7), and extends the current cubes in $M_{\boldsymbol{\alpha'}p}$ over these selected atoms (line 9). We implement this extension by calling $\mathsf{Projected}\text{-}\mathcal{T}\text{-}\mathsf{AllSAT}_{\mathsf{total}}((\varphi \wedge \mu_i)[\boldsymbol{\alpha}], \boldsymbol{\alpha'}_{\mathsf{next}})$ (see Algorithm 1) for each cube $\mu_i \in M_{\boldsymbol{\alpha'}p}$. After each iteration, the algorithm updates $\boldsymbol{\alpha'}_{\mathsf{rem}}$ (line 8) and dynamically computes the next batch size k (line 11), as follows.

We estimate the *branching factor* as $bf \stackrel{\text{def}}{=} |M_{\boldsymbol{\alpha'}p}|^{1/|\boldsymbol{\alpha'} \setminus \boldsymbol{\alpha'}_{\mathsf{rem}}|}$. Intuitively, bf is the average factor by which the number of cubes grows for each projected atom. Projecting k more atoms is therefore expected to increase the number of cubes from $|M_{\boldsymbol{\alpha'}p}|$ to approximately $|M_{\boldsymbol{\alpha'}p}| \cdot bf^k$. We choose k so that $|M_{\boldsymbol{\alpha'}p}| \cdot bf^k \geq m$ giving $k = \max\left(1, \left\lceil \log_{bf}(m/|M_{\boldsymbol{\alpha'}p}|) \right\rceil\right)$, which ensures that at least one atom is projected. We iterate this process until we reach the desired number of cubes or exhaust all atoms.

7.2.3 Projection on Theory-Atoms

Algorithm 13 $\mathsf{LemmaEnumeratorWithProjection}(\varphi[\boldsymbol{\alpha}], \mathsf{LemmaEnumerator})$
Input: $\varphi[\boldsymbol{\alpha}]$: a $\mathcal{T}$-formula; $\mathsf{LemmaEnumerator}$: a $\mathcal{T}$-lemma enumerator
Output: $Cl_{\boldsymbol{\alpha}}$: a set of $\mathcal{T}$-lemmas

1: $\boldsymbol{\alpha}^{\mathcal{T}} \leftarrow \mathsf{GetTheoryAtoms}(\boldsymbol{\alpha})$ {Extract theory atoms from $\boldsymbol{\alpha}$}
2: $Cl_{\boldsymbol{\alpha}} \leftarrow \mathsf{LemmaEnumerator}(\varphi[\boldsymbol{\alpha}], \boldsymbol{\alpha}^{\mathcal{T}})$
3: **return** $Cl_{\boldsymbol{\alpha}}$

Enumerating all total truth assignments, even when parallelized, is often infeasible on real-world instances. In this section, we spotlight that this is often unnecessary for finding a complete set of $\mathcal{T}$-lemmas, starting from the following observation.

Truth assignments explored during a $\mathcal{T}$-AllSAT enumeration, whether via the classic or the cube-and-conquer approach, consist of two components: an assignment to $\mathcal{T}$-atoms and an assignment to $\mathbb{B}$-atoms. We note that the $\mathcal{T}$-inconsistency of a truth assignment is determined exclusively by its projection onto the $\mathcal{T}$-atoms. Consequently, a standard $\mathcal{T}$-AllSAT-based lemma-enumeration procedure may inefficiently enumerate multiple truth assignments that share an identical $\mathcal{T}$-consistent projection but differ in their Boolean assignments. This redundancy is computationally wasteful, as variations in the Boolean assignment do not generate new $\mathcal{T}$-lemmas.

We can address this issue by performing $\mathcal{T}$-AllSAT *projected on the $\mathcal{T}$-atoms of* φ. The correctness follows from the following theorem (proof in Appendix D.1.1).

Theorem 7.2.1. *Let $\varphi[\boldsymbol{\alpha}]$ be a $\mathcal{T}$-formula. Let $\boldsymbol{\alpha}^{\mathbb{B}}$ and $\boldsymbol{\alpha}^{\mathcal{T}}$ be the sets of Boolean and theory atoms in $\boldsymbol{\alpha}$, respectively.*

Let $BC_{\boldsymbol{\alpha}^{\mathcal{T}}}(\varphi) \stackrel{def}{=} \{\neg\eta^{\mathcal{T}}[\boldsymbol{\alpha}^{\mathcal{T}}] \text{ s.t. } \eta^{\mathcal{T}}[\boldsymbol{\alpha}^{\mathcal{T}}] \wedge \eta^{\mathbb{B}}[\boldsymbol{\alpha}^{\mathcal{T}}] \in H_{\boldsymbol{\alpha}}(\varphi)\}$, and let $Cl_{\boldsymbol{\alpha}}$ be a set of $\mathcal{T}$-lemmas such that $BC_{\boldsymbol{\alpha}^{\mathcal{T}}}(\varphi) \cup Cl_{\boldsymbol{\alpha}}$ rules out $P_{\boldsymbol{\alpha}}(\varphi)$.

Then $Cl_{\boldsymbol{\alpha}}$ rules out $P_{\boldsymbol{\alpha}}(\varphi)$.

This leads to Algorithm 13. This procedure invokes a $\mathcal{T}$-AllSAT-based $\mathcal{T}$-lemma enumerator (e.g., Algorithm 10 or 11), projecting the enumeration onto $\mathcal{T}$-atoms only. If the lemma enumerator is based on Algorithm 1, which terminates when

$$\varphi \wedge \bigwedge_i \neg \eta_i^{\mathcal{T}}[\boldsymbol{\alpha}^{\mathcal{T}}] \wedge \bigwedge_{C \in Cl_{\boldsymbol{\alpha}}} C \equiv_{\mathbb{B}} \bot,$$

it follows from Theorem 7.2.1 that the returned set $Cl_{\boldsymbol{\alpha}}$ of $\mathcal{T}$-lemmas rules out $P_{\boldsymbol{\alpha}}(\varphi)$.

Remark 7.2.2. Note that the projected assignments can be *exponentially fewer* than the assignments on the full set of atoms. Also, the blocking clauses added by Algorithm 1 (line 9) are much shorter, as they only contain $\mathcal{T}$-atoms. This can lead to a drastic reduction in both enumeration time and memory usage.

7.2.4 Theory-driven Partitioning

Algorithm 14 LemmaEnumeratorWithPartitioning($\varphi[\boldsymbol{\alpha}]$, LemmaEnumerator)
Input: $\varphi[\boldsymbol{\alpha}]$: a $\mathcal{T}$-formula; LemmaEnumerator: a $\mathcal{T}$-lemma enumerator
Output: $Cl_{\boldsymbol{\alpha}}$: a set of $\mathcal{T}$-lemmas

1: $Cl_{\boldsymbol{\alpha}} \leftarrow \emptyset$
2: $\{\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_n\} \leftarrow$ PartitionAtoms($\boldsymbol{\alpha}$) {Partition atoms as in Fact 7.2.1}
3: **for** $i = 1$ **to** n **do**
4: $\quad \varphi_i \leftarrow$ RelevantConjuncts($\varphi, \boldsymbol{\alpha}_i$) {Get conjuncts of φ relevant to $\boldsymbol{\alpha}_i$ as in (7.2)}
5: $\quad Cl_{\boldsymbol{\alpha}i} \leftarrow$ LemmaEnumerator($\varphi_i[\boldsymbol{\alpha}], \boldsymbol{\alpha}_i$)
6: $\quad Cl_{\boldsymbol{\alpha}} \leftarrow Cl_{\boldsymbol{\alpha}} \cup Cl_{\boldsymbol{\alpha}i}$
7: **end for**
8: **return** $Cl_{\boldsymbol{\alpha}}$

We now present a further improvement for $\mathcal{T}$-lemma enumeration, for the case where the set of atoms $\boldsymbol{\alpha}$ can be partitioned into two or more disjoint sets w.r.t. variables and uninterpreted symbols. We first recall the following result.

Fact 7.2.1. *Let $\boldsymbol{\alpha} \stackrel{def}{=} \boldsymbol{\alpha}_1 \cup \boldsymbol{\alpha}_2$ be a set of atoms s.t. no atom in $\boldsymbol{\alpha}_1$ shares variables or uninterpreted symbols with any atom in $\boldsymbol{\alpha}_2$. Then, for any truth assignment $\eta^{\boldsymbol{\alpha}} \stackrel{def}{=} \eta_1[\boldsymbol{\alpha}_1] \wedge \eta_2[\boldsymbol{\alpha}_2]$, $\eta^{\boldsymbol{\alpha}}$ is $\mathcal{T}$-satisfiable iff both $\eta_1[\boldsymbol{\alpha}_1]$ and $\eta_2[\boldsymbol{\alpha}_2]$ are $\mathcal{T}$-satisfiable.*

Hence, if the atoms in $\boldsymbol{\alpha}$ can be partitioned according to the conditions of Fact 7.2.1, then in order to rule out a $\mathcal{T}$-inconsistent truth assignment $\rho^{\boldsymbol{\alpha}} \stackrel{\text{def}}{=} \rho_1[\boldsymbol{\alpha}_1] \wedge \rho_2[\boldsymbol{\alpha}_2] \in P_{\boldsymbol{\alpha}}(\varphi)$, a $\mathcal{T}$-lemma must rule out $\rho_1[\boldsymbol{\alpha}_1]$ or $\rho_2[\boldsymbol{\alpha}_2]$. This suggests a decomposition strategy: a $\mathcal{T}$-AllSAT enumeration projected on $\boldsymbol{\alpha}_1$ will find a set of $\mathcal{T}$-lemmas ruling out all $\rho^{\boldsymbol{\alpha}} \in P_{\boldsymbol{\alpha}}(\varphi)$ such that $\rho_1[\boldsymbol{\alpha}_1]$ is $\mathcal{T}$-inconsistent. Dually, $\mathcal{T}$-AllSAT projected on $\boldsymbol{\alpha}_2$ will find a set of $\mathcal{T}$-lemmas ruling out all $\rho^{\boldsymbol{\alpha}} \in P_{\boldsymbol{\alpha}}(\varphi)$ such that $\rho_2[\boldsymbol{\alpha}_2]$ is $\mathcal{T}$-inconsistent. Thus, a set $Cl_{\boldsymbol{\alpha}}(\varphi)$ can be found with two separate projected enumerations.

This intuition is formalized in the following theorem (proof in Appendix D.1.2).

Theorem 7.2.2. *Let $\varphi[\boldsymbol{\alpha}]$ be a $\mathcal{T}$-formula such that $\boldsymbol{\alpha} \stackrel{def}{=} \boldsymbol{\alpha}_1 \cup \boldsymbol{\alpha}_2$ and no atom in $\boldsymbol{\alpha}_1$ shares variables or uninterpreted symbols with any atom in $\boldsymbol{\alpha}_2$.*

Let $BC_{\boldsymbol{\alpha}_1}(\varphi) \stackrel{def}{=} \{\neg\eta_1[\boldsymbol{\alpha}_1] \text{ s.t. } \eta_1[\boldsymbol{\alpha}_1] \wedge \eta_2[\boldsymbol{\alpha}_2] \in H_{\boldsymbol{\alpha}}(\varphi)\}$, and let $Cl_{\boldsymbol{\alpha}}$ be a set of $\mathcal{T}$-lemmas such that $BC_{\boldsymbol{\alpha}_1}(\varphi) \cup Cl_{\boldsymbol{\alpha}}$ rules out $P_{\boldsymbol{\alpha}}(\varphi)$.

Let $BC_{\boldsymbol{\alpha}_2}(\varphi) \stackrel{def}{=} \{\neg\eta_2[\boldsymbol{\alpha}_2] \text{ s.t. } \eta_1[\boldsymbol{\alpha}_1] \wedge \eta_2[\boldsymbol{\alpha}_2] \in H_{\boldsymbol{\alpha}}(\varphi)\}$, and let $Cl'_{\boldsymbol{\alpha}}$ be a set of $\mathcal{T}$-lemmas such that $BC_{\boldsymbol{\alpha}_2}(\varphi) \cup Cl'_{\boldsymbol{\alpha}}$ rules out $P_{\boldsymbol{\alpha}}(\varphi)$.

Then $Cl_{\boldsymbol{\alpha}} \cup Cl'_{\boldsymbol{\alpha}}$ rules out $P_{\boldsymbol{\alpha}}(\varphi)$.

Note that this result can be easily generalized to the case where the set of atoms can be partitioned into more than two sets. Indeed, if $\boldsymbol{\alpha}_2$ can be further partitioned, we can apply the same reasoning recursively to it.

This result naturally leads to the enumeration strategy in Algorithm 14. The algorithm first partitions the set $\boldsymbol{\alpha}$ of atoms as in Fact 7.2.1, obtaining $\{\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_n\}$ (line 2). This step can be performed efficiently, for example, using a union-find data structure to compute the connected components of the graph of atoms, where two atoms are directly connected if they share a variable or an uninterpreted symbol. Note that, in principle, every $\mathbb{B}$-atom can be considered as belonging to a separate, singleton partition. In practice, we assume that all the $\mathbb{B}$-atoms belong to the same partition, and can be handled via projection as discussed in Section 7.2.3.

Then, for each partition $\boldsymbol{\alpha}_i$, the algorithm first isolates the relevant conjuncts of φ that contain at least one atom in $\boldsymbol{\alpha}_i$ (line 4), i.e., if $\varphi = \bigwedge_j \phi_j$, then

$$\varphi_i \stackrel{\text{def}}{=} \bigwedge_{j \text{ s.t. } Atoms(\phi_j) \cap \boldsymbol{\alpha}_i \neq \emptyset} \phi_j. \tag{7.2}$$

(We recall that in general φ may not be in CNF, and thus the conjuncts are not necessarily clauses.) We note that discarding the conjuncts that do not contain any atom in $\boldsymbol{\alpha}_i$ is sound, as by Fact 7.2.1 they cannot contribute to the $\mathcal{T}$-inconsistency of any truth assignment on $\boldsymbol{\alpha}_i$. Conversely, keeping them would increase the size of the formula and force the solver to explore parts of the search space that are irrelevant for the current atom partition, thereby slowing down the enumeration.

The algorithm then performs a $\mathcal{T}$-AllSAT-based $\mathcal{T}$-lemma enumeration on φ_i projected on $\boldsymbol{\alpha}_i$, and accumulates the enumerated $\mathcal{T}$-lemmas in $Cl_{\boldsymbol{\alpha}}$ (line 5). Possibly, in line 5, we could also conjoin the $\mathcal{T}$-lemmas $Cl_{\boldsymbol{\alpha}}$ learned in the previous iterations to prevent the SMT solver from rediscovering them. We observed, however, that in practice this can be counterproductive. This can be explained by the fact that the $\mathcal{T}$-lemmas learned in previous iterations typically contain atoms that are not in the current partition, and thus are not relevant to the current enumeration. They may instead increase the size of the formula significantly, slowing down the solver.

Finally (line 8), we return the set $Cl_{\boldsymbol{\alpha}}$, which, by Theorem 7.2.2, rules out $P_{\boldsymbol{\alpha}}(\varphi)$.

Similar considerations as in Remark 7.2.2 apply in this case if $n > 1$, as we decompose the single enumeration into n smaller projected enumerations.

7.3 Experimental Evaluation

We evaluate the performance of the $\mathcal{T}$-lemma enumeration techniques of Section 7.2 in terms of *time* required to enumerate $\mathcal{T}$-lemmas (Section 7.3.3). We also analyze the *number* of enumerated $\mathcal{T}$-lemmas (Section 7.3.4), and their median *size* (Section 7.3.5). We consider the following enumerators:

- Baseline, i.e., standard $\mathcal{T}$-AllSAT (Algorithm 10, Section 7.2.1);
- C&C, i.e., cube-and-conquer (Algorithm 11, Section 7.2.2), generating cubes by partial $\mathcal{T}$-AllSAT (Section 7.2.2);
- C&C+Proj., i.e., C&C plus projection on $\mathcal{T}$-atoms (Algorithm 13, Section 7.2.3);
- C&C+Proj.+Part., i.e., C&C+Proj. plus partitioning (Algorithm 14, Section 7.2.4).
- C&CP+Proj.+Part., i.e., C&C+Proj.+Part. but generating cubes by projected $\mathcal{T}$-AllSAT (Section 7.2.2).

Baseline was first proposed in [152], whereas C&C, C&C+Proj., and C&C+Proj.+Part. were proposed in [50]. C&CP+Proj.+Part. is a new enumerator that we first propose and evaluate in this chapter.

7.3.1 Information for the Reproducibility of the Experiments

We use PySMT [85] to manipulate SMT formulas and interface with solvers. As $\mathcal{T}$-AllSAT solver, we use MathSAT5 [48], as it implements the enumeration procedure described in Algorithm 1. We set the option `-dpll.allsat_minimize_model` to true or false to enumerate partial or total truth assignments, respectively. We disable potentially non-validity-preserving simplifications by setting `-preprocessor.toplevel_propagation=false` and `-preprocessor.simplification=0`. We also set `-dpll.store_tlemmas=true` to store the generated $\mathcal{T}$-lemmas. Moreover, to avoid splitting-on-demand and the consequent introduction of new atoms also in the $\mathcal{T}$-lemmas, we set `-theory.la.split_rat_eq=false` and `-theory.la.laz_internal_branch_and_bound=true`. In particular, the first option disables splitting on negated $\mathcal{LRA}$-equalities, while the latter encourages the solver to use its internal branch-and-bound procedure for $\mathcal{LIA}$ [9].

To find short partial assignments for non-CNF formulas, we use the technique in Chapter 3. The code for the algorithms[2] and for reproducing the experiments[3] are publicly available.

All the experiments were executed on three machines with identical hardware configurations. Each machine was equipped with a 28-core Intel Xeon Gold 6238R CPU at 2.20 GHz and 128 GB of RAM, and ran Ubuntu Linux 24.04. Cube-and-conquer enumerators use 50 parallel processes. We set a timeout of 3600 s for synthetic and temporal planning problems, and 14400 s for the harder numeric planning problems.

[2] `https://github.com/ecivini/tlemmas-enumeration/tree/develop`

[3] `https://github.com/ecivini/tlemmas-enumeration-testbench/tree/develop`

7.3.2 Benchmarks

We considered three sets of benchmarks.

Synthetic benchmarks. The first set, introduced in [152], consists of 450 SMT($\mathcal{LRA}$) and SMT($\mathcal{LIA}$) synthetic instances inspired by WMI problems from [161]. We generated random formulas by nesting Boolean operators up to fixed depths ranging from 4 to 8, involving 10 Boolean atoms and 10 real variables.

Temporal planning benchmarks. The second set, first proposed in [50], consists of 90 SMT($\mathcal{LRA}$) instances encoding temporal planning problems from the industrially inspired `painter` domain [224]. For each problem, we generated the SMT encoding for action applications up to a fixed horizon $h \in \{3, 4, 5\}$ using the TEMPEST tool.[4] The encoding is described in [173]; since [173] focuses on optimal temporal planning, in our encoding the optimization component and the abstract step are discarded. We also removed the goal constraints to increase the number of solutions and make the problems more challenging for $\mathcal{T}$-lemma enumeration.

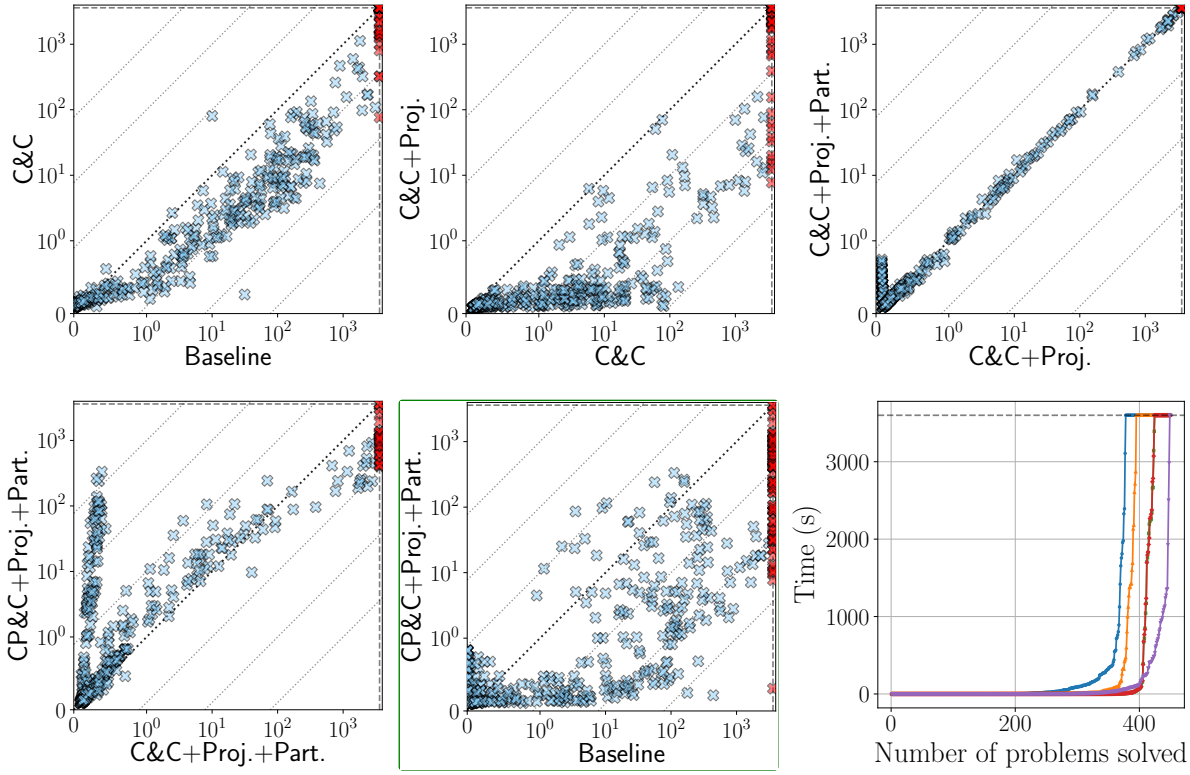
Numeric planning benchmarks. We also consider here a new set of SMT($\mathcal{LRA}$) benchmarks encoding numeric planning problems using the `block-grouping` domain from the International Planning Competition 2023.[5] Since the competition instances are too challenging for our task, we crafted a set of smaller instances by reducing the number of blocks. Specifically, we place $N \in \{2, 3, 4, 5, 6\}$ blocks randomly on a 10×10 grid, generating 4 instances for each value of N. The problem consists of moving the blocks to reach a goal configuration in which they are grouped according to a prescribed pattern (see also Section 8.5.2 where we describe the goals in more detail). We use TEMPEST[4] to generate the SMT encoding of action applications up to a fixed horizon $h \in \{3, 4, 5\}$, and, as for the temporal planning problems, we remove the goal constraints. Moreover, we slightly modify the encoding to avoid enumerating semantically equivalent plans that are represented by different truth assignments, as detailed in Appendix D.2.

7.3.3 Results on Enumeration Time

Figures 7.1 to 7.3 present the results for enumeration time on the synthetic, temporal planning, and numeric planning benchmarks, respectively. Each figure is organized as follows: the first and second rows show scatter plots comparing the strategies pairwise. (Notice the logarithmic scale of the axes!) We focus on incremental improvements, i.e., Baseline vs. C&C, C&C vs. C&C+Proj., and C&C+Proj. vs. C&C+Proj.+Part., and also compare the newest C&C strategy with the previous one, i.e. C&C+Proj.+Part. vs. C&CP+Proj.+Part.. We also show the overall speed-up (Baseline vs. C&CP+Proj.+Part.), highlighted in green. The second row also contains a cactus plot comparing all strategies, and the third row contains a table summarizing the number of instances and timeouts.

[4] `https://github.com/fbk-pso/tempest`

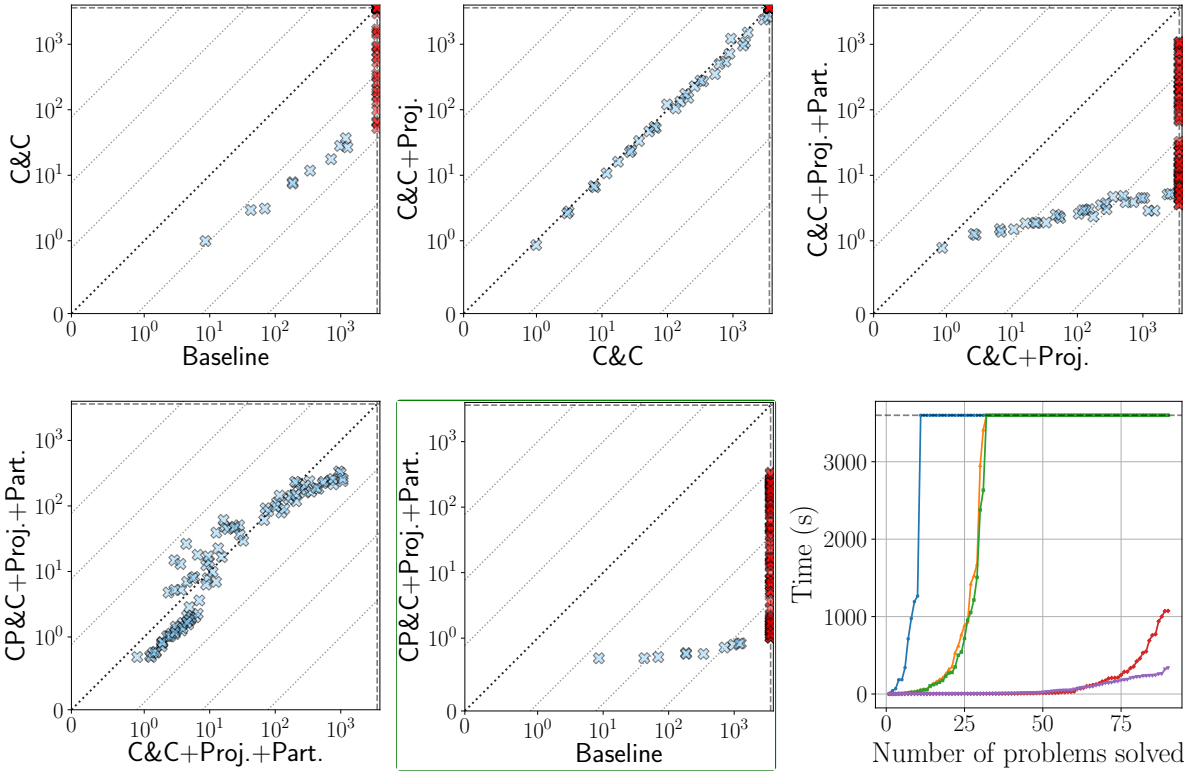
[5] `https://github.com/ipc2023-numeric/ipc2023-dataset`



Strategy	Timeouts	Total
Baseline	73	450
C&C	56	450
C&C+Proj.	26	450
C&C+Proj.+Part.	27	450
C&CP+Proj.+Part.	2	450

Figure 7.1: $\mathcal{T}$-lemma enumeration time for different strategies on synthetic problems. (Notice that in the cactus plot the green and red lines are almost indistinguishable.)

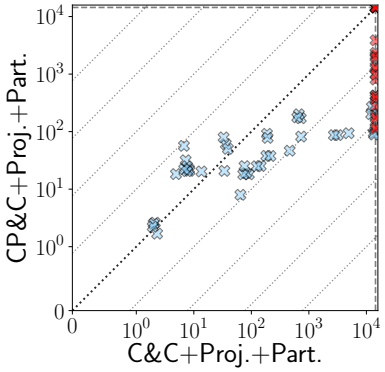
Synthetic benchmarks. (Figure 7.1) We notice that C&C significantly improves w.r.t. the baseline by up to 2 orders of magnitude, solving 17 more problems within the timeout. The improvement is even more pronounced when we add projection: C&C+Proj. solves 30 more problems and drastically reduces enumeration time for problems already solved by C&C. In contrast, partitioning brings no benefit, since only 92 of 450 problems have at least two partitions, and the hardest among them is already solved by C&C+Proj. in under 1 second, leaving no room for improvement. The new cubing strategy C&CP+Proj.+Part. allows us to solve 25 more problems w.r.t. C&C+Proj.+Part., though with some overhead on smaller instances. This occurs when a problem has few satisfying assignments: the cubing strategy of Algorithm 11 may require many iterations, each invoking $\mathcal{T}$-AllSAT on a new set of atoms, whereas the partial-$\mathcal{T}$-AllSAT-based strategy is very fast in such cases. Overall, the green-highlighted scatter plot is remarkable, showing a dramatic reduction in the time required by our best configuration w.r.t. the baseline.

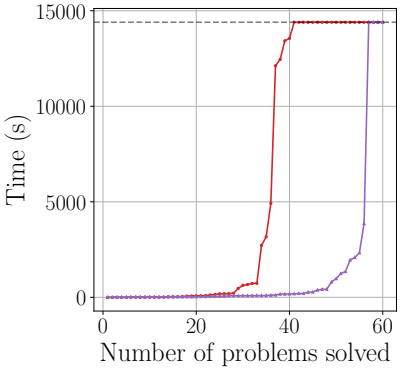


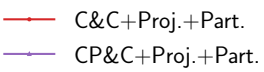
Strategy	Timeouts	Total
Baseline	80	90
C&C	59	90
C&C+Proj.	59	90
C&C+Proj.+Part.	0	90
C&CP+Proj.+Part.	0	90

Figure 7.2: $\mathcal{T}$-lemma enumeration time for different strategies on temporal planning benchmarks.

Temporal planning benchmarks. (Figure 7.2) On these problems, we can see a similar trend, with some differences. While C&C is much faster than the baseline, solving 21 more problems within the timeout, the addition of projection does not yield any improvement. This is likely due to the fact that in planning, the assignment on $\mathcal{T}$-atoms is a deterministic consequence of the assignment on $\mathbb{B}$-atoms, so that projecting does not reduce the number of $\mathcal{T}$-satisfiable truth assignments explored by the solver. On the other hand, partitioning is extremely effective here, dramatically reducing the enumeration time. In these problems, we have 5 $\mathcal{T}$-atom partitions: atoms encoding time constraints, atoms with constraints on the values of the fluents, and three partitions with constraints on the parameters of three different actions. By reasoning separately on these partitions, we achieve a speedup of several orders of magnitude, solving problems up to horizon 5 that were previously intractable. As C&C+Proj.+Part. is already very fast, the new cubing strategy brings only a slight improvement, which is mostly visible in the cactus plot.







Strategy	Timeouts	Total
C&C+Proj.+Part.	20	60
C&CP+Proj.+Part.	4	60

Figure 7.3: $\mathcal{T}$-lemma enumeration time for different strategies on numeric planning benchmarks.

Numeric planning benchmarks. (Figure 7.3) On these harder problems, we only compare the two best strategies, C&C+Proj.+Part. and C&CP+Proj.+Part., as they are too challenging for the other strategies.

Notably, here the cube-by-projected-enumeration strategy is dramatically more effective than the previous one, allowing us to solve all but 4 problems within the timeout, and achieving a speedup of up to two orders of magnitude on problems that are already solved with the previous strategy.

7.3.4 Results on the Number of Enumerated $\mathcal{T}$-Lemmas

We now analyze the number of $\mathcal{T}$-lemmas enumerated by the different strategies. Figures 7.4, 7.5 and 7.6 show scatter plots for the synthetic, temporal planning, and numeric planning benchmarks, respectively. As for the time plots, we present pairwise comparisons of incremental improvements. We remark that we show points only for problems solved by both compared strategies, so plots involving the least effective strategies have only a limited number of points.

Overall, there is no substantial difference in the number of $\mathcal{T}$-lemmas enumerated by different strategies. We can, however, observe some trends.

First, C&C tends to enumerate more $\mathcal{T}$-lemmas than Baseline. This is expected, as discussed in Section 7.2.2, since the parallel enumeration can discover redundant $\mathcal{T}$-lemmas.

Second, both projection and partitioning help reduce the number of enumerated $\mathcal{T}$-lemmas. This fact can have multiple explanations. One is that early-pruning calls to the $\mathcal{T}$-solver [9] may cause the SMT solver to enumerate $\mathcal{T}$-lemmas that also rule out $\mathcal{T}$-unsatisfiable assignments that do not propositionally satisfy the formula. By narrowing the search space, both projection and partitioning help mitigate this undesired behavior.

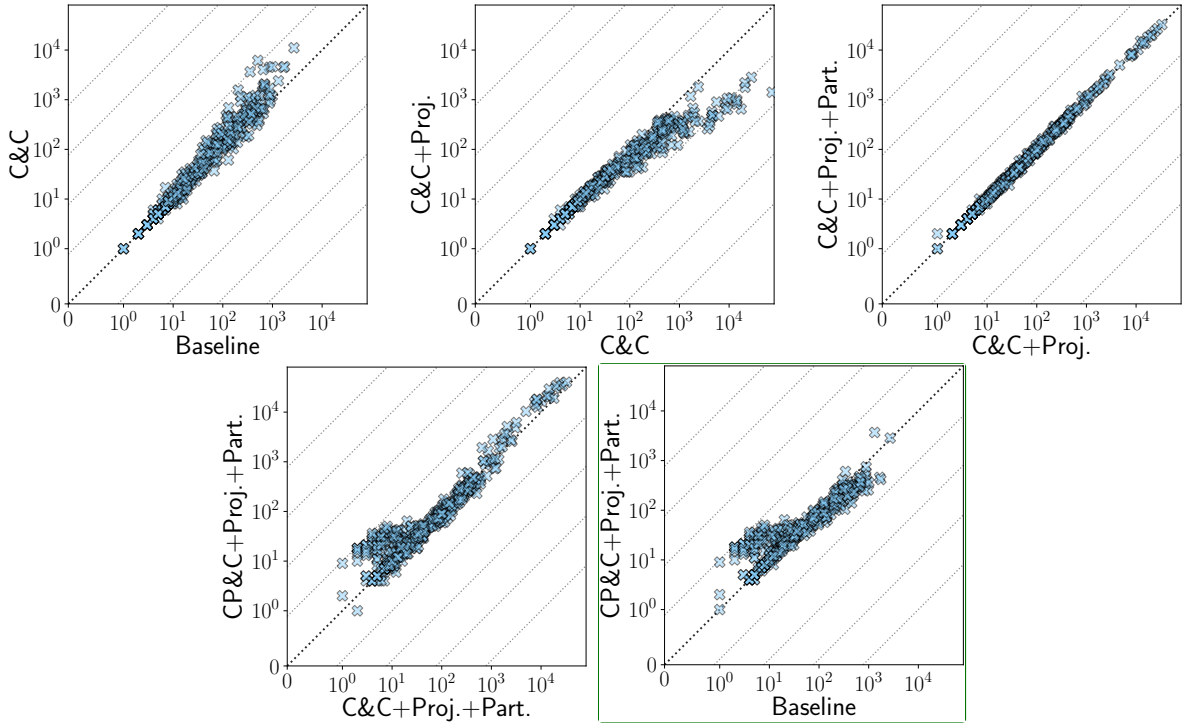


Figure 7.4: Number of $\mathcal{T}$-lemmas for different strategies on synthetic benchmarks.

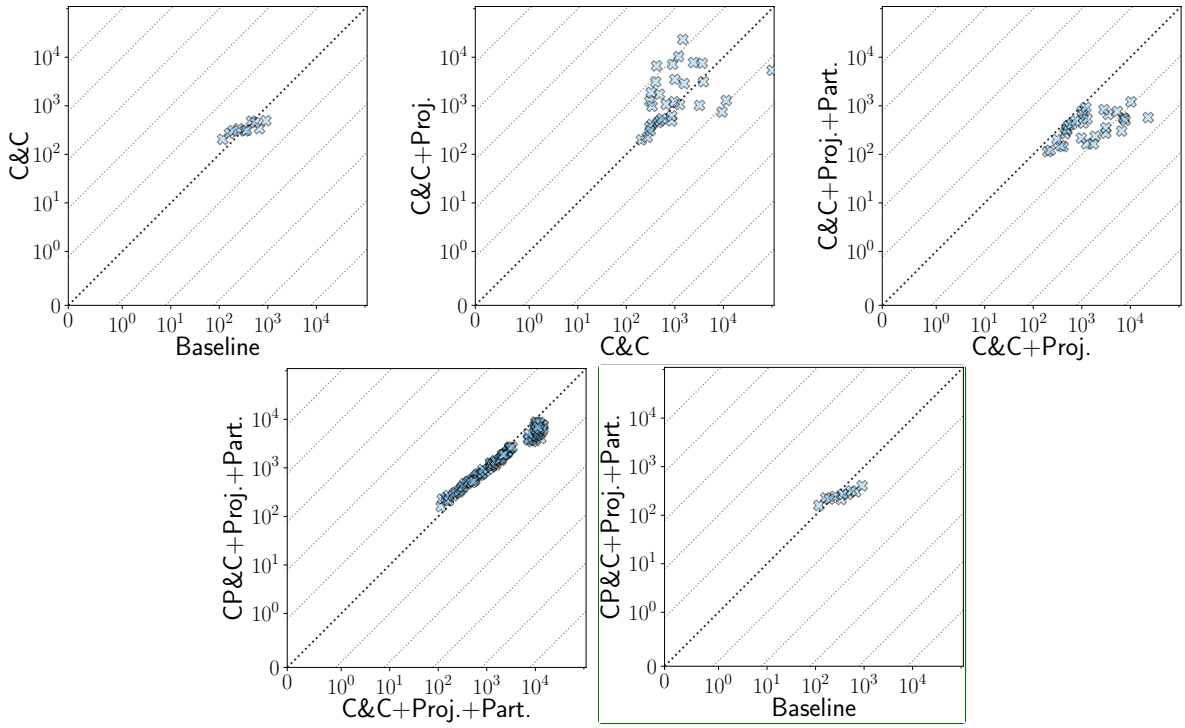


Figure 7.5: Number of $\mathcal{T}$-lemmas for different strategies on temporal planning benchmarks.

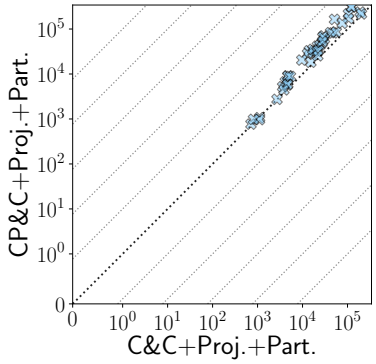


Figure 7.6: Number of $\mathcal{T}$-lemmas for different strategies on numeric planning benchmarks.

Additionally, with C&C, two parallel processes can find two $\mathcal{T}$-inconsistent assignments sharing the same $\mathcal{T}$-literals and different $\mathbb{B}$-literals, inevitably generating two redundant $\mathcal{T}$-lemmas. By contrast, with projection, the partial assignments fed to different processes differ only on $\mathcal{T}$-literals, thereby mitigating this issue. Lastly, partitioning can help enumerate smaller $\mathcal{T}$-lemmas (see next), which have stronger pruning power, thus reducing the number of enumerated $\mathcal{T}$-lemmas.

Comparing the two cubing strategies, there is no clear trend, except for the numeric planning problems (Figure 7.6), where the new cubing strategy tends to enumerate slightly more $\mathcal{T}$-lemmas. This can be explained by the fact that the new strategy is more effective and enables much better parallelization, which can lead to more redundant $\mathcal{T}$-lemmas being generated.

7.3.5 Results on the Size of $\mathcal{T}$-lemmas

As a further insight, we analyze the median size (i.e., the number of literals) of the $\mathcal{T}$-lemmas enumerated by the different strategies. The results are shown in Figures 7.7, 7.8 and 7.9 for synthetic, temporal planning, and numeric planning problems, respectively. We only show points for problems solved by both compared strategies, so plots involving the least effective strategies only have a limited number of points.

We observe that different techniques do not consistently reduce or increase the size of $\mathcal{T}$-lemmas, with a notable exception for partitioning, which, on planning problems, allows us to find much smaller $\mathcal{T}$-lemmas. We conjecture that this may be due to the fact that, with projected enumeration, the literal selection heuristics of the SMT solver may prefer selecting projection atoms first. Hence, early-pruning calls [9] to the $\mathcal{T}$-solver are more likely to cut $\mathcal{T}$-inconsistent branches involving only projection atoms. This hinders the generation of larger $\mathcal{T}$-lemmas with atoms from different partitions, which are intrinsically redundant by Fact 7.2.1. The two cubing strategies are comparable on all the benchmarks, with no clear winner.

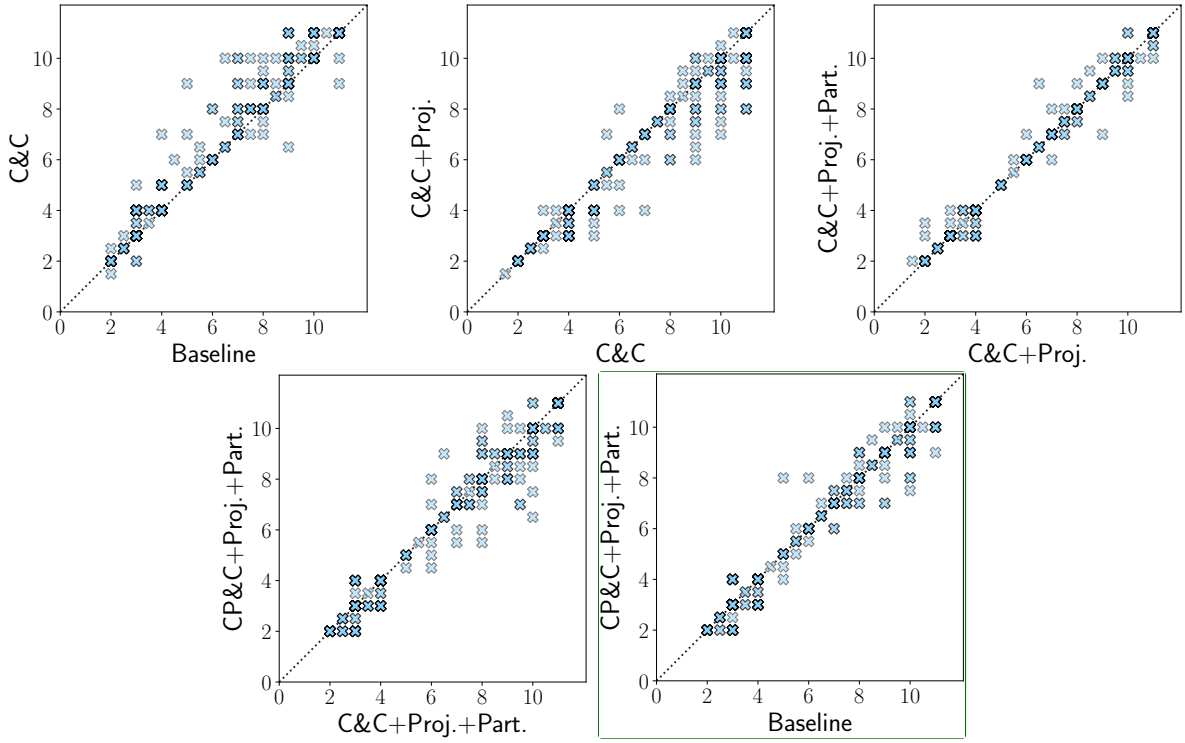


Figure 7.7: Median $\mathcal{T}$-lemma size for different strategies on synthetic benchmarks.

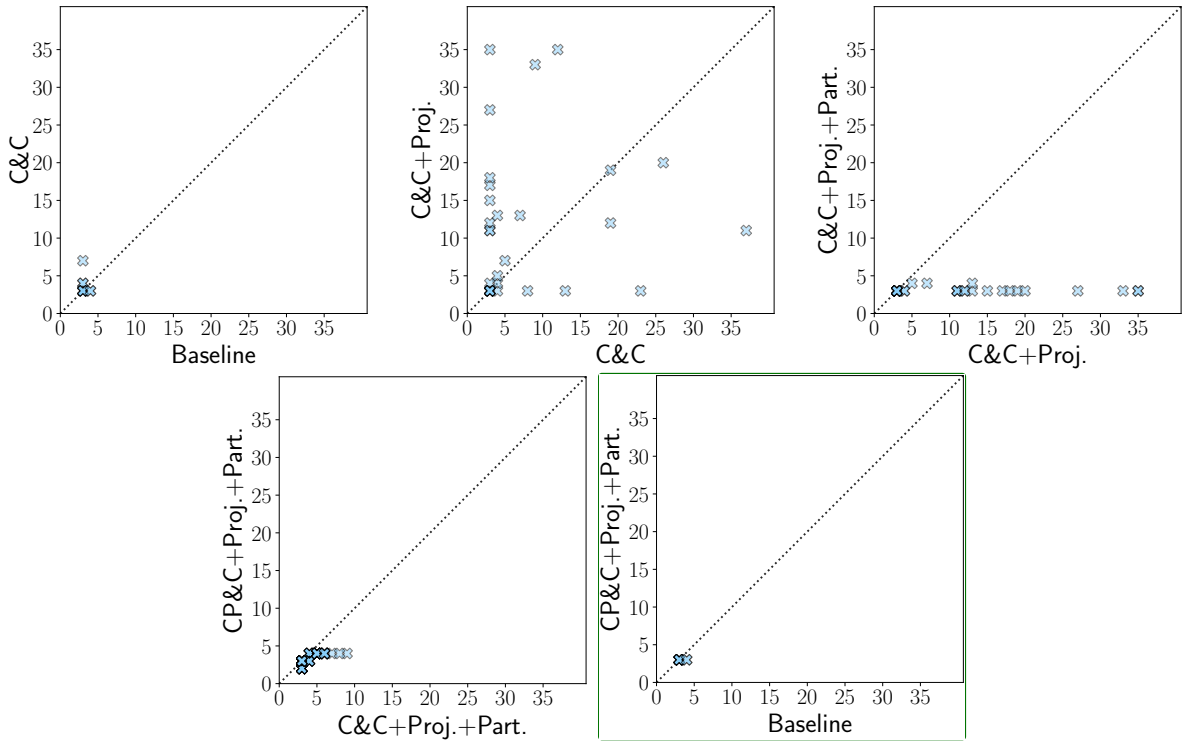


Figure 7.8: Median $\mathcal{T}$-lemma size for different strategies on temporal planning benchmarks.

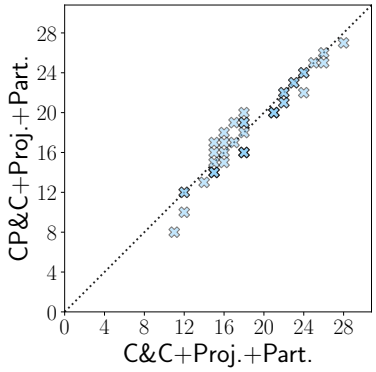


Figure 7.9: Median $\mathcal{T}$-lemma size for different strategies on numeric planning benchmarks.

7.4 Conclusions

In this chapter, we have presented theory-agnostic techniques for enumerating a set of theory lemmas ruling out all theory-inconsistent truth assignments for a given SMT formula. Starting from a baseline method based on total $\mathcal{T}$-AllSAT enumeration, we proposed three orthogonal, composable techniques to enhance efficiency.

We experimentally evaluated our techniques on synthetic SMT problems and on SMT formulas encoding temporal [224] and numeric planning problems [190]. Our techniques drastically improve the efficiency of lemma enumeration w.r.t. the baseline, allowing us to scale to much more complex instances.

This work plays a crucial role in the development of our novel framework for Knowledge Compilation Modulo Theories, presented in the next chapter. These lemma-enumeration techniques pave the way for scaling to complex, real-world instances.

Chapter 8

d-DNNF Modulo Theories: A General Framework for Polytime SMT Transformations and Queries

> **Full Disclosure.** This chapter is based on the papers "Canonical Decision Diagrams Modulo Theories" [152] and "d-DNNF Modulo Theories: A General Framework for Polytime SMT Queries" [147], co-authored by the Ph.D. candidate with the Master's students Massimo Michelutti and Emanuele Civini, Dr. Giuseppe Spallitta, and Prof. Roberto Sebastiani.
> Massimo Michelutti implemented and evaluated the first version of the framework [152], supporting the compilation of OBDDs, SDDs, and d-DNNFs. Emanuele Civini added query support and performed the experimental evaluation in [147]. The theoretical framework was primarily conceived by Prof. Roberto Sebastiani, with contributions from the candidate. The formalization of the polytime transformations and the additional experiments reported in this chapter were carried out by the candidate and are still unpublished at the time of submission.

Building on the theory-lemma enumeration techniques presented in Chapter 7, in this chapter, we present a general framework for Knowledge Compilation (KC) Modulo Theories.

As discussed in Section 2.7, several existing works have proposed extending subclasses of d-DNNFs, namely, OBDDs and SDDs, to handle SMT formulas. Most of these approaches, however, are restricted to specific and simple theories. Moreover, they are hard to implement, often requiring heavy modifications to the internals of an SMT solver, a KC package, or both. Lastly, while the existing literature heavily focuses on SMT *solving*, propositional KC (and d-DNNFs in particular) supports a wide range of polytime queries. These include the counting and enumeration of truth assignments, making them powerful tools for probabilistic reasoning and inference [59, 44, 226].

Contributions. In this chapter, we investigate for the first time the problem of lifting d-DNNF compilation and querying to the SMT level, with the general goal of preserving the properties of the corresponding propositional d-DNNFs, in particular the polytime complexity of the supported transformations and queries (see Section 1.7.3).

We first note that the traditional "lazy" lemmas-on-demand approach, which is widely adopted in standard SMT solving and enumeration (see, e.g., [129, 9]), does not seem to be suitable for SMT-level d-DNNF compilation. We then analyze the problem theoretically in detail and introduce a novel, general framework that extends d-DNNFs to the realm of SMT. We identify and prove the theoretical properties that allow us to meet our general goal above.

Intuitively, before d-DNNF compilation, the input $\mathcal{T}$-formula can be "eagerly" combined with an ad hoc list of precomputed theory lemmas, so that the transformations and queries at the SMT level reduce to those at the propositional level. Overall, our approach has several features:

- it works for every theory or theory combination thereof;
- it works for all forms of d-DNNFs (including OBDDs and SDDs);
- it is easy to implement on top of any d-DNNF compiler and any theory-lemma enumerator, which are used as black boxes;
- most importantly, these compiled SMT d-DNNFs can be queried in polytime by means of a standard propositional d-DNNF reasoner.

Notice that the "eager" generation of all the needed theory lemmas upfront, although inefficient and obsolete in standard SMT solving, is very suitable for KC, whose rationale is to shift the bulk of computational effort to the offline compilation phase, thereby optimizing the efficiency of the online query-answering phase. To this end, we move all the theory-reasoning effort to the compilation phase, by means of offline theory-lemma enumeration. (We recall that for many relevant theories (e.g. $\mathcal{LIA}$, $\mathcal{AR}$, $\mathcal{BV}$) even the satisfiability of the conjunctive fragment is NP-hard [9, 192].)

We have implemented an SMT-level d-DNNF compilation and querying tool on top of state-of-the-art d-DNNF packages and our novel theory-agnostic lemma enumerator presented in Chapter 7. An empirical evaluation supports the feasibility and effectiveness of the approach.

Organization. In Section 8.1 we analyze the challenges of lifting d-DNNFs and their properties to the SMT level. In Section 8.2, we present a formal framework for reducing several SMT-level transformations and queries to propositional queries, when the input formula is augmented with a suitable set of $\mathcal{T}$-lemmas. In Section 8.3 we show how applying this framework to d-DNNFs allows us to support a wide range of transformations and queries in polytime. In Section 8.4, we discuss how to effectively compile, transform, and query d-DNNF $\mathcal{T}$-formulas in practice. In Section 8.5, we present empirical results on the effectiveness of our approach. In Section 8.6, we discuss the limitations and outline potential applications and future research directions. Finally, in Section 8.7, we draw our conclusions and outline future research directions.

8.1 d-DNNF Modulo Theories: An Analysis of the Problem

Our general goal is to compile every $\mathcal{T}$-formula φ into a $\mathcal{T}$-equivalent d-DNNF $\mathcal{T}$-formula such that the following checks can be performed in polytime by means of their corresponding propositional checks: *$\mathcal{T}$-consistency* (CO), *$\mathcal{T}$-validity* (VA), *clausal $\mathcal{T}$-entailment* (CE), *$\mathcal{T}$-implicant* (IM), *$\mathcal{T}$-consistent assignment counting* (CT) and *$\mathcal{T}$-consistent assignment enumeration* (ME).

Also, we want *$\mathcal{T}$-equivalence* (EQ) and *sentential $\mathcal{T}$-entailment* (SE) checks to be polytime for those forms of d-DNNF $\mathcal{T}$-formulas such that EQ and SE are polytime for their propositional counterparts (e.g., SDDs and OBDDs); we also wish these forms to be $\mathcal{T}$-canonical (see Section 1.7.3).

Remark 8.1.1. Here CT and ME refer respectively to counting and enumerating all $\mathcal{T}$-consistent total truth assignments on a given atom set $\boldsymbol{\alpha} \supseteq \mathit{Atoms}(\varphi)$ that propositionally satisfy a $\mathcal{T}$-formula φ, and we also refer to them as "$\mathcal{T}$-#SAT" and "$\mathcal{T}$-AllSAT" (also "#SAT" and "AllSAT" for the propositional case).[a]

We stress the fact that, regardless of the d-DNNF format, CT and ME for $\mathcal{T}$-formulas are much trickier than their propositional counterparts. In fact, in the propositional case, producing a *partial* assignment μ such that $\mu \models_{\mathbb{B}} \varphi$ prevents from producing all the 2^k total assignments η_i extending μ, where k is the number of unassigned atoms in μ.

This is not the case with $\mathcal{T}$-formulas, where producing a *partial* $\mathcal{T}$-consistent assignment μ such that $\mu \models_{\mathbb{B}} \varphi$ does *not* prevent from producing all the 2^k total assignments η_i extending μ, *because not all of them are necessarily $\mathcal{T}$-consistent*. Therefore, being able to reduce CT and ME to their propositional counterparts may be very valuable.

[a] As discussed also in Section 1.3, we prefer "$\mathcal{T}$-#SAT" and "$\mathcal{T}$-AllSAT" to "#SMT" and "AllSMT", respectively, to avoid confusion, because we are counting or enumerating $\mathcal{T}$-satisfiable truth assignments that propositionally satisfy the formula rather than theory-specific FOL models. In particular, "#SMT" or "model counting modulo theories" and, more in general, "extension of model counting to SMT" have been proposed with very different meanings in the literature (e.g., [141, 175, 43, 161, 68, 200, 199]).

Moreover, for those subclasses whose propositional counterparts support polytime transformations (see Section 1.7.2), we want to perform *conditioning* (CD), *negation* ($\neg$C), *forgetting* (SFO), *conjunction* ($\wedge$BC), and *disjunction* ($\vee$BC) in polytime via the corresponding propositional transformations. The resulting d-DNNF $\mathcal{T}$-formula should still support the above-mentioned polytime queries. Importantly, the whole process should be *theory-agnostic*.

Our idea is thus to move all the theory-reasoning effort to the compilation phase, and encode all the necessary theory-related information directly into the d-DNNF $\mathcal{T}$-formulas, so that the latter can be queried and transformed by means of standard polytime propositional queries and transformations.

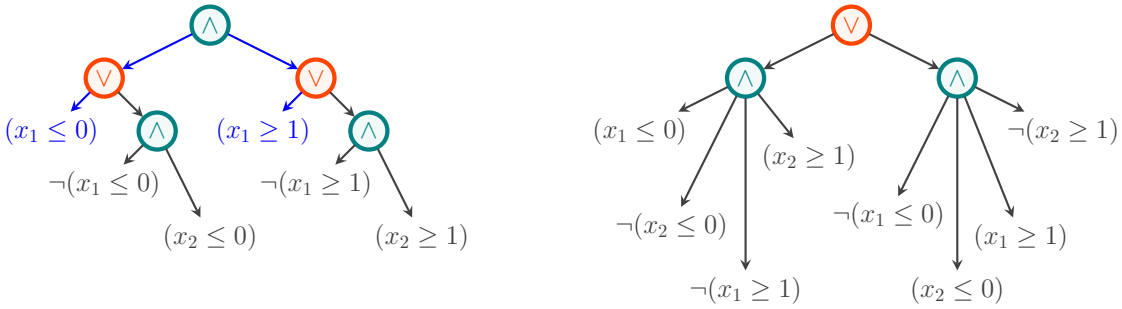


Figure 8.1: d-DNNF $\mathcal{T}$-formula with $\mathcal{T}$-inconsistent satisfying truth assignment (blue paths) from Example 8.1.1 (left), and a $\mathcal{T}$-equivalent d-DNNF $\mathcal{T}$-formula with no $\mathcal{T}$-inconsistent satisfying truth assignment (right).

Notice that, in the propositional case, there is a form of duality between the queries CO, CE, CT, and ME, which search for *satisfying* truth assignments, and the queries VA and IM, which search for *falsifying* ones. This duality suggests that we need two distinct encodings for the two groups of queries. (For EQ and SE, this works either way, because $\varphi_1 \equiv \varphi_2$ iff $\neg\varphi_1 \equiv \neg\varphi_2$ and $\varphi_1 \models \varphi_2$ iff $\neg\varphi_2 \models \neg\varphi_1$.)

We start from the observation that, if we encoded a $\mathcal{T}$-formula φ simply into $\mathsf{d\text{-}DNNF}(\varphi)$, then the latter could be propositionally satisfied [resp. falsified] by many $\mathcal{T}$-inconsistent truth assignments, which would prevent CO, CE, CT, ME, EQ, and SE [resp. VA, IM, EQ, and SE] queries from being correctly implemented by means of their propositional counterparts.

Therefore, we must get rid of such truth assignments in the two respective cases. Our idea is to do this in the compilation phase, by adding proper $\mathcal{T}$-lemmas ruling out these assignments.

Unfortunately, the traditional "lazy" lemmas-on-demand approach, which is widely adopted in standard SMT tools (see, e.g., [9, 129]), does not seem to be applicable here. Consider the case in which we want to encode φ into a $\mathcal{T}$-equivalent d-DNNF $\mathcal{T}$-formula with no satisfying $\mathcal{T}$-inconsistent truth assignments, so as to allow polytime CO, CE, CT, and ME queries.

Consider Algorithm 4 (or its more sophisticated variants). A natural way to implement a lemma-on-demand SMT extension of this procedure and its variants would be to apply a $\mathcal{T}$-consistency check on the current truth assignment μ as soon as a new $\mathcal{T}$-literal is added to it, and to backtrack when this check fails, adding a $\mathcal{T}$-lemma ruling out μ.

The problem with this approach would be in the partitioning step which generates the $\wedge$-nodes (lines 10-13 in Algorithm 4). In fact, even if μ is $\mathcal{T}$-consistent and $\varphi_1|_\mu, \ldots, \varphi_k|_\mu$ do not share atoms, the $\varphi_i|_\mu$'s cannot be safely d-DNNF-ized independently, because their $\mathcal{T}$-atoms may be implicitly linked by some $\mathcal{T}$-lemma, so that $\bigwedge_{i=1}^{k} \mathsf{dDNNFcompile}(\varphi_i|_\mu, \mu)$ (line 13 in Algorithm 4) could be propositionally satisfied by some $\mathcal{T}$-inconsistent truth assignment.

Therefore, to address this issue, we need to add *a priori* an ad hoc set of $\mathcal{T}$-lemmas ruling out those undesired truth assignments.

Example 8.1.1. Consider the $\mathcal{LRA}$-formula

$$\varphi \stackrel{\text{def}}{=} ((x_1 \leq 0) \vee (x_2 \leq 0)) \wedge ((x_1 \geq 1) \vee (x_2 \geq 1)),$$

so that $\varphi^p \stackrel{\text{def}}{=} (A_1 \vee A_2) \wedge (A_3 \vee A_4)$, and assume we apply to φ the hypothetical SMT variant of Algorithm 4 described above. Since the two clauses do not share any $\mathcal{T}$-atom, the algorithm partitions the two clauses (lines 10-13) and recursively compiles them independently into d-DNNF. Since inside each clause the two $\mathcal{LRA}$-atoms cannot generate conflicting literals, in each independent call no $\mathcal{T}$-inconsistent truth assignment is generated. Therefore, the procedure may return the d-DNNF $\mathcal{T}$-formula shown in Figure 8.1 (left).

As highlighted by the blue paths, we observe that the d-DNNF is propositionally satisfied by some $\mathcal{T}$-inconsistent truth assignment, e.g., $\mu_1 \stackrel{\text{def}}{=} (x_1 \leq 0) \wedge (x_2 \leq 0) \wedge (x_1 \geq 1) \wedge \neg(x_2 \geq 1)$. The problem could have been avoided by adding the following $\mathcal{T}$-lemmas to φ a priori: $\neg(x_1 \leq 0) \vee \neg(x_1 \geq 1)$ and $\neg(x_2 \leq 0) \vee \neg(x_2 \geq 1)$. The resulting d-DNNF could be the one shown in Figure 8.1 (right), which is $\mathcal{T}$-equivalent to the previous one, but has no $\mathcal{T}$-inconsistent satisfying truth assignment.

8.2 Theoretical Framework

In this section, we introduce novel definitions and results used in the chapter. For the sake of compactness, all proofs of the theorems [146] are deferred to Appendix E.2.

We adopt the notation $H_{\boldsymbol{\alpha}}(\varphi)$ and $P_{\boldsymbol{\alpha}}(\varphi)$ from Definition 7.1.1 in Section 7.1.

8.2.1 $\mathcal{T}$-reduced and $\mathcal{T}$-extended $\mathcal{T}$-formulas

Definition 8.2.1 ($\mathcal{T}$-reduced$_\alpha$ $\mathcal{T}$-formula). We say that a $\mathcal{T}$-formula φ is *theory-reduced* for a given superset $\boldsymbol{\alpha}$ of $Atoms(\varphi)$ ($\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$) if and only if $P_{\boldsymbol{\alpha}}(\varphi) = \emptyset$, so that (7.1) reduces to

$$\varphi \equiv_{\mathbb{B}} \bigvee_{\eta \in H_{\boldsymbol{\alpha}}(\varphi)} \eta. \tag{8.1}$$

Notice that the notion of $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ depends on the set $\boldsymbol{\alpha}$: given $Atoms(\varphi) \subseteq \boldsymbol{\alpha} \subset \boldsymbol{\alpha'}$, if φ is $\mathcal{T}$-reduced$_{\boldsymbol{\alpha'}}$ then φ is $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$, but the vice versa does not necessarily hold, as shown in the following example.

Example 8.2.1 ($\mathcal{T}$-reduced$_\alpha$ $\mathcal{T}$-formula). Let $\boldsymbol{\alpha} \stackrel{\text{def}}{=} \{(x < 0)\}$, $\boldsymbol{\alpha'} \stackrel{\text{def}}{=} \{(x < 0), (x > 1)\}$, and let $\varphi \stackrel{\text{def}}{=} (x < 0)$.

Then φ is $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ because its only satisfying assignment $(x < 0)$ on $\boldsymbol{\alpha}$ is $\mathcal{T}$-consistent, whereas φ is not $\mathcal{T}$-reduced$_{\boldsymbol{\alpha'}}$ because it has one satisfying $\mathcal{T}$-inconsistent assignment $(x < 0) \wedge (x > 1)$ on $\boldsymbol{\alpha'}$.

The following result shows how the $\mathcal{T}$-reduction property is preserved or modified by some fundamental transformations on $\mathcal{T}$-formulas.

Theorem 8.2.1. *Let $\varphi[\boldsymbol{\alpha}]$, $\varphi_1[\boldsymbol{\alpha}]$, and $\varphi_2[\boldsymbol{\alpha}]$ be $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ $\mathcal{T}$-formulas, let $\alpha \in \boldsymbol{\alpha}$ be a $\mathcal{T}$-atom, and let μ be a cube on $\boldsymbol{\alpha}$. Then we have the following facts.*

CD $\varphi|_\mu$ *is $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}\setminus Atoms(\mu)}$.*

SFO $\exists\alpha.\varphi$ *is $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}\setminus\{\alpha\}}$.*[a]

$\wedge$BC $\varphi_1 \wedge \varphi_2$ *is $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$.*

$\vee$BC $\varphi_1 \vee \varphi_2$ *is $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$.*

$\neg$C $\neg\varphi$ *is $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$ (see Definition 8.2.2 below).*

[a] With the notation $\exists\alpha.\varphi$, here we mean the refinement of $\exists\alpha^p.\varphi^p$.

The following result shows that, with $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ $\mathcal{T}$-formulas, some fundamental reasoning functionalities can be reduced to their Boolean counterparts.

Theorem 8.2.2. *Let $\varphi[\boldsymbol{\alpha}]$, $\varphi_1[\boldsymbol{\alpha}]$, and $\varphi_2[\boldsymbol{\alpha}]$ be $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ $\mathcal{T}$-formulas. Let C be a clause on $\boldsymbol{\alpha}$. Then we have the following facts.*

CO φ *is $\mathcal{T}$-satisfiable if and only if it is $\mathbb{B}$-satisfiable.*

CE $\varphi \models_{\mathcal{T}} C$ *if and only if* $\varphi \models_{\mathbb{B}} C$.

EQ $\varphi_1 \equiv_{\mathcal{T}} \varphi_2$ *if and only if* $\varphi_1 \equiv_{\mathbb{B}} \varphi_2$.

SE $\varphi_1 \models_{\mathcal{T}} \varphi_2$ *if and only if* $\varphi_1 \models_{\mathbb{B}} \varphi_2$.

CT $\mathcal{T}$-$\#\mathsf{SAT}(\varphi) = \#\mathsf{SAT}(\varphi^p)$.

ME $\mathcal{T}$-$\mathsf{AllSAT}(\varphi)$ *is the refinement of* $\mathsf{AllSAT}(\varphi^p)$.

Definition 8.2.2 ($\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$ $\mathcal{T}$-formula). We say that a $\mathcal{T}$-formula φ is *theory-extended* for a given superset $\boldsymbol{\alpha}$ of $Atoms(\varphi)$ ($\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$) if and only if $\neg\varphi$ is $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ (that is, $P_{\boldsymbol{\alpha}}(\neg\varphi) = \emptyset$).

The notion of $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$ also depends on the set $\boldsymbol{\alpha}$: given $Atoms(\varphi) \subseteq \boldsymbol{\alpha} \subset \boldsymbol{\alpha'}$, if φ is $\mathcal{T}$-extended$_{\boldsymbol{\alpha'}}$ then φ is $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$, but the vice versa does not necessarily hold (e.g., consider $\neg\varphi$ in Example 8.2.1).

The following result shows how the $\mathcal{T}$-extension property is preserved or modified by some fundamental transformations on $\mathcal{T}$-formulas.

Theorem 8.2.3. *Let $\varphi[\boldsymbol{\alpha}]$, $\varphi_1[\boldsymbol{\alpha}]$, and $\varphi_2[\boldsymbol{\alpha}]$ be $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$ $\mathcal{T}$-formulas, let $\alpha \in \boldsymbol{\alpha}$ be a $\mathcal{T}$-atom, and let μ be a cube on $\boldsymbol{\alpha}$. Then we have the following facts.*

CD $\varphi|_\mu$ *is* $\mathcal{T}$-extended$_{\boldsymbol{\alpha} \setminus Atoms(\mu)}$.

SFO $\exists \alpha.\varphi$ *is* $\mathcal{T}$-extended$_{\boldsymbol{\alpha} \setminus \{\alpha\}}$.

$\wedge$BC $\varphi_1 \wedge \varphi_2$ *is* $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$.

$\vee$BC $\varphi_1 \vee \varphi_2$ *is* $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$.

$\neg$C $\neg\varphi$ *is* $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$.

The following results show that, with $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$ $\mathcal{T}$-formulas, some fundamental reasoning functionalities can be reduced to their Boolean counterparts.

Theorem 8.2.4. *Let $\varphi[\boldsymbol{\alpha}]$, $\varphi_1[\boldsymbol{\alpha}]$, and $\varphi_2[\boldsymbol{\alpha}]$ be $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$ $\mathcal{T}$-formulas. Let μ be a cube on $\boldsymbol{\alpha}$. Then we have the following facts.*

VA φ *is $\mathcal{T}$-valid if and only if it is $\mathbb{B}$-valid.*

IM $\mu \models_{\mathcal{T}} \varphi$ *if and only if* $\mu \models_{\mathbb{B}} \varphi$.

EQ $\varphi_1 \equiv_{\mathcal{T}} \varphi_2$ *if and only if* $\varphi_1 \equiv_{\mathbb{B}} \varphi_2$.

SE $\varphi_1 \models_{\mathcal{T}} \varphi_2$ *if and only if* $\varphi_1 \models_{\mathbb{B}} \varphi_2$.

8.2.2 $\mathcal{T}$-reduction and $\mathcal{T}$-extension via $\mathcal{T}$-lemmas

We present a general way to transform an arbitrary $\mathcal{T}$-formula φ into a $\mathcal{T}$-equivalent $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ or $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$ one.

We adopt the definition of "rules out" from Definition 7.1.2 in Chapter 7.

Theorem 8.2.5. *Let φ be a $\mathcal{T}$-formula such that $Atoms(\varphi) \subseteq \boldsymbol{\alpha}$, $Cl_{\boldsymbol{\alpha}}(\varphi) \stackrel{def}{=} \{C_1, \ldots, C_K\}$ be a set of $\mathcal{T}$-lemmas that rules out $P_{\boldsymbol{\alpha}}(\varphi)$, and let*

$$\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi) \stackrel{def}{=} \varphi \wedge \bigwedge_{C_l \in Cl_{\boldsymbol{\alpha}}(\varphi)} C_l. \tag{8.2}$$

Then we have that:

(i) $\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi) \equiv_{\mathcal{T}} \varphi$;

(ii) $\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi) \models_{\mathbb{B}} \varphi$;

(iii) $\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi)$ *is* $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$.

Notice that $\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi)$ makes φ $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ by moving all $\mathcal{T}$-inconsistent assignments ρ_j

from $P_{\boldsymbol{\alpha}}(\varphi)$ to $P_{\boldsymbol{\alpha}}(\neg\varphi)$ while preserving $H_{\boldsymbol{\alpha}}(\varphi)$ and $H_{\boldsymbol{\alpha}}(\neg\varphi)$.

Theorem 8.2.6. *Let φ be a $\mathcal{T}$-formula such that $Atoms(\varphi) \subseteq \boldsymbol{\alpha}$, $Cl_{\boldsymbol{\alpha}}(\neg\varphi) \stackrel{def}{=} \{C_1, \ldots, C_K\}$ be a set of $\mathcal{T}$-lemmas that rules out $P_{\boldsymbol{\alpha}}(\neg\varphi)$, and let*

$$\mathcal{T}\mathsf{ext}_{\boldsymbol{\alpha}}(\varphi) \stackrel{def}{=} \varphi \ \vee \neg\Big(\bigwedge_{C_l \in Cl_{\boldsymbol{\alpha}}(\neg\varphi)} C_l\Big). \tag{8.3}$$

Then we have that:

(i) $\mathcal{T}\mathsf{ext}_{\boldsymbol{\alpha}}(\varphi) \equiv_{\mathcal{T}} \varphi$;

(ii) $\varphi \models_{\mathbb{B}} \mathcal{T}\mathsf{ext}_{\boldsymbol{\alpha}}(\varphi)$;

(iii) $\mathcal{T}\mathsf{ext}_{\boldsymbol{\alpha}}(\varphi)$ *is* $\mathcal{T}$*-extended*$_{\boldsymbol{\alpha}}$.

Notice that $\mathcal{T}\mathsf{ext}_{\boldsymbol{\alpha}}(\varphi)$ makes φ $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$ by moving all $\mathcal{T}$-inconsistent assignments ρ_j from $P_{\boldsymbol{\alpha}}(\neg\varphi)$ to $P_{\boldsymbol{\alpha}}(\varphi)$ while preserving $H_{\boldsymbol{\alpha}}(\varphi)$ and $H_{\boldsymbol{\alpha}}(\neg\varphi)$.

Example 8.2.2. Let $\varphi_1 \stackrel{\text{def}}{=} ((x \leq 0) \vee (x = 1))$ and its Boolean abstraction $\varphi_1^p \stackrel{\text{def}}{=} A_1 \vee A_2$, as in Example 7.1.1. Then $P_{\boldsymbol{\alpha}}(\varphi_1)$=$\{(x \leq 0) \wedge (x = 1)\}$ is ruled out by the $\mathcal{T}$-lemma $(\neg(x \leq 0) \vee \neg(x = 1))$.

Thus, $\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi_1) = ((x \leq 0) \vee (x = 1)) \wedge (\neg(x \leq 0) \vee \neg(x = 1))$.

Example 8.2.3. Let $\boldsymbol{\alpha} \stackrel{\text{def}}{=} \{(x \leq 0), (x \geq 1), (x \geq 2)\}$ and $\varphi \stackrel{\text{def}}{=} (x \leq 0) \vee (x \geq 1)$, so

$$\begin{aligned} P_{\boldsymbol{\alpha}}(\varphi) = \{&(x \leq 0) \wedge \ (x \geq 1) \wedge \ (x \geq 2), (x \leq 0) \wedge (x \geq 1) \wedge \neg(x \geq 2), \\ &(x \leq 0) \wedge \neg(x \geq 1) \wedge \ (x \geq 2)\} \end{aligned}$$

and $Cl_{\boldsymbol{\alpha}}(\varphi) = \{\neg(x \leq 0) \vee \neg(x \geq 1),\ \neg(x \leq 0) \vee \neg(x \geq 2)\}$. Hence,

$$\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi) = \Big((x \leq 0) \vee (x \geq 1)\Big) \wedge \Big(\neg(x \leq 0) \vee \neg(x \geq 1)\Big) \wedge \Big(\neg(x \leq 0) \vee \neg(x \geq 2)\Big).$$

Let $\neg\varphi \stackrel{\text{def}}{=} \neg(x \leq 0) \wedge \neg(x \geq 1)$, so that $P_{\boldsymbol{\alpha}}(\neg\varphi) = \{\neg(x \leq 0) \wedge \neg(x \geq 1) \wedge (x \geq 2)\}$, and $Cl_{\boldsymbol{\alpha}}(\neg\varphi) = \{(x \geq 1) \vee \neg(x \geq 2)\}$. Hence,

$$\begin{aligned} \mathcal{T}\mathsf{ext}_{\boldsymbol{\alpha}}(\varphi) &= \Big((x \leq 0) \vee (x \geq 1)\Big) \vee \Big(\neg(x \geq 1) \wedge (x \geq 2)\Big) \\ &\equiv_{\mathbb{B}} (x \leq 0) \vee (x \geq 1) \vee (x \geq 2). \end{aligned}$$

Notice that $\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi)$ is $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ and $\mathcal{T}\mathsf{ext}_{\boldsymbol{\alpha}}(\varphi)$ is $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$, as expected.

Remark 8.2.1. If φ is $\mathcal{T}$-unsatisfiable, then $\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi)$ is $\mathbb{B}$-unsatisfiable and thus $\mathbb{B}$-equivalent to $\bot$; if φ is $\mathcal{T}$-valid then $\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi)$ is not necessarily $\mathbb{B}$-valid, and hence not necessarily $\mathbb{B}$-equivalent to $\top$; rather, it is $\mathbb{B}$-equivalent to $\bigwedge_{C_l \in Cl_{\boldsymbol{\alpha}}(\top)} C_l$.

Dually, if φ is $\mathcal{T}$-valid then $\mathcal{T}\mathsf{ext}_{\boldsymbol{\alpha}}(\varphi)$ is $\mathbb{B}$-valid and thus $\mathbb{B}$-equivalent to $\top$; if φ is $\mathcal{T}$-unsatisfiable then $\mathcal{T}\mathsf{ext}_{\boldsymbol{\alpha}}(\varphi)$ is not necessarily $\mathbb{B}$-unsatisfiable, and hence not necessarily $\mathbb{B}$-equivalent to $\bot$; rather, it is $\mathbb{B}$-equivalent to $\neg \bigwedge_{C_l \in Cl_{\boldsymbol{\alpha}}(\top)} C_l$.

8.2.3 Dealing with Extra $\mathcal{T}$-Atoms

As we have seen in Chapter 7, $Cl_{\boldsymbol{\alpha}}(\varphi)$ can be computed via $\mathcal{T}$-AllSAT, which reduces to performing iterative calls to an SMT solver (Section 1.3.2). In order to cope with some theories, SMT solvers frequently need to introduce extra $\mathcal{T}$-atoms $\boldsymbol{\beta}$ on the fly and generate additional $\mathcal{T}$-lemmas on both $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, relating the novel atoms $\boldsymbol{\beta}$ with those occurring in the original formula φ [9].

For instance, when the $\mathcal{LRA}$-atom $(\sum_i a_i x_i = b)$ occurs in a $\mathcal{LRA}$-formula, the SMT solver may also introduce the $\mathcal{LRA}$-atoms $(\sum_i a_i x_i \geq b)$ and $(\sum_i a_i x_i \leq b)$ and add some or all of the $\mathcal{LRA}$-lemmas encoding

$$(\sum_i a_i x_i = b) \leftrightarrow ((\sum_i a_i x_i \geq b) \wedge (\sum_i a_i x_i \leq b)).$$

Also, when dealing with theory combination, the SMT solver needs to introduce interface equalities $(x_i = x_j)$ and produce $\mathcal{T}$-lemmas involving them (see, e.g., [29]). Consequently, the clauses in $Cl_{\boldsymbol{\alpha}}(\varphi)$ may contain literals on some such $\mathcal{T}$-atoms.

The following theorem shows that this does not prevent us from obtaining $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ and $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$ formulas on $\boldsymbol{\alpha}$ only by existentially quantifying over the truth values of the extra $\mathcal{T}$-atoms $\boldsymbol{\beta}$. With a little abuse of notation, given a formula φ on $\boldsymbol{\alpha} \cup \boldsymbol{\beta}$, we use the expression "$\exists\boldsymbol{\beta}.\varphi$" as a shortcut for "$\mathcal{B}2\mathcal{T}(\exists\mathbf{B}.\varphi^p)$", with $\mathbf{B}$ being the Boolean abstraction of $\boldsymbol{\beta}$.

Theorem 8.2.7. *Let φ be a $\mathcal{T}$-formula such that $Atoms(\varphi) \subseteq \boldsymbol{\alpha}$, $Cl_{\boldsymbol{\alpha},\boldsymbol{\beta}}(\varphi) \stackrel{def}{=} \{C_1, \ldots, C_K\}$ be a set of $\mathcal{T}$-lemmas on $\boldsymbol{\alpha} \cup \boldsymbol{\beta}$ which rules out $P_{\boldsymbol{\alpha}}(\varphi)$, and let*

$$\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi) \stackrel{def}{=} \varphi \wedge \exists\boldsymbol{\beta}. \bigwedge_{C_l \in Cl_{\boldsymbol{\alpha},\boldsymbol{\beta}}(\varphi)} C_l. \tag{8.4}$$

Then we have that:

(i) $\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi) \equiv_{\mathcal{T}} \varphi$;

(ii) $\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi) \models_{\mathbb{B}} \varphi$;

(iii) $\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi)$ *is* $\mathcal{T}$*-reduced*$_{\boldsymbol{\alpha}}$.

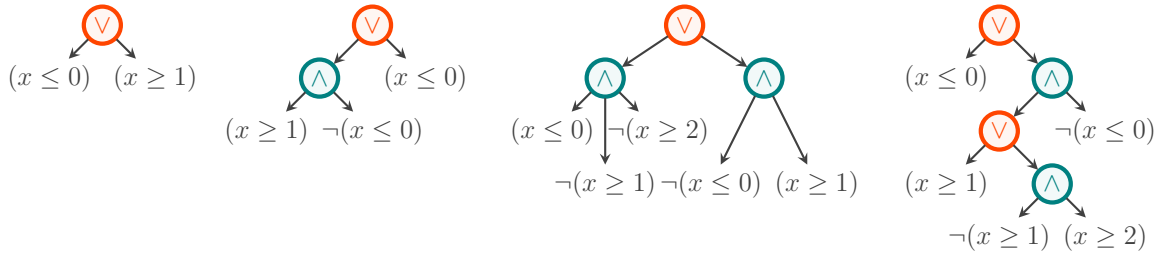


Figure 8.2: NNF representations for formulas in Example 8.2.3. Left: the NNF of φ; Center left: $\mathsf{d\text{-}DNNF}(\varphi)$; Center right: $\mathsf{d\text{-}DNNF}^{\mathsf{red}}_{\boldsymbol{\alpha}}(\varphi)$; Right: $\mathsf{d\text{-}DNNF}^{\mathsf{ext}}_{\boldsymbol{\alpha}}(\varphi)$.

Theorem 8.2.8. *Let φ be a $\mathcal{T}$-formula such that $Atoms(\varphi) \subseteq \boldsymbol{\alpha}$, $Cl_{\boldsymbol{\alpha},\boldsymbol{\beta}}(\neg\varphi) \stackrel{def}{=} \{C_1, \ldots, C_K\}$ be a set of $\mathcal{T}$-lemmas on $\boldsymbol{\alpha} \cup \boldsymbol{\beta}$ which rules out $P_{\boldsymbol{\alpha}}(\neg\varphi)$, and let*

$$\mathcal{T}\mathsf{ext}_{\boldsymbol{\alpha}}(\varphi) \stackrel{def}{=} \varphi \vee \neg \left(\exists \boldsymbol{\beta}. \bigwedge_{C_l \in Cl_{\boldsymbol{\alpha},\boldsymbol{\beta}}(\neg\varphi)} C_l \right). \tag{8.5}$$

Then we have that:

(i) $\mathcal{T}\mathsf{ext}_{\boldsymbol{\alpha}}(\varphi) \equiv_{\mathcal{T}} \varphi$;

(ii) $\varphi \models_{\mathbb{B}} \mathcal{T}\mathsf{ext}_{\boldsymbol{\alpha}}(\varphi)$;

(iii) $\mathcal{T}\mathsf{ext}_{\boldsymbol{\alpha}}(\varphi)$ *is* $\mathcal{T}$*-extended*$_{\boldsymbol{\alpha}}$.

8.3 d-DNNF $\mathcal{T}$-Formulas and Their Polytime Transformations and Queries

We say that a $\mathcal{T}$-formula ψ is in NNF [resp. d-DNNF, sd-DNNF, etc.] iff its Boolean abstraction is in NNF [resp. d-DNNF, sd-DNNF, etc.]. To this end, we extend w.l.o.g. the notion of smoothness to all atoms in $\boldsymbol{\alpha}$, that is, we say that $\varphi[\boldsymbol{\alpha}]$ is smooth$_{\boldsymbol{\alpha}}$ if $Atoms(\varphi) = \boldsymbol{\alpha}$ and φ is smooth. The function $\mathsf{Smooth}()$ is extended accordingly. Figure 8.2 shows some examples of NNF $\mathcal{T}$-formulas.

Since the smoothing operation $\mathsf{Smooth}()$ is $\mathbb{B}$-equivalence preserving (Lemma A.1 in [62]), by Item (i) of Proposition E.2.1 in Appendix E.2, we have the following facts.

Proposition 8.3.1. *Let $\psi[\boldsymbol{\alpha}]$ be a d-DNNF formula.*

(i) If $\psi[\boldsymbol{\alpha}]$ is $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$, then $\mathsf{Smooth}(\psi[\boldsymbol{\alpha}])$ is $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$.

(ii) If $\psi[\boldsymbol{\alpha}]$ is $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$, then $\mathsf{Smooth}(\psi[\boldsymbol{\alpha}])$ is $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$.

We present some results on polytime transformations and queries for d-DNNF $\mathcal{T}$-

formulas. We note that these results can be extended to other NNFs that admit polytime transformations and queries (e.g., d-NNF, DNNF, DNF, CNF, MODS, PI, IP, etc. [62]), though we omit them for simplicity.

8.3.1 Polytime Transformations on d-DNNF $\mathcal{T}$-Formulas by Exploiting $\mathcal{T}$-reduction and $\mathcal{T}$-extension

Theorem 8.2.1 [resp. Theorem 8.2.3] shows that, for a $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ [resp. $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$] $\mathcal{T}$-formula φ, conditioning (CD), forgetting (SFO), conjunction ($\wedge$BC), and disjunction ($\vee$BC) can be reduced to the corresponding polytime transformations on the Boolean abstraction of φ. These transformations preserve or modify the $\mathcal{T}$-reduction [resp. $\mathcal{T}$-extension] properties. This leads to the following results.

Corollary 8.3.1. *Let ψ be a $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ [resp. $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$] d-DNNF $\mathcal{T}$-formula, let μ be a $\mathbb{B}$-satisfiable cube on $\boldsymbol{\alpha}$, and let $\boldsymbol{\alpha'} \stackrel{def}{=} \boldsymbol{\alpha} \setminus Atoms(\mu)$.*

CD *A $\mathcal{T}$-reduced$_{\boldsymbol{\alpha'}}$ [resp. $\mathcal{T}$-extended$_{\boldsymbol{\alpha'}}$] d-DNNF $\mathcal{T}$-formula equivalent to $\psi|_\mu$ can be computed in polytime w.r.t. the size of ψ and μ.*

Corollary 8.3.2. *Let ψ, ψ_1, and ψ_2 be $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ [resp. $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$] d-DNNF $\mathcal{T}$-formulas. Let $\alpha \in \boldsymbol{\alpha}$ and $\boldsymbol{\alpha'} \stackrel{def}{=} \boldsymbol{\alpha} \setminus \{\alpha\}$.*

$\neg$C *If ψ is in an NNF form $\mathcal{L}$ for which a formula in $\mathcal{L}$ that is $\mathbb{B}$-equivalent to $\neg\psi$ can be computed in polytime (e.g., OBDDs and SDDs), then a $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$[resp. $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$] formula in $\mathcal{L}$ that is $\mathcal{T}$-equivalent to $\neg\psi$ can be computed in polytime.*

SFO *If ψ is in an NNF form $\mathcal{L}$ for which a formula in $\mathcal{L}$ that is $\mathbb{B}$-equivalent to $\exists\alpha.\psi$[a] can be computed in polytime (e.g., OBDDs), then a $\mathcal{T}$-reduced$_{\boldsymbol{\alpha'}}$[resp. $\mathcal{T}$-extended$_{\boldsymbol{\alpha'}}$] formula in $\mathcal{L}$ that is $\mathcal{T}$-equivalent to $\exists\alpha.\psi$ can be computed in polytime.*

$\wedge$BC *If ψ_1 and ψ_2 are in an NNF form $\mathcal{L}$ for which a formula in $\mathcal{L}$ that is $\mathbb{B}$-equivalent to $\psi_1 \wedge \psi_2$ can be computed in polytime (e.g., OBDDs), then a $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$[resp. $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$] formula in $\mathcal{L}$ that is $\mathcal{T}$-equivalent to $\psi_1 \wedge \psi_2$ can be computed in polytime.*

$\vee$BC *If ψ_1 and ψ_2 are in an NNF form $\mathcal{L}$ for which a formula in $\mathcal{L}$ that is $\mathbb{B}$-equivalent to $\psi_1 \vee \psi_2$ can be computed in polytime (e.g., OBDDs), then a $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$[resp. $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$] formula in $\mathcal{L}$ that is $\mathcal{T}$-equivalent to $\psi_1 \vee \psi_2$ can be computed in polytime.*

[a] With the notation $\exists\alpha.\psi$, here we mean the refinement of $\exists\alpha^p.\psi^p$.

Notice that, by Definition 8.2.2, negation "flips" $\mathcal{T}$-reduction and $\mathcal{T}$-extension.

8.3.2 Polytime Queries on d-DNNF $\mathcal{T}$-Formulas by Exploiting $\mathcal{T}$-reduction

Theorem 8.2.2 shows that, for a $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ $\mathcal{T}$-formula φ, $\mathcal{T}$-satisfiability (CO), clausal $\mathcal{T}$-entailment (CE), $\mathcal{T}$-consistent assignment counting (CT), $\mathcal{T}$-consistent assignment enumeration (ME), $\mathcal{T}$-equivalence (EQ), and $\mathcal{T}$-entailment (SE), reduce to the (computationally much cheaper) Boolean case, regardless of the theory $\mathcal{T}$considered. This leads to the following results.

Theorem 8.3.1. *Let ψ be a $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ d-DNNF $\mathcal{T}$-formula and let C be a clause on $\boldsymbol{\alpha}$.*

CO *The $\mathcal{T}$-satisfiability of ψ can be checked in polytime w.r.t. the size of ψ.*

CE *$\psi \models_{\mathcal{T}} C$ can be computed in polytime w.r.t. the size of ψ and C.*

CT *$\mathcal{T}$-$\#\mathsf{SAT}(\psi)$ can be computed in polytime w.r.t. the size of ψ.*

ME *$\mathcal{T}$-$\mathsf{AllSAT}(\psi)$ can be computed in polytime w.r.t. the size of ψ and the size of the output set.*

Theorem 8.3.2. *Let ψ_1 and ψ_2 be $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ d-DNNF $\mathcal{T}$-formulas.*

EQ *If ψ_1 and ψ_2 are in a form for which $\psi_1 \equiv_{\mathbb{B}} \psi_2$ can be checked in polytime (e.g., OBDDs and SDDs), then $\psi_1 \equiv_{\mathcal{T}} \psi_2$ can be computed in polytime.*

SE *If ψ_1 and ψ_2 are in a form for which $\psi_1 \models_{\mathbb{B}} \psi_2$ can be checked in polytime (e.g., OBDDs and SDDs), then $\psi_1 \models_{\mathcal{T}} \psi_2$ can be computed in polytime.*

8.3.3 Polytime Queries on d-DNNF $\mathcal{T}$-Formulas by Exploiting $\mathcal{T}$-extension

Theorem 8.2.4 shows that, for a $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$ $\mathcal{T}$-formula φ, $\mathcal{T}$-validity (VA), $\mathcal{T}$-implicant check (IM), $\mathcal{T}$-equivalence (EQ), and $\mathcal{T}$-entailment (SE), reduce to the (computationally much cheaper) Boolean case, regardless of the theory $\mathcal{T}$considered. This leads to the following results.

Theorem 8.3.3. *Let ψ be a $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$ d-DNNF $\mathcal{T}$-formula and let μ be a cube on $\boldsymbol{\alpha}$.*

VA *The $\mathcal{T}$-validity of ψ can be checked in polytime w.r.t. the size of ψ.*

IM *$\mu \models_{\mathcal{T}} \psi$ can be computed in polytime w.r.t. the size of ψ and μ.*

Theorem 8.3.4. *Let ψ_1 and ψ_2 be $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$ d-DNNF $\mathcal{T}$-formulas.*

EQ *If ψ_1 and ψ_2 are in a form for which $\psi_1 \equiv_{\mathbb{B}} \psi_2$ can be checked in polytime (e.g., OBDDs and SDDs), then $\psi_1 \equiv_{\mathcal{T}} \psi_2$ can be computed in polytime.*

SE *If ψ_1 and ψ_2 are in a form for which $\psi_1 \models_{\mathbb{B}} \psi_2$ can be checked in polytime (e.g., OBDDs and SDDs), then $\psi_1 \models_{\mathcal{T}} \psi_2$ can be computed in polytime.*

8.3.4 $\mathcal{T}$-Canonicity of $\mathcal{T}$-reduced and $\mathcal{T}$-extended d-DNNF

The following results follow from Definitions 8.2.1 and 8.2.2.

Theorem 8.3.5 ($\mathcal{T}$-canonicity of $\mathcal{T}$-reduced). *Let φ and ψ be two $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ $\mathcal{T}$-formulas in a canonical form of d-DNNF w.r.t. some canonicity condition (e.g., OBDDs with a fixed atom ordering, or SDDs with a fixed vtree, see Section 1.7.1). Then, $\varphi \equiv_{\mathcal{T}} \psi$ iff $\varphi = \psi$.*

Theorem 8.3.6 ($\mathcal{T}$-canonicity of $\mathcal{T}$-extended). *Let φ and ψ be two $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$ $\mathcal{T}$-formulas in a canonical form of d-DNNF w.r.t. some canonicity condition (e.g., OBDDs with a fixed atom ordering or SDDs with a fixed vtree, see Section 1.7.1). Then $\varphi \equiv_{\mathcal{T}} \psi$ iff $\varphi = \psi$.*

Remark 8.3.1. We recall that, by Remark 8.2.1: $\mathcal{T}$-unsatisfiable $\mathcal{T}$-reduced OBDDs [resp. SDDs] are equal to the $\bot$ node. $\mathcal{T}$-valid $\mathcal{T}$-extended OBDDs [resp. SDDs] are equal to the $\top$ node. $\mathcal{T}$-valid canonical $\mathcal{T}$-reduced OBDDs [resp. SDDs] are equal to the OBDD [resp. SDD] of the disjunction of all $\mathcal{T}$-satisfiable total assignments on $\boldsymbol{\alpha}$. $\mathcal{T}$-unsatisfiable canonical $\mathcal{T}$-extended OBDDs [resp. SDDs] are equal to the OBDD [resp. SDD] of the disjunction of all $\mathcal{T}$-unsatisfiable total assignments on $\boldsymbol{\alpha}$.

8.4 Compiling, Transforming, and Querying d-DNNF $\mathcal{T}$-Formulas

We now show how the results in Section 8.2 translate in practice for compiling $\mathcal{T}$-formulas into d-DNNF and performing polytime transformations and queries.

8.4.1 Compilation

We denote by *d-DNNF an arbitrary d-DNNF form from [62, 60] (e.g., d-DNNF, sd-DNNF, OBDD, and SDD), and by *$\mathsf{d\text{-}DNNF}(.)$ a corresponding encoding function

Algorithm 15 $\mathcal{T}^{\mathsf{red}}_{\boldsymbol{\alpha}}$-$*$dDNNFcompile($\varphi[\boldsymbol{\alpha}]$)
Input: $\varphi[\boldsymbol{\alpha}]$: a $\mathcal{T}$-formula over atoms $\boldsymbol{\alpha}$;
Output: *d-DNNF$^{\mathsf{red}}_{\boldsymbol{\alpha}}(\varphi)$

1: $\{C_1, \ldots, C_K\} \leftarrow$ LemmaEnumerator(φ)
2: $\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi) \leftarrow \varphi \wedge \left(\bigwedge_{l=1}^{K} C_l\right)$ {see (8.2)}
3: $\psi^p \leftarrow *$dDNNFcompile($\mathcal{T}2\mathcal{B}(\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi))$)
4: **return** $\mathcal{B}2\mathcal{T}(\psi^p)$

Algorithm 16 $\mathcal{T}^{\mathsf{ext}}_{\boldsymbol{\alpha}}$-$*$dDNNFcompile($\varphi[\boldsymbol{\alpha}]$)
Input: $\varphi[\boldsymbol{\alpha}]$: a $\mathcal{T}$-formula over atoms $\boldsymbol{\alpha}$;
Output: *d-DNNF$^{\mathsf{ext}}_{\boldsymbol{\alpha}}(\varphi)$

1: $\{C_1, \ldots, C_K\} \leftarrow$ LemmaEnumerator($\neg\varphi$)
2: $\mathcal{T}\mathsf{ext}_{\boldsymbol{\alpha}}(\varphi) \leftarrow \varphi \vee \neg\left(\bigwedge_{l=1}^{K} C_l\right)$ {see (8.3)}
3: $\psi^p \leftarrow *$dDNNFcompile($\mathcal{T}2\mathcal{B}(\mathcal{T}\mathsf{ext}_{\boldsymbol{\alpha}}(\varphi))$)
4: **return** $\mathcal{B}2\mathcal{T}(\psi^p)$

into *d-DNNF form. Then, for each NNF form *d-DNNF, we define:

$$^*\mathsf{d\text{-}DNNF}^{\mathsf{red}}_{\boldsymbol{\alpha}}(\varphi) \stackrel{\text{def}}{=} {}^*\mathsf{d\text{-}DNNF}(\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi)); \tag{8.6}$$

$$^*\mathsf{d\text{-}DNNF}^{\mathsf{ext}}_{\boldsymbol{\alpha}}(\varphi) \stackrel{\text{def}}{=} {}^*\mathsf{d\text{-}DNNF}(\mathcal{T}\mathsf{ext}_{\boldsymbol{\alpha}}(\varphi)). \tag{8.7}$$

For instance, $\mathsf{d\text{-}DNNF}^{\mathsf{red}}_{\boldsymbol{\alpha}}(\varphi) \stackrel{\text{def}}{=} \mathsf{d\text{-}DNNF}(\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi))$ and $\mathsf{SDD}^{\mathsf{ext}}_{\boldsymbol{\alpha}}(\varphi) \stackrel{\text{def}}{=} \mathsf{SDD}(\mathcal{T}\mathsf{ext}_{\boldsymbol{\alpha}}(\varphi))$.

Since each encoding function *d-DNNF(.) is $\mathbb{B}$-equivalence preserving, by Theorems 8.2.5 and 8.2.6, all $^*\mathsf{d\text{-}DNNF}^{\mathsf{red}}_{\boldsymbol{\alpha}}(\varphi)$'s are $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ and all $^*\mathsf{d\text{-}DNNF}^{\mathsf{ext}}_{\boldsymbol{\alpha}}(\varphi)$'s are $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$. Figure 8.2 (center right and right) shows examples of d-DNNF $\mathcal{T}$-formulas from Example 8.2.3, obtained by applying the above definitions.

We can compute $^*\mathsf{d\text{-}DNNF}^{\mathsf{red}}_{\boldsymbol{\alpha}}(\varphi)$ [resp. $^*\mathsf{d\text{-}DNNF}^{\mathsf{ext}}_{\boldsymbol{\alpha}}(\varphi)$] as shown in Algorithm 15 [resp. Algorithm 16]. For simplicity, we assume that no extra $\mathcal{T}$-atoms are introduced by LemmaEnumerator, and discuss this case in the next paragraph.

The algorithm takes as input a $\mathcal{T}$-formula φ and uses a theory-lemma enumerator LemmaEnumerator (see Chapter 7) and a Boolean *d-DNNF compiler $*$dDNNFcompile (e.g., Algorithm 4 or more advanced compilers [202, 126]). It first computes the set $\{C_1, \ldots, C_K\}$ of theory lemmas by calling LemmaEnumerator on φ [resp. on $\neg\varphi$]. Then, it builds $\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi)$ [resp. $\mathcal{T}\mathsf{ext}_{\boldsymbol{\alpha}}(\varphi)$] as in (8.2) [resp. (8.3)]. Finally, it compiles the Boolean abstraction of the resulting formula into *d-DNNF and returns its refinement.

Dealing with extra $\mathcal{T}$-atoms. As discussed in Section 8.2.3, the enumerated $\mathcal{T}$-lemmas may contain atoms not in $\boldsymbol{\alpha}$, so that, by Theorems 8.2.7 and 8.2.8, to compile a $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ [resp. $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$] $\mathcal{T}$-formula we need to existentially quantify them away. Some d-DNNF compilers, such as [126], support projected compilation, which performs this quantification during compilation; OBDD and SDD packages, such as [202], instead support quantification after compilation.

8.4.2 Transformations

Combining the results of Section 8.4.1 with Corollaries 8.3.1 and 8.3.2 we have the following facts:

- every $^*\mathsf{d\text{-}DNNF}^{\mathsf{red}}_{\boldsymbol{\alpha}}(\varphi)$ and every $^*\mathsf{d\text{-}DNNF}^{\mathsf{ext}}_{\boldsymbol{\alpha}}(\varphi)$ satisfies the hypotheses of Corollary 8.3.1, so conditioning (CD) can be computed in polytime;
- every $^*\mathsf{d\text{-}DNNF}^{\mathsf{red}}_{\boldsymbol{\alpha}}(\varphi)$ and every $^*\mathsf{d\text{-}DNNF}^{\mathsf{ext}}_{\boldsymbol{\alpha}}(\varphi)$ for which *d-DNNF is either OBDD or SDD satisfies the hypotheses of Corollary 8.3.2, so negation ($\neg$C) can be computed in polytime;
- every $^*\mathsf{d\text{-}DNNF}^{\mathsf{red}}_{\boldsymbol{\alpha}}(\varphi)$ and every $^*\mathsf{d\text{-}DNNF}^{\mathsf{ext}}_{\boldsymbol{\alpha}}(\varphi)$ for which *d-DNNF is OBDD satisfies the hypotheses of Corollary 8.3.2, so forgetting (SFO), conjunction ($\wedge$BC), and disjunction ($\vee$BC) can be computed in polytime.

Algorithmically, we can compute these transformations by means of a Boolean *d-DNNF package, feeding it with the Boolean abstraction of the compiled $\mathcal{T}$-formulas.

8.4.3 Queries

Combining the results of Section 8.4.1 with Theorems 8.3.1 and 8.3.3 we have the following facts:

- by Theorem 8.3.1, every $^*\mathsf{d\text{-}DNNF}^{\mathsf{red}}_{\boldsymbol{\alpha}}(\varphi)$ supports polytime $\mathcal{T}$-satisfiability (CO), clausal $\mathcal{T}$-entailment (CE), $\mathcal{T}$-#SAT (CT), and $\mathcal{T}$-AllSAT (ME);
- every $^*\mathsf{d\text{-}DNNF}^{\mathsf{ext}}_{\boldsymbol{\alpha}}(\varphi)$ satisfies the hypotheses of Theorem 8.3.3, so that $\mathcal{T}$-validity (VA) and $\mathcal{T}$-implicant check (IM) with these formulas can be computed in polytime;
- every $^*\mathsf{d\text{-}DNNF}^{\mathsf{red}}_{\boldsymbol{\alpha}}(\varphi)$ [resp. $^*\mathsf{d\text{-}DNNF}^{\mathsf{ext}}_{\boldsymbol{\alpha}}(\varphi)$] for which *d-DNNF is one of OBDD and SDD satisfies the hypotheses of Theorem 8.3.2 EQ [resp. Theorem 8.3.4 EQ], so that $\mathcal{T}$-equivalence with these formulas can be computed in polytime;
- every $^*\mathsf{d\text{-}DNNF}^{\mathsf{red}}_{\boldsymbol{\alpha}}(\varphi)$ [resp. $^*\mathsf{d\text{-}DNNF}^{\mathsf{ext}}_{\boldsymbol{\alpha}}(\varphi)$] for which *d-DNNF is one of OBDD and SDD satisfies the hypotheses of Theorem 8.3.2 SE [resp. Theorem 8.3.4 SE], so that $\mathcal{T}$-entailment with these formulas can be computed in polytime.

Algorithmically, we can compute these queries by means of a Boolean *d-DNNF reasoner, feeding it with the Boolean abstraction of the compiled $\mathcal{T}$-formulas and the Boolean abstraction of the query, and then interpreting the result in the theory $\mathcal{T}$.

8.5 Experimental Evaluation

As a proof of concept, we present a preliminary experimental evaluation of the framework described in the previous sections. (We recall that we currently do not have any direct competitor for SMT-level d-DNNF querying.) We focus on $\mathcal{T}$-reduced d-DNNFs, as they support a wide range of queries. We consider both general d-DNNFs [147] and the subclasses of OBDDs and SDDs [152]. We assess the following aspects:

(i) the compilation time, and the contribution of each step to the total time;

(ii) the size of the compiled representation, compared to the size of the input $\mathcal{T}$-formula;

(iii) the effectiveness and efficiency of the compiled representation in answering queries against SMT-based approaches. We focus on CE, and on CT under assumptions, as detailed next.

Baselines. Since there is no direct competitor for SMT-level d-DNNF querying, we compare query-answering performance with SMT-based approaches using the SMT solver MathSAT5 [48].

For CE, given a formula φ and a clause C, we check the $\mathcal{T}$-(un)satisfiability of $\varphi \wedge \neg C$. For multiple queries on the same formula, we run MathSAT incrementally under assumptions.

For CT, to the best of our knowledge, there is no SMT tool specifically designed for counting the number of $\mathcal{T}$-satisfiable truth assignments. This is not surprising because this task is very challenging: *it is not possible to exploit $\mathcal{T}$-consistent partial-assignment enumeration*, as partial assignments *do not* guarantee that *all* their extensions are $\mathcal{T}$-satisfiable (recall Remark 8.1.1). Thus, we are forced to use the $\mathcal{T}$-AllSAT functionality of MathSAT to enumerate all the total $\mathcal{T}$-satisfiable truth assignments, counting and discharging them as they are produced. (We reckon this could be considered as an easy-to-win comparison, but we see no alternative competitor.)

8.5.1 Information for the Reproducibility of the Experiments

We have implemented the framework described in Section 8.4 in a prototype Python tool. The tool[1] and the code to run the experiments[2] are publicly available.

The tool uses PySMT [85] to manipulate $\mathcal{T}$-formulas. For the enumeration of $\mathcal{T}$-lemmas, we use our novel technique presented in Chapter 7, and in particular the C&CP+Proj.+Part. configuration with 50 parallel processes[3] (see Section 7.3).

We use D4[4] [126] for d-DNNF compilation, using the algorithm for non-CNF formulas [67]; we use CUDD [202] via the Python interface of the dd package[5] for OBDD compilation, and PySDD[6] for SDD compilation.

Since OBDDs, SDDs, and d-DNNFs are obtained with different compilers and use different representations, queries are answered using different tools. For OBDDs and SDDs, we use the respective compilers' APIs. For d-DNNFs, we use ddnnife[7] [214], which supports a wide range of queries on propositional d-DNNFs, including CO (possibly under conditioning), ME (possibly under conditioning), and CT (possibly

[1] https://github.com/masinag/tddnnf

[2] https://github.com/masinag/tddnnf-testbench

[3] https://github.com/ecivini/tlemmas-enumeration/tree/develop

[4] https://github.com/crillab/d4v2

[5] https://github.com/tulip-control/dd/

[6] https://github.com/ML-KULeuven/PySDD

[7] https://github.com/SoftVarE-Group/d-dnnf-reasoner

under conditioning and/or projected on a subset of atoms). CE can be reduced to CO under conditioning, and thus it is also supported.

All the experiments were executed on three machines with identical hardware configurations. Each machine was equipped with a 28-core Intel Xeon Gold 6238R CPU at 2.20 GHz and 128 GB of RAM, and ran Ubuntu Linux 24.04. For synthetic benchmarks, we set a timeout of 3600 s for theory-lemma enumeration and an additional 3600 s for Boolean compilation. For the harder numeric planning benchmarks, we set a timeout of 14400 s for theory-lemma enumeration and an additional 14400 s for Boolean compilation. For query answering, we set a timeout of 600 s for each query.

8.5.2 Benchmarks

We use the synthetic and numeric planning benchmarks described in Section 7.3.2.

Synthetic benchmarks. We consider all 450 synthetic non-CNF SMT($\mathcal{LRA}$) instances of Section 7.3.2. For CE queries, we generate 10 random clauses of sizes 1 to 3 for each problem, with literals over atoms occurring in the formula. The second type of queries we consider is CT under assumptions, i.e., given a formula φ and a cube μ, we ask for $\mathcal{T}$-#SAT$(\varphi \wedge \mu)$, where $\boldsymbol{\alpha} = \mathit{Atoms}(\varphi)$. We consider cubes derived from the negated clauses used for CE. This task counts the number of "counterexamples" to clausal entailment, where a counterexample is a truth assignment to the atoms of the formula violating the entailment. Hence, for CT we only consider problems where CE does not hold.

Numeric planning benchmarks. We consider all 60 SMT($\mathcal{LRA}$) instances from the `block-grouping` domain of Section 7.3.2. In this domain, N blocks $\{1, \ldots, N\}$ are initially placed on a 10×10 grid, and at each step each block can be moved to an adjacent cell. Given a partition of the blocks into groups, the goal is to move the blocks so that all blocks in the same group end up in the same grid cell, while blocks in different groups end up in different cells.

Here, the SMT formula encodes a general k-step planning problem, together with an initial position of the blocks on the grid. CE queries represent negated goals: if CE holds, then the goal state is unreachable in k steps. Since CE only supports queries corresponding to "conjunctive goals", we generate goals of the form

$$\bigwedge_{i,j \text{ s.t. } \mathit{Group}(i)=\mathit{Group}(j)} (x_i = x_j \wedge y_i = y_j) \wedge \bigwedge_{i,j \text{ s.t. } \mathit{Group}(i)\neq\mathit{Group}(j)} (x_i \neq x_j \vee y_i \neq y_j),$$

where $\mathit{Group}(i)$ denotes the group of block i, and (x_i, y_i) are the integer coordinates of block i. For CT under assumptions, we use the same goals, and the answer is the number of plans that reach the goal in k steps, given the initial position of the blocks.

Importantly, in this setting, a plan is a sequence of actions, i.e., a truth assignment to Boolean atoms representing whether an action, such as moving a block to the left, is executed at a given step, as well as to atoms encoding the values of the action parameters, such as the block to move. In principle, one should perform projected model

counting on these atoms in order to count the number of such plans. However, as we also empirically verified, this is not necessary: the truth assignment to all other atoms is uniquely determined by the truth assignment to the action and parameter atoms, so the non-projected count is equal to the projected count. Thus, we perform non-projected compilation and counting, since we observed them to be more efficient.

8.5.3 Results on Compilation Time

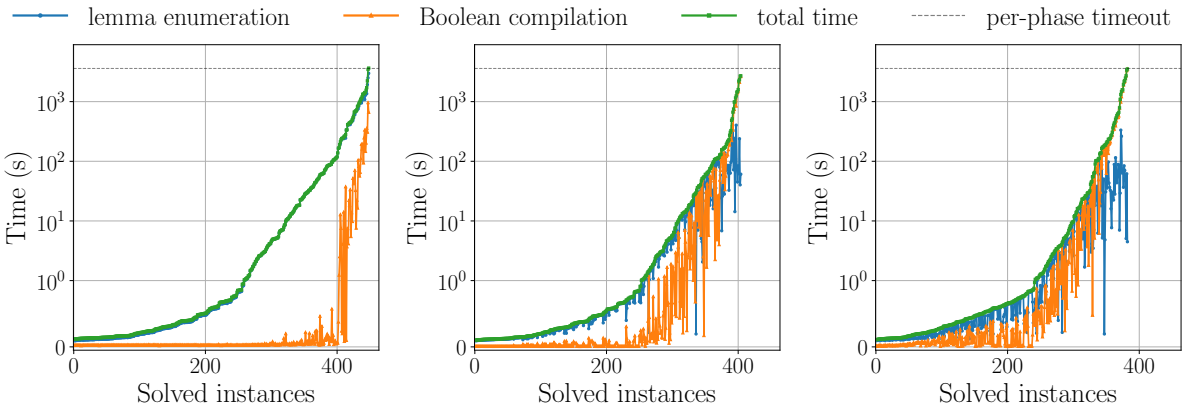
We first analyze the compilation time for $\mathcal{T}$-reduced NNFs. In Figures 8.3 and 8.4, we show results for synthetic and numeric planning benchmarks, respectively. For synthetic benchmarks, we consider $\mathcal{T}$-reduced d-DNNFs, OBDDs, and SDDs, whereas for numeric planning benchmarks we only consider $\mathcal{T}$-reduced d-DNNFs, as OBDDs and SDDs are too expensive to compile. For each NNF type, we show a cactus plot of the total compilation time, as well as the contributions of the two compilation steps in Algorithm 15, i.e., lemma enumeration (line 1) and Boolean compilation (line 3). The table at the bottom reports the number of timeouts for each compilation step.

Synthetic benchmarks. (Figure 8.3) First, we observe that substantially more instances can be compiled into d-DNNF and in significantly less time than into OBDDs or SDDs. This is not surprising since OBDDs and SDDs impose stronger structural restrictions on the compiled representation, and are canonical under a fixed variable order or vtree, which typically makes compilation much more expensive.

Second, for d-DNNFs, the compilation time is dominated by lemma enumeration, whereas the Boolean compilation step also contributes significantly for OBDDs and SDDs, which explains the smaller number of compiled instances. This is expected, since d-DNNF compilation is typically much faster than OBDD and SDD compilation. Also, despite the optimizations discussed in Chapter 7, the enumeration procedure is still expensive, dominating the relatively fast d-DNNF compilation. (The cube-and-conquer lemma-enumeration strategy is highly parallelizable, however, suggesting that the enumeration time can be significantly reduced with more computational resources.) More efficient strategies for conjoining $\mathcal{T}$-lemmas during construction could potentially improve the compilation for OBDDs and SDDs.

Numeric planning benchmarks. (Figure 8.4) These problems are significantly harder than the synthetic ones, and we observe several timeouts also in the Boolean d-DNNF compilation phase. Indeed, in some instances, this phase requires much more time than lemma enumeration. This could be due to the high number of $\mathcal{T}$-lemmas generated for these instances, which can be up to 10^5 (see Figure 7.6 in Section 7.3).

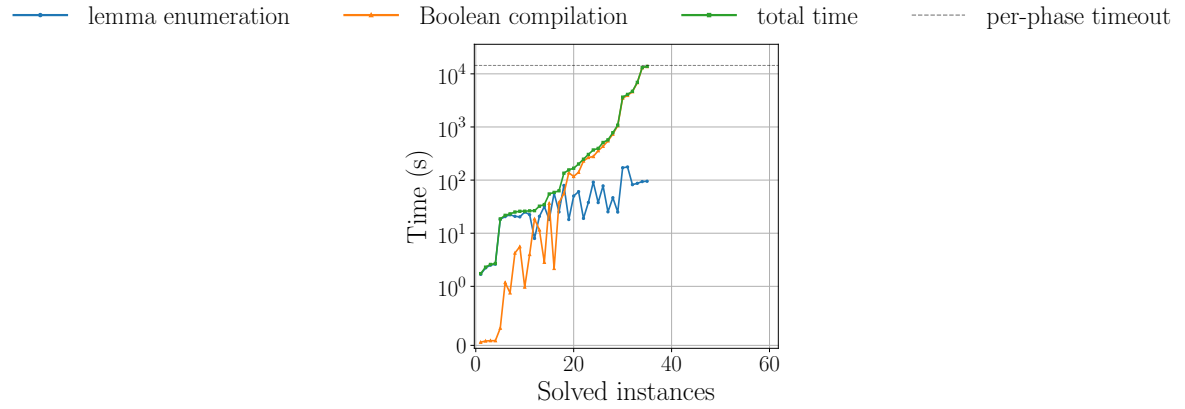
On the positive side, this confirms that the $\mathcal{T}$-lemma enumeration techniques presented in Chapter 7 are very effective.



(a) $\mathsf{d\text{-}DNNF}^{\mathsf{red}}_{\boldsymbol{\alpha}}$ (b) $\mathsf{OBDD}^{\mathsf{red}}_{\boldsymbol{\alpha}}$ (c) $\mathsf{SDD}^{\mathsf{red}}_{\boldsymbol{\alpha}}$

Method	Lemma enum. T.O.	Compilation T.O.	Tot.
$\mathsf{d\text{-}DNNF}^{\mathsf{red}}_{\boldsymbol{\alpha}}$	2	0	450
$\mathsf{OBDD}^{\mathsf{red}}_{\boldsymbol{\alpha}}$	2	44	450
$\mathsf{SDD}^{\mathsf{red}}_{\boldsymbol{\alpha}}$	2	66	450

Figure 8.3: Compilation times for $\mathcal{T}$-reduced NNFs on synthetic benchmarks. Top: cactus plots showing the total compilation time for d-DNNF (left), OBDD (center), and SDD (right). We also show the time for lemma enumeration (blue) and Boolean compilation (orange) for each instance. For d-DNNFs, the green and blue lines overlap. We only show instances compiled within the timeout. Bottom: Number of timeouts for each compilation step.



(a) $\mathcal{T}$-red d-DNNF

Method	Lemma enum. T.O.	Compilation T.O.	Tot.
$\mathsf{d\text{-}DNNF}^{\mathsf{red}}_{\boldsymbol{\alpha}}$	4	21	60

Figure 8.4: Compilation times for $\mathcal{T}$-reduced NNFs on numeric planning benchmarks. Top: a cactus plot showing the total compilation time for d-DNNF. We also show the time for lemma enumeration (blue) and Boolean compilation (orange) for each instance. Bottom: Number of timeouts for each compilation step.

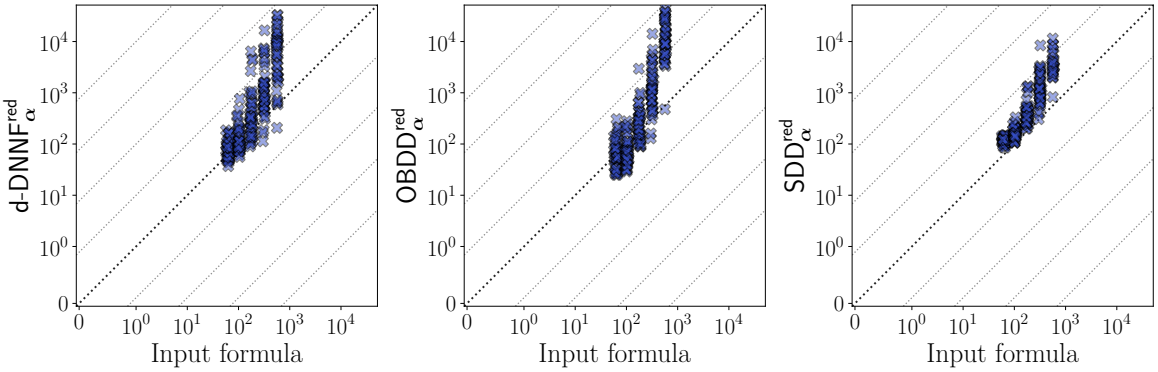


Figure 8.5: Size of the compiled $\mathcal{T}$-reduced NNFs on synthetic benchmarks. We compare the size of d-DNNFs (left), OBDDs (center), and SDDs (right) with the size of the input formulas. Sizes are measured by the number of nodes in their DAG representations.

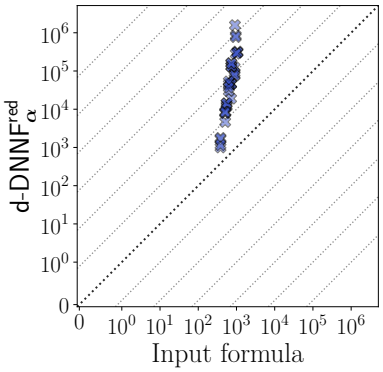


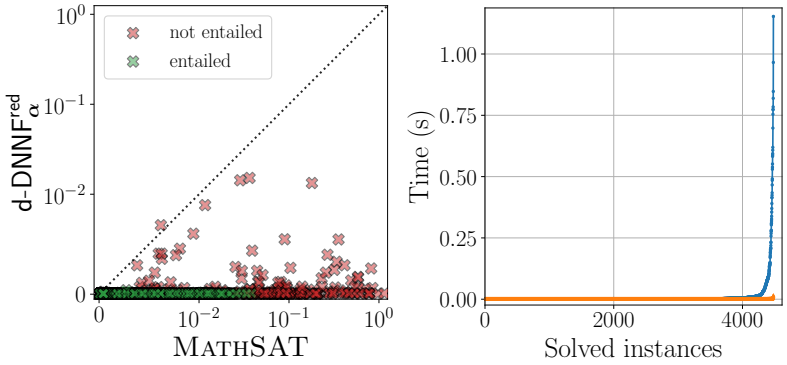
Figure 8.6: Size of the compiled $\mathcal{T}$-reduced NNFs on numeric planning benchmarks. We compare the size of d-DNNFs with the size of the input formulas. Sizes are measured by the number of nodes in their DAG representations.

8.5.4 Results on NNF Size

Next, we analyze the size of the compiled NNFs in Figures 8.5 and 8.6. In each figure, we compare with scatter plots the size of the compiled $\mathcal{T}$-reduced NNFs with the size of the input formulas, both measured as the number of nodes in their DAG representations.

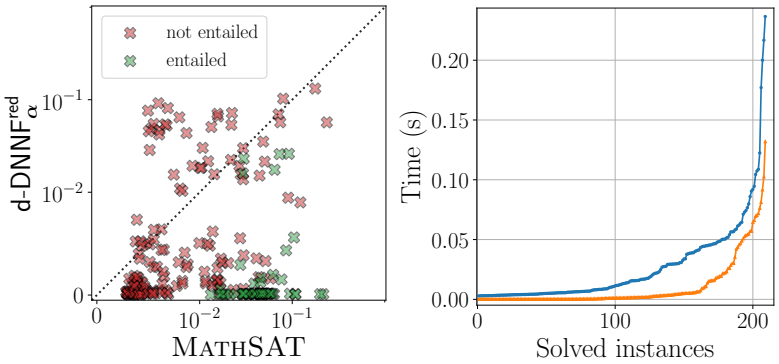
Synthetic benchmarks. (Figure 8.5) We observe that, as one might expect, compiled NNFs are typically (much) larger than the input formulas, for all three types of NNFs. In fact, it is well-known that the size of propositional d-DNNFs, OBDDs, and SDDs can be exponentially larger than the size of the input formula. Nevertheless, the size stays manageable for many instances, with all of them having fewer than 10^5 nodes. As most queries can be answered in time linear in the NNF size, this is a promising result.

Numeric planning benchmarks. (Figure 8.6) Here we observe a similar trend, with the compiled $\mathcal{T}$-reduced d-DNNFs having up to 10^6 nodes, which is still manageable for query answering.



Method	CE	!CE	Tot.
d-DNNF$^{\text{red}}_{\alpha}$	2161	2319	4480
MathSAT	2161	2319	4480

Figure 8.7: Times for CE queries on synthetic problems. Left: scatter plot for $\mathcal{T}$-reduced d-DNNFs vs SMT with MathSAT. Center: comparison with cactus plot. Right: instances where CE does (CE) and does not (!CE) hold, and total instances (Tot.).



Method	CE	!CE	Tot.
d-DNNF$^{\text{red}}_{\alpha}$	63	146	209
MathSAT	63	146	209

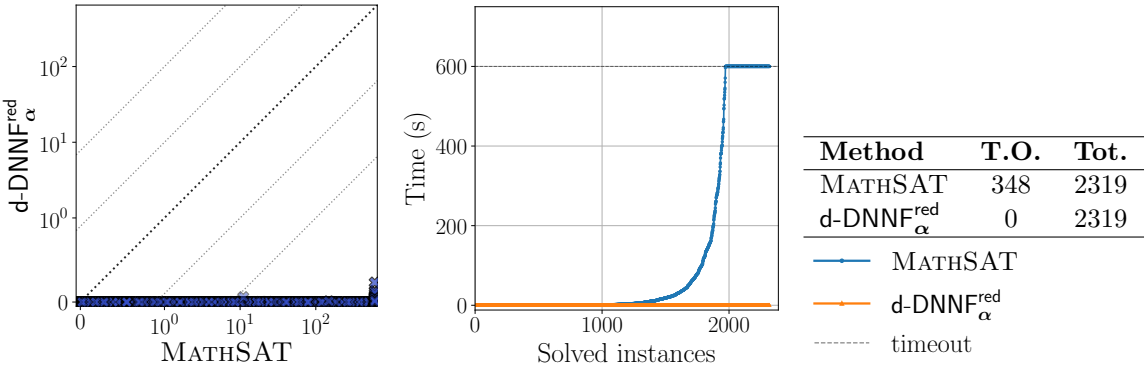
Figure 8.8: Times for CE queries on numeric planning problems. Left: scatter plot for $\mathcal{T}$-red. d-DNNFs vs SMT with MathSAT. Center: comparison with cactus plot. Right: instances where CE does (CE) and does not (!CE) hold, and total instances (Tot.).

8.5.5 Results on CE Queries

We now assess the effectiveness and efficiency of $\mathcal{T}$-reduced NNFs on CE queries, focusing on d-DNNFs, as this is the form for which we can compile the largest number of instances. Figures 8.7 and 8.8 show, for the synthetic and numeric planning problems, respectively, scatter and cactus plots comparing the time for $\mathcal{T}$-reduced d-DNNFs against MathSAT.

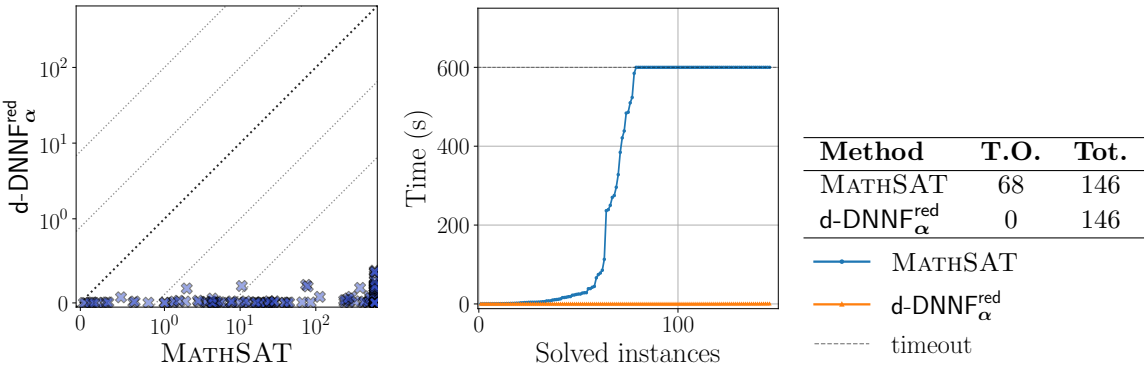
Synthetic benchmarks. (Figure 8.7) We first notice that the problems and queries considered are easy and can be answered by both methods in a fraction of a second. Nevertheless, d-DNNFs are effective and significantly faster than incremental SMT with MathSAT. Note that SMT solvers are highly optimized for checking satisfiability, and are thus expected to perform well on CE queries.

Numeric planning benchmarks. (Figure 8.8) Here we observe a similar trend, with d-DNNFs being significantly faster than MathSAT for most queries. There are, however, some outliers, mostly for satisfiable instances (i.e., where CE does not hold). Note that, in this case as well, the instances are solved in a fraction of a second.



Method	T.O.	Tot.
MathSAT	348	2319
d-DNNF$^{\text{red}}_{\alpha}$	0	2319

Figure 8.9: Times for CT queries on synthetic problems. Left: scatter plot for $\mathcal{T}$-reduced d-DNNF vs $\mathcal{T}$-AllSAT-based counting with MathSAT. Center: comparison with cactus plot. Right: number of timeouts.



Method	T.O.	Tot.
MathSAT	68	146
d-DNNF$^{\text{red}}_{\alpha}$	0	146

Figure 8.10: Times for CT queries on numeric planning problems. Left: scatter plot for $\mathcal{T}$-reduced d-DNNF vs $\mathcal{T}$-AllSAT-based counting with MathSAT. Center: comparison with cactus plot. Right: number of timeouts for each method.

8.5.6 Results on CT Queries

Finally, we evaluate the performance on CT queries. Figures 8.9 and 8.10 show, for the synthetic and numeric planning problems, respectively, scatter and cactus plots comparing $\mathcal{T}$-reduced d-DNNFs with $\mathcal{T}$-AllSAT-based counting using MathSAT.

Synthetic benchmarks. (Figure 8.9) Here, the advantage of $\mathcal{T}$-reduced d-DNNFs over SMT-based approaches is dramatic. While MathSAT fails to solve several queries within the timeout, $\mathcal{T}$-reduced d-DNNFs answer all queries in a fraction of a second. As discussed, $\mathcal{T}$-AllSAT-based CT answering is extremely expensive, whereas d-DNNFs answer these queries in time linear w.r.t. their size. Overall, the results highlight the potential of $\mathcal{T}$-reduced d-DNNFs for queries that are intractable for standard SMT approaches, amortizing the cost of compilation over many expensive queries.

Numeric planning benchmarks. (Figure 8.10) As with the synthetic benchmarks, $\mathcal{T}$-reduced d-DNNFs can answer all queries in a fraction of a second, whereas MathSAT fails to solve almost half of the queries within the timeout.

8.6 Limitations, Research Avenues, and Potential Applications

In this section, we discuss the current limitations of our approach and outline promising research directions for addressing them (Section 8.6.1). We then present several potential applications for future investigation (Section 8.6.2). In what follows, we assume that $\varphi[\boldsymbol{\alpha}]$ is a $\mathcal{T}$-formula and

$$\psi_{\mathsf{red}} \stackrel{\text{def}}{=} \mathcal{T}^{\mathsf{red}}_{\boldsymbol{\alpha}}\text{-}*\mathsf{dDNNFcompile}(\varphi), \quad \psi_{\mathsf{ext}} \stackrel{\text{def}}{=} \mathcal{T}^{\mathsf{ext}}_{\boldsymbol{\alpha}}\text{-}*\mathsf{dDNNFcompile}(\varphi)$$

for some form of *d-DNNF.

8.6.1 Limitations and Research Avenues

Our proposed approach inherits the limitations of propositional d-DNNFs, particularly the requirement that the (super)set of atoms $\boldsymbol{\alpha}$ in the formula and in the queries be known at compilation time. For SMT, this is much more limiting than in the propositional case because new atoms that express new constraints on the $\mathcal{T}$-variables in the formula often appear at query time. Also, our approach requires that a complete set $Cl_{\boldsymbol{\alpha}}(\varphi)$ of $\mathcal{T}$-lemmas must be enumerated upfront, which may sometimes be a bottleneck. To address these limitations, we outline some possible solutions that share a common idea: a *hybrid eager/lazy approach*.

Queries with extra $\mathcal{T}$-atoms. Queries involving extra $\mathcal{T}$-atoms, i.e., not in $\boldsymbol{\alpha}$, cannot be handled with our approach directly. For example, consider the CO problem of checking the $\mathcal{T}$-unsatisfiability of $\mu \wedge \psi_{\mathsf{red}}|_\mu$, for $\mu \stackrel{\text{def}}{=} \bigwedge_i \ell_i$, and let $\boldsymbol{\beta} \stackrel{\text{def}}{=} \mathit{Atoms}(\mu)$ be a set of $\mathcal{T}$-atoms s.t. $\boldsymbol{\beta} \setminus \boldsymbol{\alpha} \neq \emptyset$. Thus $\mu \wedge \psi_{\mathsf{red}}|_\mu$ is not necessarily $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}\cup\boldsymbol{\beta}}$ because some $\eta \in H_{\boldsymbol{\alpha}}(\psi_{\mathsf{red}})$ can be such that $\mu \wedge \eta$ is $\mathcal{T}$-unsatisfiable. To make $\mu \wedge \psi_{\mathsf{red}}$ $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}\cup\boldsymbol{\beta}}$, we should compute and conjoin to it a set of novel $\mathcal{T}$-lemmas on $\boldsymbol{\alpha}\cup\boldsymbol{\beta}$ which rule out all the $\mathcal{T}$-unsatisfiable $\mu \wedge \eta$'s s.t. $\eta \in H_{\boldsymbol{\alpha}}(\psi_{\mathsf{red}})$. This would require (a) enumerating such a set of $\mathcal{T}$-lemmas, which may be expensive; (b) conjoining them to the compiled formula, which would require recompiling the augmented formula from scratch unless the target is, for example, an OBDD or SDD, which permits efficient conjunctive combination.

To cope with these scenarios, we can investigate *hybrid eager/lazy querying techniques*, e.g., by enumerating the $\mathcal{T}$-satisfiable partial assignments μ_i by very efficiently traversing ψ_{red} in an and-or fashion [125], invoking a $\mathcal{T}$-solver on each $(\mu \wedge \mu_i)$. To this end:

(i) as in the propositional case, a d-DNNF would restrict the enumeration to partial assignments μ_i that propositionally satisfy ψ_{red}, and the enumeration would be very efficient;

(ii) with precompiled $\mathcal{T}$-lemmas, the $\mathcal{T}$-solver would check only the $\mathcal{T}$-satisfiable μ_i's, and the $\mathcal{T}$-solver could be customized to restrict its search only to conflicts involving μ.

Analogous considerations may hold for ψ_{ext} and for other queries as well.

Coping with partially-enumerated $\mathcal{T}$-lemma sets. Our theory-agnostic $\mathcal{T}$-lemma-enumeration strategies (Chapter 7) are highly parallelizable, and leave several research avenues for further improvements. Nevertheless, the upfront enumeration of a set $Cl_{\boldsymbol{\alpha}}(\varphi)$ of $\mathcal{T}$-lemmas could still be a bottleneck for the whole process.

Suppose that, due to limited resources, it is not possible to produce a complete set of $\mathcal{T}$-lemmas $Cl_{\boldsymbol{\alpha}}(\varphi)$ ruling out $P_{\boldsymbol{\alpha}}(\varphi)$, so that ψ_{red} is not "completely $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$", that is, $P_{\boldsymbol{\alpha}}(\psi_{\mathsf{red}}) \neq \emptyset$. Therefore, its queries CO, CE, CT, ME, EQ, and SE can no longer be reduced to their propositional counterparts.

This reduces to the same case as in the previous paragraph, and a hybrid eager/lazy approach can benefit from Items (i) and (ii) here as well.

8.6.2 Potential Applications

This chapter opens many research avenues for developing and testing efficient procedures supporting *KC Modulo Theories* [147] functionality and for investigating their applications. In particular, we envision extending techniques and applications from propositional knowledge compilation to more expressive domains.

In propositional KC applications, a d-DNNF formula $\psi \stackrel{\text{def}}{=} \mathsf{d\text{-}DNNF}(\varphi)$ often encodes some general knowledge base φ, and the queries are applied each time to different restrictions $\mu_i \wedge \psi|_{\mu_i}$, where μ_i represents a list of truth-value assumptions on $\mathit{Atoms}(\varphi)$, for many possible μ_i's. We can extend this approach to the SMT level because $\mu_i \wedge \psi_{\mathsf{red}}|_{\mu_i}$ is still $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ (Lemma E.2.1).

Selective clause deduction. The original knowledge-base $\mathcal{T}$-formula is often a conjunction $\varphi \stackrel{\text{def}}{=} \bigwedge_j \varphi_j$, where each subformula φ_j represents a distinct piece of knowledge. Let $\Phi \stackrel{\text{def}}{=} \{\varphi_j\}_j$; different deduction queries may depend only on subsets $\Phi_k \subseteq \Phi$. To support such selective queries, we can introduce fresh selection atoms A_j and compile

$$\varphi' \stackrel{\text{def}}{=} \bigwedge_j (\neg A_j \vee \varphi_j), \qquad \psi'_{\mathsf{red}} \stackrel{\text{def}}{=} \mathcal{T}^{\mathsf{red}}_{\boldsymbol{\alpha}}\text{-}{*}\mathsf{dDNNFcompile}(\varphi').$$

For a subset Φ_k, let $\mu_k \stackrel{\text{def}}{=} \bigwedge_{\varphi_j \in \Phi_k} A_j$. Then, for any clause C_i, $\bigwedge_{\varphi_j \in \Phi_k} \varphi_j \models_{\mathcal{T}} C_i$ iff $\mu_k \wedge \varphi' \models_{\mathcal{T}} C_i$, iff $\mu_k \wedge \psi'_{\mathsf{red}}|_{\mu_k} \models_{\mathcal{T}} C_i$. Thus, after one compilation, many selective clause-deduction queries can be answered efficiently: choose the desired subset Φ_k via μ_k, then check the resulting residual for $\mathbb{B}$-inconsistency in polynomial time.

Probabilistic reasoning. In the propositional world, model counting is used as a way of estimating probabilities of complex events φ encoded as propositional formulas over random Boolean variables, typically using the format $\psi \stackrel{\text{def}}{=} \mathsf{d\text{-}DNNF}(\varphi)$ to exploit polytime model counting (CT), as follows:

$$\Pr(\psi) = \frac{\#\mathsf{SAT}(\psi)}{2^{|\mathbf{A}|}} \quad \Pr(\mu \wedge \psi) = \frac{\#\mathsf{SAT}(\mu \wedge \psi|_{\mu})}{2^{|\mathbf{A}|}} \quad \Pr(\mu \mid \psi) = \frac{\#\mathsf{SAT}(\mu \wedge \psi|_{\mu})}{\#\mathsf{SAT}(\psi)} \quad (8.8)$$

where $\mu \stackrel{\text{def}}{=} \bigwedge_i \ell_i$ represents a set of assumptions over propositional atoms in ψ. Hence, one can efficiently compute these probabilities for many distinct assumption sets μ_i.

In many hybrid propositional/numerical domains, such as deterministic numeric planning [230, 98], the truth assignments to propositional atoms $\{A_i\}_i$ are restricted by theory-specific constraints, so that not all of the satisfying truth assignments to the A_i's are possible. If so, Equation (8.8) can be extended to hybrid systems:

$$\Pr(\psi) = \frac{\mathcal{T}\text{-}\#\mathsf{SAT}(\psi)}{\mathcal{T}\text{-}\#\mathsf{SAT}(\top)} \quad \Pr(\mu \wedge \psi) = \frac{\mathcal{T}\text{-}\#\mathsf{SAT}(\mu \wedge \psi|_\mu)}{\mathcal{T}\text{-}\#\mathsf{SAT}(\top)} \quad \Pr(\mu \mid \psi) = \frac{\mathcal{T}\text{-}\#\mathsf{SAT}(\mu \wedge \psi|_\mu)}{\mathcal{T}\text{-}\#\mathsf{SAT}(\psi)}$$

where $\psi \stackrel{\text{def}}{=} \psi_{\mathsf{red}}$ and $\mu \stackrel{\text{def}}{=} \bigwedge_i \ell_i$ represents a set of assumptions over (some of) the propositional atoms in ψ. (The "$\mathcal{T}$-$\#\mathsf{SAT}(\top)$" in the denominator of prior probabilities denotes the count of $\mathcal{T}$-satisfiable total truth assignments on $\boldsymbol{\alpha}$.) Thus, as with the propositional case above, given ψ_{red}, one can quickly compute such probabilities for many different assumption sets μ_i.

To this end, we note that our framework straightforwardly extends to *weighted* $\mathcal{T}$-#SAT, also allowing us to consider a probability distribution over the propositional atoms.

Volume Computation and Weighted Model Integration. Other frameworks for probabilistic reasoning in hybrid propositional/numerical domains are based on the enumeration of *disjoint partial* $\mathcal{T}$-satisfiable assignments, e.g., #SMT($\mathcal{LRA}$), i.e., volume computation [141, 43], and WMI [13, 161, 204, 205] (see also Section 1.4 and Chapter 4). This suggests that polytime ME achieved with $\mathcal{T}$-reduced d-DNNFs could be leveraged to improve the efficiency of such procedures.

Planning and BMC. As also shown in Section 8.5, d-DNNFs Modulo Theories can be applied to temporal and numeric planning problems and, analogously, to SMT-based Bounded Model Checking (BMC) problems. In these scenarios, we can compile a $\mathcal{T}$-formula encoding k steps of the system and then query it by checking, for example, the reachability of different goal states, possibly while setting different initial states and constraining the actions taken at each step. Polytime quantitative planning reasoning [208] can also be extended to SMT.

Optimization. We note that, in the propositional case, d-DNNFs also allow a minimum-cardinality truth assignment to be computed in polytime, i.e., a truth assignment satisfying the formula with the minimum literal cost [183, 177]. This can be exploited, e.g., to compute (weighted) MaxSAT [183, 177]. This fact suggests that our framework could also be leveraged for polytime (weighted) MaxSMT problems.

8.7 Conclusions

In this chapter, we have introduced for the first time a formal framework for compiling SMT formulas into $\mathcal{T}$-equivalent d-DNNF $\mathcal{T}$-formulas in a way that supports polytime

SMT transformations and queries. The approach is theory-agnostic, independent of the target d-DNNF language, and can be implemented on top of existing tools. In particular, compilation relies on a lemma enumerator and a propositional compiler, both treated as black boxes. Transformations and queries can be reduced to their propositional counterparts and can be implemented by means of any propositional d-DNNF reasoner. We have implemented our approach in a prototype tool and performed an experimental evaluation on synthetic and real-world-inspired numeric planning benchmarks. The results confirm the feasibility and effectiveness of our approach.

Remarkably, our approach inherits from propositional d-DNNFs the intrinsic limitation that the (super)set of atoms in the formula and in the queries should be known at compilation time. Relaxing this assumption is non-trivial, as it would require extending the compilation process to handle incremental changes to the formula structure. This limitation opens several promising directions for future work, including hybrid techniques combining d-DNNF compilation and SMT solving, which we have discussed in the chapter. Finally, we have outlined several potential applications of the framework, highlighting its broader relevance beyond the benchmarks we have considered.

Chapter 9

Conclusions and Future Work

This dissertation has investigated SMT enumeration as a versatile algorithmic tool for solving a broad range of problems involving SMT constraints. The central insight emerging from this work is that the practical effectiveness of SMT enumeration depends not only on the efficiency of the underlying solver, but also on the representation of the input formula, the form of the assignments being enumerated, and the way in which enumeration is integrated into application-specific reasoning procedures.

A recurring theme throughout this dissertation is the importance of enumerating few, short partial truth assignments rather than total ones. Producing compact assignments requires more than an efficient enumeration algorithm: formula encodings, projection strategies, and application-specific requirements all play a role in this process.

The first part of this dissertation studied how formula encodings affect enumeration. We showed that standard CNF transformations may inhibit the generation of short partial assignments. We then proposed a novel encoding that allows us to drastically reduce both the number of enumerated assignments and the overall enumeration time. This result highlights that encodings designed for satisfiability checking are not necessarily suitable for enumeration.

We then applied these insights to Weighted Model Integration, developing a structure-aware encoding and a specialized enumeration strategy that reduce the number of integrals that must be computed. The resulting algorithm achieves a significant speedup over state-of-the-art WMI approaches.

We also adapted partial-assignment enumeration to Optimization Modulo Theories (OMT), developing OMT-specific assignment-reduction techniques that accelerate the optimization loop. These techniques improve performance in both exact and anytime OMT solving.

A related decomposition strategy was applied to Maximum A Posteriori inference, where SMT enumeration partitions the feasible space into subproblems that can be handled by external numerical optimizers. This makes it possible to address highly nonlinear or implicitly defined objective functions that cannot be handled by standard OMT procedures.

The dissertation further addressed the computation of a complete set of theory lemmas that rule out all theory-unsatisfiable truth assignments propositionally satisfying

a given SMT formula. To this end, we developed orthogonal and composable techniques, including a parallelizable cube-and-conquer approach and projected enumeration on subsets of atoms of the input formula. These methods substantially improve scalability while preserving completeness.

Finally, the generated theory lemmas were used to formalize and implement a framework for d-DNNFs Modulo Theories. By augmenting the input formula with such lemmas, the framework provably supports polytime SMT transformations and queries, including the counting and enumeration of theory-satisfiable assignments. This is confirmed by experimental results showing that, for example, counting queries on the compiled d-DNNF $\mathcal{T}$-formula can be answered in negligible time, whereas the same queries are prohibitively expensive for standard SMT-based approaches.

Overall, these contributions show that SMT enumeration should not be viewed as an isolated solver functionality. Its effectiveness depends on several factors, from the encoding of the input formula to the form of the enumerated assignments and their use within application-specific procedures. Appropriate choices at these different levels can improve performance by orders of magnitude without requiring modifications to the underlying SMT solver.

Future Work

This work opens several avenues for future research.

First, the techniques developed in this dissertation for Weighted Model Integration and constrained Maximum A Posteriori inference may enable the application of these reasoning methods to a broader range of real-world problems. Further work should investigate their scalability on larger and more complex instances, as well as their integration into practical probabilistic reasoning systems.

The most promising directions, however, stem from our work on d-DNNFs Modulo Theories, which opens a broader field of Knowledge Compilation Modulo Theories [147], of which we believe we have only scratched the surface. As discussed in Section 8.6, the framework can be extended and improved in several ways. In particular, more efficient compilation techniques and a tighter integration with existing SMT solvers could significantly improve its practical applicability. Another important direction is the application of the framework to new reasoning tasks. Besides broadening its scope, new applications may reveal additional requirements and suggest further improvements to the compilation process, the supported queries, and the interaction with SMT solvers.

Bibliography

[1] Ralph Abboud, İsmail İlkan Ceylan, and Radoslav Dimitrov. "On the Approximability of Weighted Model Integration on DNF Structures". In: *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*. Vol. 17. 2020, pp. 828–837. DOI: 10.24963/kr.2020/85.

[2] Aws Albarghouthi, Loris D'Antoni, Samuel Drews, and Aditya V. Nori. "FairSquare: Probabilistic Verification of Program Fairness". In: *Proceedings of the ACM on Programming Languages* 1.OOPSLA (2017), pp. 1–30. DOI: 10.1145/3133904.

[3] Elvira Albert, Jesús Correas, Pablo Gordillo, Guillermo Román-Díez, and Albert Rubio. "GASOL: Gas Analysis and Optimization for Ethereum Smart Contracts". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2020, pp. 118–125. DOI: 10.1007/978-3-030-45237-7_7.

[4] Luca Amarù, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. "The EPFL Combinational Benchmark Suite". In: *Proceedings of the 24th International Workshop on Logic & Synthesis*. 2015. URL: http://infoscience.epfl.ch/record/207551.

[5] Dennis S. Arnon, George E. Collins, and Scott McCallum. "Cylindrical Algebraic Decomposition I: The Basic Algorithm". In: *SIAM Journal on Computing* 13.4 (1984), pp. 865–877. DOI: 10.1137/0213054.

[6] Rehan Abdul Aziz, Geoffrey Chu, Christian Muise, and Peter Stuckey. "#∃SAT: Projected Model Counting". In: *18th International Conference on Theory and Applications of Satisfiability Testing*. Springer, 2015, pp. 121–137. DOI: 10.1007/978-3-319-24318-4_10.

[7] F. Bacchus, S. Dalmao, and T. Pitassi. "Solving #SAT and Bayesian Inference with Backtracking Search". In: *Journal of Artificial Intelligence Research* 34 (2009), pp. 391–442. DOI: 10.1613/jair.2648.

[8] Bahareh Badban and Jaco van de Pol. "Zero, Successor and Equality in BDDs". In: *Annals of Pure and Applied Logic*. Festschrift on the Occasion of Helmut Schwichtenberg's 60th Birthday 133.1 (2005), pp. 101–123. DOI: 10.1016/j.apal.2004.10.005.

[9] Clark Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. "Satisfiability Modulo Theories". In: *Handbook of Satisfiability*. 2nd ed. Vol. 336. FAIA. IOS Press, 2021, pp. 1267–1329. DOI: 10.3233/FAIA201017.

[10] Roberto J. Bayardo and Robert C. Schrag. "Using CSP Look-Back Techniques to Solve Real-World SAT Instances". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. AAAI'97. AAAI Press, 1997, pp. 203–208. URL: https://cdn.aaai.org/AAAI/1997/AAAI97-032.pdf?utm_source=chatgpt.com.

[11] Vaishak Belle, Andrea Passerini, and Guy Van den Broeck. "Probabilistic Inference in Hybrid Domains by Weighted Model Integration". In: *Proceedings of the 24th International Joint Conference on Artificial Intelligence*. AAAI Press, 2015, pp. 2770–2776. DOI: 10.24963/ijcai.2017/756.

[12] Vaishak Belle, Guy Van den Broeck, and Andrea Passerini. "Component Caching in Hybrid Domains with Piecewise Polynomial Densities". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30. 2016. DOI: 10.1609/aaai.v30i1.10441.

[13] Vaishak Belle, Guy Van den Broeck, Andrea Passerini, M. Meila, and T. Heskes. "Hashing-Based Approximate Probabilistic Inference in Hybrid Domains". In: *31st Conference on Uncertainty in Artificial Intelligence*. AUAI PRESS, 2015, pp. 141–150. URL: https://lirias.kuleuven.be/1599178.

[14] Anna Bernasconi, Valentina Ciriani, Fabrizio Luccio, and Linda Pagli. "Compact DSOP and Partial DSOP Forms". In: *Theory of Computing Systems* 53.4 (2013), pp. 583–608. DOI: 10.1007/s00224-013-9447-2.

[15] Zhengbing Bian, Fabian Chudak, William Macready, Aidan Roy, Roberto Sebastiani, and Stefano Varotti. "Solving SAT (and MaxSAT) with a Quantum Annealer: Foundations, Encodings, and Preliminary Results". In: *Information and Computation* 275 (2020), p. 104609. DOI: 10.1016/j.ic.2020.104609.

[16] Armin Biere. "PicoSAT Essentials". In: *Journal on Satisfiability, Boolean Modeling and Computation* 4.2-4 (2008), pp. 75–97. DOI: 10.3233/SAT190039.

[17] Armin Biere, Marijn J. H. Heule, and Hans van Maaren. *Handbook of Satisfiability*. 2nd ed. Vol. 336. FAIA. IOS Press, 2021. DOI: 10.3233/FAIA336.

[18] Filippo Bigarella, Alessandro Cimatti, Alberto Griggio, Ahmed Irfan, Martin Jonáš, Marco Roveri, Roberto Sebastiani, and Patrick Trentin. "Optimization Modulo Non-linear Arithmetic via Incremental Linearization". In: *Frontiers of Combining Systems*. LNCS. Springer, 2021, pp. 213–231. DOI: 10.1007/978-3-030-86205-3_12.

[19] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. 1st ed. Information Science and Statistics. Springer, 2006. URL: https://link.springer.com/book/9780387310732.

[20] Magnus Björk. "Successful SAT Encoding Techniques". In: *Journal on Satisfiability, Boolean Modeling and Computation* 7.4 (2009), pp. 189–201. DOI: 10.3233/SAT190085.

[21] Nikolaj Bjørner and Anh-Dung Phan. "νZ - Maximal Satisfaction with Z3". In: *Proc International Symposium on Symbolic Computation in Software Science*. 2014, pp. 1–9. DOI: 10.29007/jmxj.

[22] Nikolaj Bjørner, Anh-Dung Phan, and Lars Fleckenstein. "νZ - An Optimizing SMT Solver". In: *Tools and Algorithms for the Construction and Analysis of Systems*. LNCS. Springer, 2015, pp. 194–199. DOI: 10.1007/978-3-662-46681-0_14.

[23] Miquel Bofill, Jordi Coll, Josep Suy, and Mateu Villaret. "An Efficient SMT Approach to Solve MRCPSP/Max Instances with Tight Constraints on Resources". In: *Principles and Practice of Constraint Programming*. LNCS. Springer, 2017, pp. 71–79. DOI: 10.1007/978-3-319-66158-2_5.

[24] Paul T. Boggs and Jon W. Tolle. "Sequential Quadratic Programming". In: *Acta Numerica* 4 (1995), pp. 1–51. DOI: 10.1017/S0962492900002518.

[25] Samuele Bortolotti, Emanuele Marconato, Tommaso Carraro, Paolo Morettin, Emile van Krieken, Antonio Vergari, Stefano Teso, and Andrea Passerini. "A Neuro-Symbolic Benchmark Suite for Concept Quality and Reasoning Shortcuts". In: *Advances in Neural Information Processing Systems*. Vol. 37. 2024, pp. 115861–115905. DOI: 10.52202/079017-3679.

[26] Abdelhamid Boudane, Said Jabbour, Lakhdar Sais, and Yakoub Salhi. "A SAT-based Approach for Mining Association Rules". In: *Proceedings of the 25th International Joint Conference on Artificial Intelligence*. AAAI Press, 2016, pp. 2472–2478. DOI: 10.5555/3060832.3060967.

[27] Thierry Boy de la Tour. "An Optimality Result for Clause Form Translation". In: *Journal of Symbolic Computation* 14.4 (1992), pp. 283–301. DOI: 10.1016/0747-7171(92)90009-S.

[28] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. DOI: 10.1017/CBO9780511804441.

[29] Marco Bozzano, Roberto Bruttomesso, Alessandro Cimatti, Tommi Junttila, Silvio Ranise, Peter van Rossum, and Roberto Sebastiani. "Efficient Theory Combination via Boolean Search". In: *Information and Computation* 204.10 (2006), pp. 1493–1525. DOI: 10.1016/j.ic.2005.05.011.

[30] F. Brglez and H. Fujiwara. "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran". In: *Proceedings of IEEE International Symposium Circuits and Systems (ISCAS 85)*. IEEE Press, 1985, pp. 677–692. URL: https://cir.nii.ac.jp/crid/1573387448859527168.

[31] Guy Van den Broeck and Adnan Darwiche. "On the Role of Canonicity in Knowledge Compilation". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 29. 2015. DOI: 10.1609/aaai.v29i1.9423.

[32] Randal E. Bryant. "Graph-Based Algorithms for Boolean Function Manipulation". In: *IEEE Transactions on Computers* C-35.8 (1986), pp. 677–691. DOI: 10.1109/TC.1986.1676819.

[33] Randal E. Bryant, Derek Beatty, Karl Brace, Kyeongsoon Cho, and Thomas Sheffler. "COSMOS: A Compiled Simulator for MOS Circuits". In: *Proceedings of the 24th ACM/IEEE Design Automation Conference*. DAC '87. ACM, 1987, pp. 9–16. DOI: 10.1145/37888.37890.

[34] Randal E. Bryant and Miroslav N. Velev. "Boolean Satisfiability with Transitivity Constraints". In: *ACM Transactions on Computational Logic* 3.4 (2002), pp. 604–627. DOI: 10.1145/566385.566390.

[35] Roberto Cavada, Alessandro Cimatti, Anders Franzén, Krishnamani Kalyanasundaram, Marco Roveri, and R.K. Shyamasundar. "Computing Predicate Abstractions by Integrating BDDs and SMT Solvers". In: *Formal Methods in Computer Aided Design*. 2007, pp. 69–76. DOI: 10.1109/FAMCAD.2007.35.

[36] Sagar Chaki, Arie Gurfinkel, and Ofer Strichman. "Decision Diagrams for Linear Arithmetic". In: *Formal Methods in Computer Aided Design*. 2009, pp. 53–60. DOI: 10.1109/FMCAD.2009.5351143.

[37] Apostolos Chalkis, Ioannis Emiris, Vissarion Fisikopoulos, Elias Tsigaridas, and Haris Zafeiropoulos. "Geometric Algorithms for Sampling the Flux Space of Metabolic Networks". In: *Journal of Computational Geometry* 14.1 (2023), pp. 195–220. DOI: 10.20382/jocg.v14i1a8.

[38] Apostolos Chalkis, Ioannis Z. Emiris, and Vissarion Fisikopoulos. "A Practical Algorithm for Volume Estimation Based on Billiard Trajectories and Simulated Annealing". In: *ACM Journal of Experimental Algorithmics* 28 (2023), 1.3:1–1.3:34. DOI: 10.1145/3584182.

[39] Apostolos Chalkis and Vissarion Fisikopoulos. "Volesti: Volume Approximation and Sampling for Convex Polytopes in R". In: *The R Journal* 13.2 (2021), p. 561. DOI: 10.32614/RJ-2021-077.

[40] William Chan, Richard Anderson, Paul Beame, and David Notkin. "Combining Constraint Solving and Symbolic Model Checking for a Class of Systems with Non-linear Constraints". In: *Computer Aided Verification*. LNCS. Springer, 1997, pp. 316–327. DOI: 10.1007/3-540-63166-6_32.

[41] Mark Chavira and Adnan Darwiche. "On Probabilistic Inference by Weighted Model Counting". In: *Artificial Intelligence* 172.6 (2008), pp. 772–799. DOI: 10.1016/j.artint.2007.11.002.

[42] Chaoran Cheng, Boran Han, Danielle Maddix, Abdul Fatir Ansari, Andrew Stuart, Michael W. Mahoney, and Bernie Wang. "Gradient-Free Generation for Hard-Constrained Systems". In: *International Conference on Learning Representations*. Vol. 2025. 2025, pp. 100510–100539. URL: https://proceedings.iclr.cc/paper_files/paper/2025/hash/f9d7d6c695bc983fcfb5b70a5fbdfd2f-Abstract-Conference.html.

[43] Dmitry Chistikov, Rayna Dimitrova, and Rupak Majumdar. "Approximate Counting in SMT and Value Estimation for Probabilistic Programs". In: *Acta Informatica* 54.8 (2017), pp. 729–764. DOI: 10.1007/s00236-017-0297-2.

[44] Arthur Choi, Doga Kisa, and Adnan Darwiche. "Compiling Probabilistic Graphical Models Using Sentential Decision Diagrams". In: *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*. LNCS. Springer, 2013, pp. 121–132. DOI: 10.1007/978-3-642-39091-3_11.

[45] YooJung Choi, Adnan Darwiche, and Guy Van den Broeck. "Optimal Feature Selection for Decision Robustness in Bayesian Networks". In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. 2017. DOI: 10.24963/ijcai.2017/215.

[46] Alessandro Cimatti, Anders Franzén, Alberto Griggio, Krishnamani Kalyanasundaram, and Marco Roveri. "Tighter Integration of BDDs and SMT for Predicate Abstraction". In: *2010 Design, Automation & Test in Europe Conference & Exhibition*. 2010, pp. 1707–1712. DOI: 10.1109/DATE.2010.5457090.

[47] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. "A Modular Approach to MaxSAT Modulo Theories". In: *16th International Conference on Theory and Applications of Satisfiability Testing*. LNCS. Springer, 2013, pp. 150–165. DOI: 10.1007/978-3-642-39071-5_12.

[48] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. "The MathSAT5 SMT Solver". In: *Tools and Algorithms for the Construction and Analysis of Systems*. LNCS. Springer, 2013, pp. 93–107. DOI: 10.1007/978-3-642-36742-7_7.

[49] Alessandro Cimatti, Alberto Griggio, and Roberto Sebastiani. "Computing Small Unsatisfiable Cores in Satisfiability Modulo Theories". In: *Journal of Artificial Intelligence Research* 40 (2011), pp. 701–728. DOI: 10.1613/jair.3196.

[50] Emanuele Civini, Gabriele Masina, Giuseppe Spallitta, and Roberto Sebastiani. "Beyond Eager Encodings: A Theory-Agnostic Approach to Theory-Lemma Enumeration in SMT". In: *International Joint Conference on Automated Reasoning*. Springer, 2026, pp. 286–304. DOI: 10.1007/978-3-032-32589-1_18.

[51] Emanuele Civini, Gabriele Masina, Giuseppe Spallitta, and Roberto Sebastiani. *Beyond Eager Encodings: A Theory-Agnostic Approach to Theory-Lemma Enumeration in SMT*. Extended version with proofs. Conference version published in Proceedings of IJCAR 2026. 2026. DOI: 10.48550/arXiv.2602.14634.

[52] Keith L. Clark. "Negation as Failure". In: *Logic and Data Bases*. Springer, 1978, pp. 293–322. DOI: 10.1007/978-1-4684-3384-5_11.

[53] Edmund Clarke, Daniel Kroening, Natasha Sharygina, and Karen Yorav. "Predicate Abstraction of ANSI-C Programs Using SAT". In: *Formal Methods in System Design* 25.2 (2004), pp. 105–127. DOI: 10.1023/B:FORM.0000040025.89719.f3.

[54] George E. Collins. "Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition–Preliminary Report". In: *ACM SIGSAM Bulletin* 8.3 (1974), pp. 80–90. DOI: 10.1145/1086837.1086852.

[55] Andrew R. Conn, Nicholas I. M. Gould, and Philippe L. Toint. "Part III Trust-Region Methods for Constrained Optimization with Convex Constraints". In: *Trust Region Methods*. MOS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics, 2000, pp. 439–439. DOI: 10.1137/1.9780898719857.pt3.

[56] Adnan Darwiche. "Compiling Knowledge into Decomposable Negation Normal Form". In: *Proceedings of the 16th International Joint Conference on Artificial Intelligence*. Vol. 1. IJCAI'99. Morgan Kaufmann Publishers Inc., 1999, pp. 284–289. URL: https://www.ijcai.org/Proceedings/99-1/Papers/042.pdf.

[57] Adnan Darwiche. "Decomposable Negation Normal Form". In: *Journal of the ACM* 48.4 (2001), pp. 608–647. DOI: 10.1145/502090.502091.

[58] Adnan Darwiche. "On the Tractable Counting of Theory Models and Its Application to Truth Maintenance and Belief Revision". In: *Journal of Applied Non-Classical Logics* 11.1-2 (2001), pp. 11–34. DOI: 10.3166/jancl.11.11-34.

[59] Adnan Darwiche. "A Differential Approach to Inference in Bayesian Networks". In: *Journal of the ACM (JACM)* 50.3 (2003), pp. 280–305. DOI: 10.1145/765568.765570.

[60] Adnan Darwiche. "SDD: A New Canonical Representation of Propositional Knowledge Bases". In: *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*. Vol. 2. IJCAI'11. AAAI Press, 2011, pp. 819–826. DOI: 10.5591/978-1-57735-516-8/IJCAI11-143.

[61] Adnan Darwiche and Auguste Hirth. "On the (Complete) Reasons Behind Decisions". In: *Journal of Logic, Language and Information* 32.1 (2023), pp. 63–88. DOI: 10.1007/s10849-022-09377-8.

[62] Adnan Darwiche and Pierre Marquis. "A Knowledge Compilation Map". In: *Journal of Artificial Intelligence Research* 17.1 (2002), pp. 229–264. DOI: 10.1613/jair.989.

[63] Jesús A. De Loera, Raymond Hemmecke, Jeremiah Tauzer, and Ruriko Yoshida. "Effective Lattice Point Counting in Rational Convex Polytopes". In: *Journal of Symbolic Computation*. Symbolic Computation in Algebra and Geometry 38.4 (2004), pp. 1273–1302. DOI: 10.1016/j.jsc.2003.04.003.

[64] Rodrigo De Salvo Braz, Ciaran O'Reilly, Vibhav Gogate, and Rina Dechter. "Probabilistic Inference Modulo Theories". In: *Proceedings of the 25th International Joint Conference on Artificial Intelligence*. AAAI Press, 2016, pp. 3591–3599. URL: https://www.ijcai.org/Proceedings/16/Papers/506.pdf.

[65] David Deharbe and Silvio Ranise. "Light-Weight Theorem Proving for Debugging and Verifying Units of Code". In: *Proceedings of the 1st International Conference on Software Engineering and Formal Methods*. 2003, pp. 220–228. DOI: 10.1109/SEFM.2003.1236224.

[66] Vincent Derkinderen. "Pruning Boolean d-DNNF Circuits Through Tseitin-Awareness". In: *2024 IEEE 36th International Conference on Tools with Artificial Intelligence*. 2024, pp. 663–670. DOI: 10.1109/ICTAI62512.2024.00099.

[67] Vincent Derkinderen and Jean-Marie Lagniez. "Circuit-Aware d-DNNF Compilation". In: *Proceedings of the 34th International Joint Conference on Artificial Intelligence*. Vol. 1. 2025, pp. 4454–4462. DOI: 10.24963/ijcai.2025/496.

[68] Vincent Derkinderen, Pedro Zuidberg Dos Martires, Samuel Kolb, and Paolo Morettin. *Top-Down Knowledge Compilation for Counting Modulo Theories*. Presented at the Workshop on Counting and Sampling at SAT 2023. 2023. DOI: 10.48550/arXiv.2306.04541.

[69] Michelangelo Diligenti, Marco Gori, and Claudio Saccà. "Semantic-Based Regularization for Learning and Inference". In: *Artificial Intelligence*. Combining Constraint Solving with Mining and Learning 244 (2017), pp. 143–165. DOI: 10.1016/j.artint.2015.08.011.

[70] Jingwen Ding, Giuseppe Spallitta, and Roberto Sebastiani. "Effective Prime Factorization via Quantum Annealing by Modular Locally-Structured Embedding". In: *Scientific Reports* 14.1 (2024), p. 3518. DOI: 10.1038/s41598-024-53708-7.

[71] Imen Ouled Dlala, Said Jabbour, Lakhdar Sais, and Boutheina Ben Yaghlane. "A Comparative Study of SAT-Based Itemsets Mining". In: *Research and Development in Intelligent Systems XXXIII*. Springer, 2016, pp. 37–52. DOI: 10.1007/978-3-319-47175-4_3.

[72] Pedro Zuidberg Dos Martires, Anton Dries, and Luc De Raedt. "Exact and Approximate Weighted Model Integration with Probability Density Functions Using Knowledge Compilation". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 7825–7833. DOI: 10.1609/aaai.v33i01.33017825.

[73] Anton Dries, Angelika Kimmig, Wannes Meert, Joris Renkens, Guy Van den Broeck, Jonas Vlasselaer, and Luc De Raedt. "ProbLog2: Probabilistic Logic Programming". In: *Machine Learning and Knowledge Discovery in Databases*. Springer, 2015, pp. 312–315. DOI: 10.1007/978-3-319-23461-8_37.

[74] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[75] Bruno Dutertre and Leonardo de Moura. "A Fast Linear-Arithmetic Solver for DPLL(T)". In: *Computer Aided Verification*. LNCS. Springer, 2006, pp. 81–94. DOI: 10.1007/11817963_11.

[76] Stefan C. Endres, Carl Sandrock, and Walter W. Focke. "A Simplicial Homology Algorithm for Lipschitz Optimisation". In: *Journal of Global Optimization* 72.2 (2018), pp. 181–217. DOI: 10.1007/s10898-018-0645-y.

[77] Katalin Fazekas, Fahiem Bacchus, and Armin Biere. "Implicit Hitting Set Algorithms for Maximum Satisfiability Modulo Theories". In: *International Joint Conference on Automated Reasoning*. Vol. 10900. LNCS. Springer, 2018, pp. 134–151. DOI: 10.1007/978-3-319-94205-6_10.

[78] Jonathan Feldstein and Vaishak Belle. "Lifted Reasoning Meets Weighted Model Integration". In: *37th Conference on Uncertainty in Artificial Intelligence*. PMLR, 2021, pp. 322–332. URL: https://proceedings.mlr.press/v161/feldstein21a.html.

[79] Daan Fierens, Guy Van Den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. "Inference and Learning in Probabilistic Logic Programs Using Weighted Boolean Formulas". In: *Theory and Practice of Logic Programming* 15.3 (2015), pp. 358–401. DOI: 10.1017/S1471068414000076.

[80] Marc Fischer, Mislav Balunovic, Dana Drachsler-Cohen, Timon Gehr, Ce Zhang, and Martin Vechev. "DL2: Training and Querying Neural Networks with Logic". In: *Proceedings of the 36th International Conference on Machine Learning*. PMLR, 2019, pp. 1931–1941. URL: https://proceedings.mlr.press/v97/fischer19a.html.

[81] Pascal Fontaine and E. Pascal Gribomont. "Using BDDs with Combinations of Theories". In: *Logic for Programming, Artificial Intelligence, and Reasoning*. LNCS. Springer, 2002, pp. 190–201. DOI: 10.1007/3-540-36078-6_13.

[82] Dror Fried, Alexander Nadel, Roberto Sebastiani, and Yogev Shalmon. "Entailing Generalization Boosts Enumeration". In: *27th International Conference on Theory and Applications of Satisfiability Testing*. Vol. 305. LIPIcs. LZI, 2024, 13:1–13:14. DOI: 10.4230/LIPIcs.SAT.2024.13.

[83] Dror Fried, Alexander Nadel, and Yogev Shalmon. "AllSAT for Combinational Circuits". In: *26th International Conference on Theory and Applications of Satisfiability Testing*. Vol. 271. LIPIcs. LZI, 2023, 9:1–9:18. DOI: 10.4230/LIPIcs.SAT.2023.9.

[84] Jan Friso Groote and Jaco van de Pol. "Equational Binary Decision Diagrams". In: *Logic for Programming, Artificial Intelligence, and Reasoning*. Lecture Notes in Artificial Intelligence. Springer, 2000, pp. 161–178. DOI: 10.1007/3-540-44404-1_11.

[85] Marco Gario and Andrea Micheli. "PySMT: A Solver-Agnostic Library for Fast Prototyping of SMT-based Algorithms". In: *SMT Workshop 2015*. 2015. URL: https://github.com/pysmt/pysmt.

[86] Bernd Gärtner and Sven Schönherr. "An Efficient, Exact, and Generic Quadratic Programming Solver for Geometric Optimization". In: *Proceedings of the 16th Annual Symposium on Computational Geometry*. SCG '00. ACM, 2000, pp. 110–118. DOI: 10.1145/336154.336191.

[87] Cunjing Ge, Feifei Ma, Peng Zhang, and Jian Zhang. "Computing and Estimating the Volume of the Solution Space of SMT(LA) Constraints". In: *Theoretical Computer Science* 743 (2018), pp. 110–129. DOI: 10.1016/j.tcs.2016.10.019.

[88] Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. "Conflict-Driven Answer Set Enumeration". In: *Logic Programming and Nonmonotonic Reasoning*. Vol. 4483. LNCS. Springer, 2007, pp. 136–148. DOI: 10.1007/978-3-540-72200-7_13.

[89] Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. "Solution Enumeration for Projected Boolean Search Problems". In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Springer, 2009, pp. 71–86. DOI: 10.1007/978-3-642-01929-6_7.

[90] Timon Gehr, Sasa Misailovic, and Martin Vechev. "PSI: Exact Symbolic Inference for Probabilistic Programs". In: *Computer Aided Verification*. LNCS. Springer, 2016, pp. 62–83. DOI: 10.1007/978-3-319-41528-4_4.

[91] Andrew Gelman and Donald B. Rubin. "Inference from Iterative Simulation Using Multiple Sequences". In: *Statistical Science* 7.4 (1992), pp. 457–472. DOI: 10.1214/ss/1177011136.

[92] Soroush Ghandi, Benjamin Quost, and Cassio de Campos. "Probabilistic Circuits with Constraints via Convex Optimization". In: *Machine Learning and Knowledge Discovery in Databases. Research Track*. Springer, 2024, pp. 161–177. DOI: 10.1007/978-3-031-70352-2_10.

[93] Eleonora Giunchiglia, Mihaela Cătălina Stoian, Salman Khan, Fabio Cuzzolin, and Thomas Lukasiewicz. "ROAD-R: The Autonomous Driving Dataset with Logical Requirements". In: *Machine Learning* 112.9 (2023), pp. 3261–3291. DOI: 10.1007/s10994-023-06322-z.

[94] Enrico Giunchiglia, Yuliya Lierler, and Marco Maratea. "Answer Set Programming Based on Propositional Satisfiability". In: *Journal of Automated Reasoning* 36.4 (2006), pp. 345–377. DOI: 10.1007/s10817-006-9033-2.

[95] Anuj Goel, Khurram Sajid, Hai Zhou, Adnan Aziz, and Vigyan Singhal. "BDD Based Procedures for a Theory of Equality with Uninterpreted Functions". In: *Formal Methods in System Design* 22.3 (2003), pp. 205–224. DOI: 10.1023/A:1022988809947.

[96] Anuj Goel, Khurrarn Sajid, Hai Zhou, Adnan Aziz, and Vigyan Singhal. "BDD Based Procedures for a Theory of Equality with Uninterpreted Functions". In: *Computer Aided Verification*. LNCS. Springer, 1998, pp. 244–255. DOI: 10.1007/BFb0028749.

[97] Carla P. Gomes, Ashish Sabharwal, and Bart Selman. "Model Counting". In: *Handbook of Satisfiability*. IOS Press, 2021, pp. 993–1014. DOI: 10.3233/FAIA201009.

[98] Peter Gregory, Derek Long, Maria Fox, and J. Christopher Beck. "Planning Modulo Theories: Extending the Planning Paradigm". In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 22. 2012, pp. 65–73. DOI: 10.1609/icaps.v22i1.13505.

[99] Orna Grumberg, Assaf Schuster, and Avi Yadgar. "Memory Efficient All-Solutions SAT Solver and Its Application for Reachability Analysis". In: *Formal Methods in Computer Aided Design*. Vol. 3312. Springer, 2004, pp. 275–289. DOI: 10.1007/978-3-540-30494-4_20.

[100] Ofer Guthmann, Ofer Strichman, and Anna Trostanetski. "Minimal Unsatisfiable Core Extraction for SMT". In: *2016 Formal Methods in Computer-Aided Design*. 2016, pp. 57–64. DOI: 10.1109/FMCAD.2016.7886661.

[101] *Handbook of Transport Modelling: 2nd Edition*. Vol. 1. Handbooks in Transport. Emerald Group Publishing Limited, 2007. DOI: 10.1108/9780857245670.

[102] Derek Hansen, Danielle C. Maddix, Shima Alizadeh, Gaurav Gupta, and Michael W. Mahoney. "Learning Physical Models That Can Respect Conservation Laws". In: *Proceedings of the 40th International Conference on Machine Learning*. PMLR, 2023, pp. 12469–12510. URL: https://proceedings.mlr.press/v202/hansen23b.html.

[103] M.C. Hansen, H. Yalcin, and J.P. Hayes. "Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering". In: *IEEE Design & Test of Computers* 16.3 (1999), pp. 72–80. DOI: 10.1109/54.785838.

[104] Julien Henry, Mihail Asavoae, David Monniaux, and Claire Maïza. "How to Compute Worst-Case Execution Time by Optimization Modulo Theory and a Clever Encoding of Program Semantics". In: *15th ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems*. ACM, 2014, pp. 43–52. DOI: 10.1145/2597809.2597817.

[105] Marijn J. H. Heule, Oliver Kullmann, Siert Wieringa, and Armin Biere. "Cube and Conquer: Guiding CDCL SAT Solvers by Lookaheads". In: *Hardware and Software: Verification and Testing*. Springer, 2012, pp. 50–65. DOI: 10.1007/978-3-642-34188-5_8.

[106] Jinbo Huang and Adnan Darwiche. "Using DPLL for Efficient OBDD Construction". In: *7th International Conference on Theory and Applications of Satisfiability Testing*. LNCS. Springer, 2004, pp. 157–172. DOI: 10.1007/11527695_13.

[107] Antti E. J. Hyvärinen, Matteo Marescotti, and Natasha Sharygina. "Lookahead in Partitioning SMT". In: *2021 Formal Methods in Computer Aided Design*. 2021, pp. 271–279. DOI: 10.34727/2021/isbn.978-3-85448-046-4_37.

[108] Markus Iser, Carsten Sinz, and Mana Taghdiri. "Minimizing Models for Tseitin-Encoded SAT Instances". In: *16th International Conference on Theory and Applications of Satisfiability Testing*. LNCS. Springer, 2013, pp. 224–232. DOI: 10.1007/978-3-642-39071-5_17.

[109] Said Jabbour, Joao Marques-Silva, Lakhdar Sais, and Yakoub Salhi. "Enumerating Prime Implicants of Propositional Formulae in Conjunctive Normal Form". In: *Logics in Artificial Intelligence*. Vol. 8761. LNCS. Springer, 2014, pp. 152–165. DOI: 10.1007/978-3-319-11558-0_11.

[110] Paul Jackson and Daniel Sheridan. "Clause Form Conversions for Boolean Circuits". In: *7th International Conference on Theory and Applications of Satisfiability Testing*. LNCS. Springer, 2005, pp. 183–198. DOI: 10.1007/11527695_15.

[111] Karthick Jayaraman, Nikolaj Bjørner, Geoff Outhred, and Charlie Kaufman. *Automated Analysis and Debugging of Network Connectivity Policies*. Tech. rep. MSR-TR-2014-102. Microsoft, 2014. URL: https://www.microsoft.com/en-us/research/publication/automated-analysis-and-debugging-of-network-connectivity-policies/.

[112] Jihwan Jeong, Scott Sanner, and Akshat Kumar. "A Mixed-Integer Linear Programming Reduction of Disjoint Bilinear Programs via Symbolic Variable Elimination". In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 2023, pp. 79–95. DOI: 10.1007/978-3-031-33271-5_6.

[113] Fuqi Jia, Yuhang Dong, Rui Han, Pei Huang, Minghao Liu, Feifei Ma, and Jian Zhang. "A Complete Algorithm for Optimization Modulo Nonlinear Real Arithmetic". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 39.11 (2025), pp. 11255–11263. DOI: 10.1609/aaai.v39i11.33224.

[114] HoonSang Jin, HyoJung Han, and Fabio Somenzi. "Efficient Conflict Analysis for Finding All Satisfying Assignments of a Boolean Circuit". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 3440. Springer, 2005, pp. 287–300. DOI: 10.1007/978-3-540-31980-1_19.

[115] HoonSang Jin and Fabio Somenzi. "Prime Clauses for Fast Enumeration of Satisfying Assignments to Boolean Circuits". In: *Proceedings. 42nd Design Automation Conference, 2005.* 2005, pp. 750–753. DOI: 10.1109/DAC.2005.193911.

[116] Sarfraz Khurshid, Darko Marinov, Ilya Shlyakhter, and Daniel Jackson. "A Case for Efficient Solution Enumeration". In: *7th International Conference on Theory and Applications of Satisfiability Testing*. LNCS. Springer, 2004, pp. 272–286. DOI: 10.1007/978-3-540-24605-3_21.

[117] Rafael Kiesel and Thomas Eiter. "Knowledge Compilation and More with SharpSAT-TD". In: *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*. Vol. 19. 2023, pp. 406–416. DOI: 10.24963/kr.2023/40.

[118] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representation*. arXiv, 2015. DOI: 10.48550/arXiv.1412.6980.

[119] Diederik P. Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: *International Conference on Learning Representation*. arXiv, 2014. DOI: 10.48550/arXiv.1312.6114.

[120] Samuel Kolb, Pedro Zuidberg Dos Martires, and Luc De Raedt. "How to Exploit Structure While Solving Weighted Model Integration Problems". In: *35th Conference on Uncertainty in Artificial Intelligence*. PMLR, 2020, pp. 744–754. URL: https://proceedings.mlr.press/v115/kolb20a.html.

[121] Samuel Kolb, Martin Mladenov, Scott Sanner, Vaishak Belle, and Kristian Kersting. "Efficient Symbolic Integration for Probabilistic Inference". In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, 2018, pp. 5031–5037. DOI: 10.24963/ijcai.2018/698.

[122] Elias Kuiter, Sebastian Krieter, Chico Sundermann, Thomas Thüm, and Gunter Saake. "Tseitin or Not Tseitin? The Impact of CNF Transformations on Feature-Model Analyses". In: *37th IEEE/ACM International Conference on Automated Software Engineering*. ACM, 2022, pp. 1–13. DOI: 10.1145/3551349.3556938.

[123] Leander Kurscheidt, Gabriele Masina, Roberto Sebastiani, and Antonio Vergari. "The Theory and Practice of MAP Inference over Non-Convex Constraints". In: *Proceedings of the 43rd International Conference on Machine Learning*. 2026. URL: https://openreview.net/forum?id=jIZqAemuqk.

[124] Leander Kurscheidt, Paolo Morettin, Roberto Sebastiani, Andrea Passerini, and Antonio Vergari. "A Probabilistic Neuro-symbolic Layer for Algebraic Constraint Satisfaction". In: *The 41st Conference on Uncertainty in Artificial Intelligence*. 2025. URL: https://openreview.net/forum?id=9UkxftKU4I.

[125] Jean-Marie Lagniez and Emmanuel Lonca. "Leveraging Decision-DNNF Compilation for Enumerating Disjoint Partial Models". In: *21st International Conference on Principles of Knowledge Representation and Reasoning*. 2024. DOI: 10.24963/kr.2024/48.

[126] Jean-Marie Lagniez and Pierre Marquis. "An Improved Decision-DNNF Compiler". In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, 2017, pp. 667–673. DOI: 10.24963/ijcai.2017/93.

[127] Jean-Marie Lagniez and Pierre Marquis. "A Recursive Algorithm for Projected Model Counting". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 1536–1543. DOI: 10.1609/aaai.v33i01.33011536.

[128] Shuvendu K. Lahiri, Randal E. Bryant, and Byron Cook. "A Symbolic Approach to Predicate Abstraction". In: *Computer Aided Verification*. LNCS. Springer, 2003, pp. 141–153. DOI: 10.1007/978-3-540-45069-6_15.

[129] Shuvendu K. Lahiri, Robert Nieuwenhuis, and Albert Oliveras. "SMT Techniques for Fast Predicate Abstraction". In: *Computer Aided Verification*. LNCS. Springer, 2006, pp. 424–437. DOI: 10.1007/11817963_39.

[130] Jean B. Lasserre. "Global Optimization with Polynomials and the Problem of Moments". In: *SIAM Journal on Optimization* 11.3 (2001), pp. 796–817. DOI: 10.1137/S1052623400366802.

[131] Edward A. Lee. "Cyber Physical Systems: Design Challenges". In: *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*. 2008, pp. 363–369. DOI: 10.1109/ISORC.2008.25.

[132] Francesco Leofante, Enrico Giunchiglia, Erika Ábrahám, and Armando Tacchella. "Optimal Planning Modulo Theories". In: *Proceedings of the 29th International Joint Conference on Artificial Intelligence*. Vol. 4. 2020, pp. 4128–4134. DOI: 10.24963/ijcai.2020/571.

[133] Bin Li, M.S. Hsiao, and Shuo Sheng. "A Novel SAT All-Solutions Solver for Efficient Preimage Computation". In: *Automation and Test in Europe Conference and Exhibition Proceedings Design*. Vol. 1. 2004, pp. 272–277. DOI: 10.1109/DATE.2004.1268860.

[134] Yi Li, Aws Albarghouthi, Zachary Kincaid, Arie Gurfinkel, and Marsha Chechik. "Symbolic Optimization with SMT Solvers". In: *41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM, 2014, pp. 607–618. DOI: 10.1145/2535838.2535857.

[135] Jiaxin Liang, Feifei Ma, Junping Zhou, and Minghao Yin. "AllSATCC: Boosting AllSAT Solving with Efficient Component Analysis". In: *Proceedings of the 31st International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, 2022, pp. 1866–1872. DOI: 10.24963/ijcai.2022/259.

[136] Fangzhen Lin and Yuting Zhao. "ASSAT: Computing Answer Sets of a Logic Program by SAT Solvers". In: *Artificial Intelligence*. Nonmonotonic Reasoning 157.1 (2004), pp. 115–137. DOI: 10.1016/j.artint.2004.04.004.

[137] Sheng Liu and Jian Zhang. "Program Analysis: From Qualitative Analysis to Quantitative Analysis (NIER Track)". In: *33rd International Conference on Software Engineering*. 2011, pp. 956–959. DOI: 10.1145/1985793.1985957.

[138] Nuno P. Lopes, Nikolaj Bjørner, Patrice Godefroid, Karthick Jayaraman, and George Varghese. "Checking Beliefs in Dynamic Networks". In: *12th USENIX Symposium on Networked Systems Design and Implementation*. 2015, pp. 499–512. URL: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/lopes.

[139] Nuno P. Lopes, Nikolaj Bjørner, Patrice Godefroid, and George Varghese. *Network Verification in the Light of Program Verification*. Tech. rep. Microsoft, 2013. URL: https://www.microsoft.com/en-us/research/publication/network-verification-in-the-light-of-program-verification/.

[140] Weilin Luo, Hai Want, Hongzhen Zhong, Ou Wei, Biqing Fang, and Xiaotong Song. "An Efficient Two-phase Method for Prime Compilation of Non-clausal Boolean Formulae". In: *2021 IEEE/ACM International Conference On Computer Aided Design*. 2021, pp. 1–9. DOI: 10.1109/ICCAD51958.2021.9643520.

[141] Feifei Ma, Sheng Liu, and Jian Zhang. "Volume Computation for Boolean Combination of Linear Arithmetic Constraints". In: *Automated Deduction*. LNCS. Springer, 2009, pp. 453–468. DOI: 10.1007/978-3-642-02959-2_33.

[142] Feifei Ma, Jun Yan, and Jian Zhang. "Solving Generalized Optimization Problems Subject to SMT Constraints". In: *Frontiers in Algorithmics and Algorithmic Aspects in Information and Management*. Springer, 2012, pp. 247–258. DOI: 10.1007/978-3-642-29700-7_23.

[143] Joao Marques-Silva, Ines Lynce, and Sharad Malik. "Conflict-Driven Clause Learning SAT Solvers". In: *Handbook of Satisfiability*. Vol. 336. FAIA. IOS Press, 2021. DOI: 10.3233/FAIA200987.

[144] Gabriele Masina. *On CNF Conversion for SAT and SMT Enumeration: Benchmarks, Results and Plots*. 2024. DOI: 10.5281/zenodo.14035595.

[145] Gabriele Masina. *On CNF Conversion for SAT and SMT Enumeration: Source Code*. Zenodo. 2024. DOI: 10.5281/zenodo.14033421.

[146] Gabriele Masina, Emanuele Civini, Massimo Michelutti, Giuseppe Spallitta, and Roberto Sebastiani. *d-DNNF Modulo Theories: A General Framework for Polytime SMT Queries*. Extended version with proofs. Conference version published in Proceedings of SAT 2026. 2026. DOI: 10.48550/arXiv.2603.09975.

[147] Gabriele Masina, Emanuele Civini, Massimo Michelutti, Giuseppe Spallitta, and Roberto Sebastiani. "d-DNNF Modulo Theories: A General Framework for Polytime SMT Queries". In: *29th International Conference on Theory and Applications of Satisfiability Testing*. Vol. 377. LIPIcs. LZI, 2026, 25:1–25:19. DOI: 10.4230/LIPIcs.SAT.2026.25.

[148] Gabriele Masina and Roberto Sebastiani. "Exploiting Partial-Assignment Enumeration in Optimization Modulo Theories". In: *Frontiers of Combining Systems*. Springer, 2026, pp. 117–134. DOI: 10.1007/978-3-032-04167-8_7.

[149] Gabriele Masina, Giuseppe Spallitta, and Roberto Sebastiani. "On CNF Conversion for Disjoint SAT Enumeration". In: *26th International Conference on Theory and Applications of Satisfiability Testing*. Vol. 271. LIPIcs. LZI, 2023, 15:1–15:16. DOI: 10.4230/LIPIcs.SAT.2023.15.

[150] Gabriele Masina, Giuseppe Spallitta, and Roberto Sebastiani. "On CNF Conversion for SAT and SMT Enumeration". In: *Journal of Artificial Intelligence Research* 83 (2025). DOI: 10.1613/jair.1.16870.

[151] Kenneth L. McMillan. "Applying SAT Methods in Unbounded Symbolic Model Checking". In: *Computer Aided Verification*. LNCS. Springer, 2002, pp. 250–264. DOI: 10.1007/3-540-45657-0_19.

[152] Massimo Michelutti, Gabriele Masina, Giuseppe Spallitta, and Roberto Sebastiani. "Canonical Decision Diagrams Modulo Theories". In: *27th European Conference on Artificial Intelligence*. Vol. 392. FAIA. IOS Press, 2024, pp. 4319–4327. DOI: 10.3233/FAIA241007.

[153] Peter Bro Miltersen, Jaikumar Radhakrishnan, and Ingo Wegener. "On Converting CNF to DNF". In: *Theoretical Computer Science* 347.1-2 (2005), pp. 325–335. DOI: 10.1016/j.tcs.2005.07.029.

[154] Shin-Ichi Minato and Giovanni De Micheli. "Finding All Simple Disjunctive Decompositions Using Irredundant Sum-of-Products Forms". In: *1998 IEEE/ACM International Conference on Computer-Aided Design*. 1998, pp. 111–117. DOI: 10.1145/288548.288586.

[155] Sibylle Möhle and Armin Biere. "Dualizing Projected Model Counting". In: *30th IEEE International Conference on Tools with Artificial Intelligence*. 2018, pp. 702–709. DOI: 10.1109/ICTAI.2018.00111.

[156] Sibylle Möhle, Roberto Sebastiani, and Armin Biere. "Four Flavors of Entailment". In: *23rd International Conference on Theory and Applications of Satisfiability Testing*. LNCS. Springer, 2020, pp. 62–71. DOI: 10.1007/978-3-030-51825-7_5.

[157] Sibylle Möhle, Roberto Sebastiani, and Armin Biere. "On Enumerating Short Projected Models". In: *Discrete Applied Mathematics* 361 (2025), pp. 412–439. DOI: 10.1016/j.dam.2024.10.021.

[158] Jesper Møller, Jakob Lichtenberg, Henrik Reif Andersen, and Henrik Hulgaard. "Difference Decision Diagrams". In: *Computer Science Logic*. Vol. 1683. LNCS. Springer, 1999, pp. 111–125. DOI: 10.1007/3-540-48168-0_9.

[159] Paolo Morettin, Samuel Kolb, Stefano Teso, and Andrea Passerini. "Learning Weighted Model Integration Distributions". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 2020, pp. 5224–5231. DOI: 10.1609/aaai.v34i04.5967.

[160] Paolo Morettin, Andrea Passerini, and Roberto Sebastiani. "Efficient Weighted Model Integration via SMT-Based Predicate Abstraction". In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, 2017, pp. 720–728. DOI: 10.24963/ijcai.2017/100.

[161] Paolo Morettin, Andrea Passerini, and Roberto Sebastiani. "Advanced SMT Techniques for Weighted Model Integration". In: *Artificial Intelligence* 275.C (2019), pp. 1–27. DOI: 10.1016/j.artint.2019.04.003.

[162] Paolo Morettin, Andrea Passerini, and Roberto Sebastiani. *Probabilistic ML Verification via Weighted Model Integration*. 2024. DOI: 10.48550/arXiv.2402.04892.

[163] Paolo Morettin, Pedro Zuidberg Dos Martires, Samuel Kolb, and Andrea Passerini. "Hybrid Probabilistic Inference with Logical and Algebraic Constraints: A Survey". In: *Proceedings of the 30th International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, 2021, pp. 4533–4542. DOI: 10.24963/ijcai.2021/617.

[164] Antonio Morgado and Joao Marques-Silva. "Good Learning and Implicit Model Enumeration". In: *17th IEEE International Conference on Tools with Artificial Intelligence*. IEEE Computer Society, 2005, pp. 131–136. DOI: 10.1109/ICTAI.2005.69.

[165] Alexander Nadel and Vadim Ryvchin. "Bit-Vector Optimization". In: *Tools and Algorithms for the Construction and Analysis of Systems*. LNCS. Springer, 2016, pp. 851–867. DOI: 10.1007/978-3-662-49674-9_53.

[166] Sai Shankar Narasimhan, Shubhankar Agarwal, Oguzhan Akcin, Sujay Sanghavi, and Sandeep P. Chinchali. "Time Weaver: A Conditional Time Series Generation Model". In: *Proceedings of the 41st International Conference on Machine Learning*. PMLR, 2024, pp. 37293–37320. URL: https://proceedings.mlr.press/v235/narasimhan24a.html.

[167] Mark E. J. Newman and Gerard T. Barkema. *Monte Carlo Methods in Statistical Physics*. Oxford University Press, 1999. DOI: 10.1093/oso/9780198517962.001.0001.

[168] Chi Mai Nguyen, Roberto Sebastiani, Paolo Giorgini, and John Mylopoulos. "Multi-Objective Reasoning with Constrained Goal Models". In: *Requirements Engineering* 23.2 (2018), pp. 189–225. DOI: 10.1007/s00766-016-0263-5.

[169] Robert Nieuwenhuis and Albert Oliveras. "On SAT Modulo Theories and Optimization Problems". In: *9th International Conference on Theory and Applications of Satisfiability Testing*. LNCS. Springer, 2006, pp. 156–169. DOI: 10.1007/11814948_18.

[170] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. 2nd ed. Springer Series in Operations Research and Financial Engineering. Springer, 2006. DOI: 10.1007/978-0-387-40065-5.

[171] Héctor Palacios, Blai Bonet, Adnan Darwiche, and Héctor Geffner. "Pruning Conformant Plans by Counting Models on Compiled d-DNNF Representations". In: *Proceedings of the 15th International Conference on Automated Planning and Scheduling*. ICAPS'05. AAAI Press, 2005, pp. 141–150. URL: https://cdn.aaai.org/ICAPS/2005/ICAPS05-015.pdf.

[172] Luigi Palopoli, Fiora Pirri, and Clara Pizzuti. "Algorithms for Selective Enumeration of Prime Implicants". In: *Artificial Intelligence* 111.1-2 (1999), pp. 41–72. DOI: 10.1016/S0004-3702(99)00035-1.

[173] Stefan Panjkovic and Andrea Micheli. "Expressive Optimal Temporal Planning via Optimization Modulo Theory". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 37. 2023, pp. 12095–12102. DOI: 10.1609/aaai.v37i10.26426.

[174] Stefan Panjkovic and Andrea Micheli. "Abstract Action Scheduling for Optimal Temporal Planning via OMT". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. 2024, pp. 20222–20229. DOI: 10.1609/aaai.v38i18.30002.

[175] Quoc-Sang Phan. "Model Counting Modulo Theories". Thesis. Queen Mary University of London, 2015. URL: https://qmro.qmul.ac.uk/xmlui/handle/123456789/15130.

[176] Quoc-Sang Phan and Pasquale Malacaria. "All-Solution Satisfiability Modulo Theories: Applications, Algorithms and Benchmarks". In: *10th International Conference on Availability, Reliability and Security*. IEEE, 2015, pp. 100–109. DOI: 10.1109/ARES.2015.14.

[177] Knot Pipatsrisawat, Akop Palyan, Mark Chavira, Arthur Choi, and Adnan Darwiche. "Solving Weighted Max-SAT Problems in a Reduced Search Space: A Performance Analysis". In: *Journal on Satisfiability, Boolean Modelling and Computation* 4.2-4 (2008), pp. 191–217. DOI: 10.3233/SAT190044.

[178] David A. Plaisted and Steven Greenbaum. "A Structure-preserving Clause Form Translation". In: *Journal of Symbolic Computation* 2.3 (1986), pp. 293–304. DOI: 10.1016/S0747-7171(86)80028-1.

[179] Victoria Powers and Thorsten Wörmann. "An Algorithm for Sums of Squares of Real Polynomials". In: *Journal of Pure and Applied Algebra* 127.1 (1998), pp. 99–104. DOI: 10.1016/S0022-4049(97)83827-3.

[180] Steven Prestwich. "CNF Encodings". In: *Handbook of Satisfiability*. 2nd ed. Vol. 336. FAIA. IOS Press, 2021, pp. 75–100. DOI: 10.3233/FAIA200985.

[181] Alessandro Previti, Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. "Prime Compilation of Non-Clausal Formulae". In: *Proceedings of the 24th International Joint Conference on Artificial Intelligence*. 2015. URL: https://www.ijcai.org/Proceedings/15/Papers/281.pdf.

[182] Parikshit Ram and Alexander G. Gray. "Density Estimation Trees". In: *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '11. ACM, 2011, pp. 627–635. DOI: 10.1145/2020408.2020507.

[183] Miquel Ramírez and Hector Geffner. "Structural Relaxations by Variable Renaming and Their Compilation for Solving MinCostSAT". In: *Principles and Practice of Constraint Programming*. Springer, 2007, pp. 605–619. DOI: 10.1007/978-3-540-74970-7_43.

[184] Kavita Ravi and Fabio Somenzi. "Minimal Assignments for Bounded Model Checking". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 2988. LNCS. Springer, 2004, pp. 31–45. DOI: 10.1007/978-3-540-24730-2_3.

[185] Alexandre Robicquet, Amir Sadeghian, Alexandre Alahi, and Silvio Savarese. "Learning Social Etiquette: Human Trajectory Understanding In Crowded Scenes". In: *Proceedings of the 14th European Conference on Computer Vision*. Springer, 2016, pp. 549–565. DOI: 10.1007/978-3-319-46484-8_33.

[186] Sartaj Sahni. "Computationally Related Problems". In: *SIAM Journal on Computing* 3.4 (1974), pp. 262–279. DOI: 10.1137/0203021.

[187] T. Sang, F. Bacchus, P. Beame, Henry A. Kautz, and T. Pitassi. "Combining Component Caching and Clause Learning for Effective Model Counting". In: *7th International Conference on Theory and Applications of Satisfiability Testing*. 2004. URL: http://www.cs.rochester.edu/u/kautz/papers/modelcount-sat04.pdf.

[188] Scott Sanner and Ehsan Abbasnejad. "Symbolic Variable Elimination for Discrete and Continuous Graphical Models". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 26. 2012, pp. 1954–1960. DOI: 10.1609/aaai.v26i1.8406.

[189] Scott Sanner and David McAllester. "Affine Algebraic Decision Diagrams (AADDs) and Their Application to Structured Probabilistic Inference". In: *Proceedings of the 19th International Joint Conference on Artificial Intelligence*. IJCAI'05. Morgan Kaufmann Publishers Inc., 2005, pp. 1384–1390. URL: https://dl.acm.org/doi/10.5555/1642293.1642513.

[190] Enrico Scala, Patrik Haslum, Sylvie Thiébaux, and Miquel Ramirez. "Subgoaling Techniques for Satisficing and Optimal Numeric Planning". In: *Journal of Artificial Intelligence Research* 68 (2020), pp. 691–752. DOI: 10.1613/jair.1.11875.

[191] Roberto Sebastiani. "Lazy Satisfiability Modulo Theories". In: *Journal on Satisfiability, Boolean Modeling and Computation* 3.3-4 (2007), pp. 141–224. DOI: 10.3233/SAT190034.

[192] Roberto Sebastiani. "Colors Make Theories Hard". In: *International Joint Conference on Automated Reasoning*. Vol. 9706. LNCS. Springer, 2016, pp. 152–170. DOI: 10.1007/978-3-319-40229-1_11.

[193] Roberto Sebastiani. *Are You Satisfied by This Partial Assignment?* 2020. DOI: 10.48550/arXiv.2003.04225.

[194] Roberto Sebastiani. "Entailment vs. Verification for Partial-Assignment Satisfiability and Enumeration". In: *Automated Deduction*. Springer, 2025, pp. 717–735. DOI: 10.1007/978-3-031-99984-0_37.

[195] Roberto Sebastiani and Silvia Tomasi. "Optimization in SMT with LA(Q) Cost Functions". In: *International Joint Conference on Automated Reasoning*. Vol. 7364. LNCS. Springer, 2012, pp. 484–498. DOI: 10.1007/978-3-642-31365-3_38.

[196] Roberto Sebastiani and Silvia Tomasi. "Optimization Modulo Theories with Linear Rational Costs". In: *ACM Transactions on Computational Logic* 16.2 (2015), 12:1–12:43. DOI: 10.1145/2699915.

[197] Roberto Sebastiani and Patrick Trentin. "Pushing the Envelope of Optimization Modulo Theories with Linear-Arithmetic Cost Functions". In: *Tools and Algorithms for the Construction and Analysis of Systems*. LNCS. Springer, 2015, pp. 335–349. DOI: 10.1007/978-3-662-46681-0_27.

[198] Roberto Sebastiani and Patrick Trentin. "OptiMathSAT: A Tool for Optimization Modulo Theories". In: *Journal of Automated Reasoning* 64.3 (2020), pp. 423–460. DOI: 10.1007/s10817-018-09508-6.

[199] Arijit Shaw and Kuldeep S. Meel. "Approximate SMT Counting Beyond Discrete Domains". In: *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. 2025, pp. 1–7. DOI: 10.1109/DAC63849.2025.11133351.

[200] Arijit Shaw, Uddalok Sarkar, and Kuldeep S. Meel. "Efficient Volume Computation for SMT Formulas". In: *22nd International Conference on Principles of Knowledge Representation and Reasoning*. Vol. 22. 2025, pp. 544–554. DOI: 10.24963/kr.2025/53.

[201] Lennert De Smet, Pedro Zuidberg Dos Martires, Robin Manhaeve, Giuseppe Marra, Angelika Kimmig, and Luc De Raedt. "Neural Probabilistic Logic Programming in Discrete-Continuous Domains". In: *Proceedings of the 39th Conference on Uncertainty in Artificial Intelligence*. PMLR, 2023, pp. 529–538. URL: https://proceedings.mlr.press/v216/de-smet23a.html.

[202] Fabio Somenzi. *CUDD: CU Decision Diagram Package*. University of Colorado Boulder. 2009. URL: https://github.com/cuddorg/cudd.

[203] Giuseppe Spallitta. "Efficient and Effective Model Enumeration in SAT and SMT: Investigating Novel Procedures and Applications". PhD thesis. Trento, 2025. URL: https://iris.unitn.it/handle/11572/443352.

[204] Giuseppe Spallitta, Gabriele Masina, Paolo Morettin, Andrea Passerini, and Roberto Sebastiani. "SMT-based Weighted Model Integration with Structure Awareness". In: *38th Conference on Uncertainty in Artificial Intelligence*. Vol. 180. PMLR, 2022, pp. 1876–1885. URL: https://proceedings.mlr.press/v180/spallitta22a.html.

[205] Giuseppe Spallitta, Gabriele Masina, Paolo Morettin, Andrea Passerini, and Roberto Sebastiani. "Enhancing SMT-based Weighted Model Integration by Structure Awareness". In: *Artificial Intelligence* 328 (2024), p. 104067. DOI: 10.1016/j.artint.2024.104067.

[206] Giuseppe Spallitta, Roberto Sebastiani, and Armin Biere. "Disjoint Partial Enumeration without Blocking Clauses". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. 2024, pp. 8126–8135. DOI: 10.1609/aaai.v38i8.28652.

[207] Giuseppe Spallitta, Roberto Sebastiani, and Armin Biere. "Disjoint Projected Enumeration for SAT and SMT without Blocking Clauses". In: *Artificial Intelligence* 345 (2025), p. 104346. DOI: 10.1016/j.artint.2025.104346.

[208] David Speck, Markus Hecher, Daniel Gnad, Johannes K. Fichte, and Augusto B. Corrêa. "Counting and Reasoning with Plans". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 39. 2025, pp. 26688–26696. DOI: 10.1609/aaai.v39i25.34871.

[209] Russell Stewart and Stefano Ermon. "Label-Free Supervision of Neural Networks with Physics and Domain Knowledge". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 31. 2017. DOI: 10.1609/aaai.v31i1.10934.

[210] Mihaela Stoian and Eleonora Giunchiglia. "Beyond the Convexity Assumption: Realistic Tabular Data Generation under Quantifier-Free Real Linear Constraints". In: *International Conference on Learning Representations*. Vol. 2025. 2025, pp. 28812–28841. URL: https://proceedings.iclr.cc/paper_files/paper/2025/hash/47360925c45a166c96f652589265dee9-Abstract-Conference.html.

[211] Ofer Strichman. "On Solving Presburger and Linear Arithmetic with SAT". In: *Formal Methods in Computer-Aided Design*. Springer, 2002, pp. 160–170. DOI: 10.1007/3-540-36126-X_10.

[212] Ofer Strichman, Sanjit A. Seshia, and Randal E. Bryant. "Deciding Separation Formulas with SAT". In: *Computer Aided Verification*. Springer, 2002, pp. 209–222. DOI: 10.1007/3-540-45657-0_16.

[213] Chico Sundermann, Jacob Loth, and Thomas Thüm. "Efficient Slicing of Feature Models via Projected d-DNNF Compilation". In: *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. ASE '24. ACM, 2024, pp. 1332–1344. DOI: 10.1145/3691620.3695594.

[214] Chico Sundermann, Heiko Raab, Tobias Heß, Thomas Thüm, and Ina Schaefer. "Reusing d-DNNFs for Efficient Feature-Model Counting". In: *ACM Transactions on Software Engineering and Methodology* 33.8 (2024), 208:1–208:32. DOI: 10.1145/3680465.

[215] Stefano Teso, Roberto Sebastiani, and Andrea Passerini. "Structured Learning Modulo Theories". In: *Artificial Intelligence*. Combining Constraint Solving with Mining and Learning 244 (2017), pp. 166–187. DOI: 10.1016/j.artint.2015.04.002.

[216] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. Intelligent Robotics and Autonomous Agents. MIT Press, 2005.

[217] Abraham Temesgen Tibebu and Goerschwin Fey. "Augmenting All Solution SAT Solving for Circuits with Structural Information". In: *IEEE 21st International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*. 2018, pp. 117–122. DOI: 10.1109/DDECS.2018.00028.

[218] Takahisa Toda and Takeru Inoue. "Exploiting Functional Dependencies of Variables in All Solutions SAT Solvers". In: *Journal of Information Processing* 25.0 (2017), pp. 459–468. DOI: 10.2197/ipsjjip.25.459.

[219] Takahisa Toda and Takehide Soh. "Implementing Efficient All Solutions SAT Solvers". In: *ACM Journal of Experimental Algorithmics* 21 (2016), pp. 1–44. DOI: 10.1145/2975585.

[220] Takahisa Toda and Koji Tsuda. "BDD Construction for All Solutions SAT and Efficient Caching Mechanism". In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM, 2015, pp. 1880–1886. DOI: 10.1145/2695664.2695941.

[221] Patrick Trentin and Roberto Sebastiani. "Optimization Modulo the Theories of Signed Bit-Vectors and Floating-Point Numbers". In: *Journal of Automated Reasoning* 65.7 (2021), pp. 1071–1096. DOI: 10.1007/s10817-021-09600-4.

[222] Grigori S. Tseitin. "On the Complexity of Derivation in Propositional Calculus". In: *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*. Symbolic Computation. Springer, 1983, pp. 466–483. DOI: 10.1007/978-3-642-81955-1_28.

[223] Nestan Tsiskaridze, Clark Barrett, and Cesare Tinelli. "Generalized Optimization Modulo Theories". In: *International Joint Conference on Automated Reasoning*. Springer, 2024, pp. 458–479. DOI: 10.1007/978-3-031-63498-7_27.

[224] Alessandro Valentini, Andrea Micheli, and Alessandro Cimatti. "Temporal Planning with Intermediate Conditions and Effects". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 2020, pp. 9975–9982. DOI: 10.1609/aaai.v34i06.6553.

[225] Jaco van de Pol and Olga Tveretina. "A BDD-Representation for the Logic of Equality and Uninterpreted Functions". In: *Mathematical Foundations of Computer Science*. LNCS. Springer, 2005, pp. 769–780. DOI: 10.1007/11549345_66.

[226] Guy Van den Broeck and Dan Suciu. "Query Processing on Probabilistic Data: A Survey". In: *Foundations and Trends in Databases* 7.3-4 (2017), pp. 197–341. DOI: 10.1561/1900000052.

[227] Guy Van den Broeck, Nima Taghipour, Wannes Meert, Jesse Davis, and Luc De Raedt. "Lifted Probabilistic Inference by First-Order Knowledge Compilation". In: *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*. IJCAI'11. AAAI Press, 2011, pp. 2178–2185. DOI: 10.5591/978-1-57735-516-8/IJCAI11-363.

[228] Miroslav N. Velev and Randal E. Bryant. "Effective Use of Boolean Satisfiability Procedures in the Formal Verification of Superscalar and VLIW". In: *Proceedings of the 38th Annual Design Automation Conference*. ACM, 2001, pp. 226–231. DOI: 10.1145/378239.378469.

[229] David J. Wales and Jonathan P. K. Doye. "Global Optimization by Basin-Hopping and the Lowest Energy Structures of Lennard-Jones Clusters Containing up to 110 Atoms". In: *The Journal of Physical Chemistry A* 101.28 (1997), pp. 5111–5116. DOI: 10.1021/jp970984n.

[230] Steven A. Wolfman and Daniel S. Weld. "The LPSAT Engine & Its Application to Resource Planning". In: *Proceedings of the 16th International Joint Conference on Artificial Intelligence*. IJCAI'99. Morgan Kaufmann Publishers Inc., 1999, pp. 310–316. URL: https://www.ijcai.org/Proceedings/99-1/Papers/046.pdf.

[231] Wolfram. *Exact Global Optimization*. URL: https://reference.wolfram.com/language/tutorial/ConstrainedOptimizationExact.html.en.

[232] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Broeck. "A Semantic Loss Function for Deep Learning with Symbolic Knowledge". In: *Proceedings of the 35th International Conference on Machine Learning*. PMLR, 2018, pp. 5502–5511. URL: https://proceedings.mlr.press/v80/xu18h.html.

[233] Zhenghang Xu, Minghao Yin, and Jean Marie Lagniez. "An Embarrassingly Parallel Model Counter". In: *Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning*. Vol. 22. 2025, pp. 670–681. DOI: 10.24963/kr.2025/65.

[234] Zhijiang Ye, Buser Say, and Scott Sanner. "Symbolic Bucket Elimination for Piecewise Continuous Constrained Optimization". In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Vol. 10848. LNCS. Springer, 2018, pp. 585–594. DOI: 10.1007/978-3-319-93031-2_42.

[235] Yinlei Yu, Pramod Subramanyan, Nestan Tsiskaridze, and Sharad Malik. "All-SAT Using Minimal Blocking Clauses". In: *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*. 2014, pp. 86–91. DOI: 10.1109/VLSID.2014.22.

[236] Zahra Zamani, Scott Sanner, and Cheng Fang. "Symbolic Dynamic Programming for Continuous State and Action MDPs". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 26. 2012, pp. 1839–1845. DOI: 10.1609/aaai.v26i1.8372.

[237] Zhe Zeng, Paolo Morettin, Fanqi Yan, Antonio Vergari, and Guy Van den Broeck. "Probabilistic Inference with Algebraic Constraints: Theoretical Limits and Practical Approximations". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 11564–11575. URL: https://proceedings.neurips.cc/paper/2020/hash/85934679f30131d812a8c7475a7d0f74-Abstract.html.

[238] Zhe Zeng, Paolo Morettin, Fanqi Yan, Antonio Vergari, and Guy Van den Broeck. "Scaling up Hybrid Probabilistic Inference with Logical and Arithmetic Constraints via Message Passing". In: *Proceedings of the 37th International Conference on Machine Learning*. PMLR, 2020, pp. 10990–11000. URL: https://proceedings.mlr.press/v119/zeng20a.html.

[239] Zhe Zeng and Guy Van den Broeck. "Efficient Search-Based Weighted Model Integration". In: *35th Conference on Uncertainty in Artificial Intelligence*. PMLR, 2020, pp. 175–185. URL: https://proceedings.mlr.press/v115/zeng20a.html.

[240] Yueling Zhang, Geguang Pu, and Jun Sun. "Accelerating All-SAT Computation with Short Blocking Clauses". In: *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. ACM, 2020, pp. 6–17. DOI: 10.1145/3324884.3416569.

[241] Min Zhou, Fei He, Xiaoyu Song, Shi He, Gangyi Chen, and Ming Gu. "Estimating the Volume of Solution Space for Satisfiability Modulo Linear Real Arithmetic". In: *Theory of Computing Systems* 56.2 (2015), pp. 347–371. DOI: 10.1007/s00224-014-9553-9.

Part III

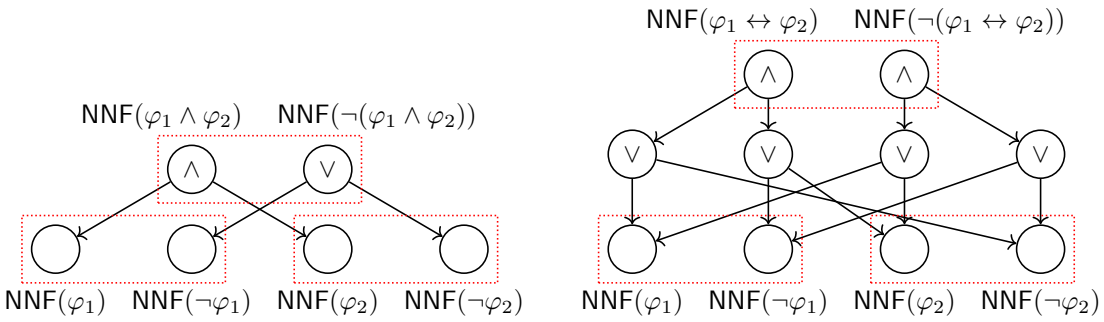
Appendices

Appendix A

On CNF Conversion for SAT and SMT Enumeration

A.1 Proofs

A.1.1 Proof of Lemma 1.2.1



(a) Case $\varphi \stackrel{\text{def}}{=} \varphi_1 \wedge \varphi_2$.

(b) Case $\varphi \stackrel{\text{def}}{=} \varphi_1 \leftrightarrow \varphi_2$.

Figure A.1: 2-root DAGs for the pair $\langle \mathsf{NNF}(\varphi), \mathsf{NNF}(\neg\varphi)\rangle$ for $\varphi \stackrel{\text{def}}{=} \varphi_1 \wedge \varphi_2$ and $\varphi \stackrel{\text{def}}{=} \varphi_1 \leftrightarrow \varphi_2$ (those for $\varphi_1 \vee \varphi_2$ and $\varphi_1 \rightarrow \varphi_2$ are similar to the one for $\varphi_1 \wedge \varphi_2$).

Proof. $\mathsf{NNF}(\varphi)$ is a subgraph of the 2-root DAG for the pair $\langle \mathsf{NNF}(\varphi), \mathsf{NNF}(\neg\varphi)\rangle$. We prove that the latter grows linearly in size w.r.t. φ by reasoning inductively on the structure of φ. (The size of a DAG is denoted by "$|\ldots|$".)

if φ is an atom: $\mathsf{NNF}(\varphi) = \varphi$ and $\mathsf{NNF}(\neg\varphi) = \neg\varphi$, so that: $|\langle \mathsf{NNF}(\varphi), \mathsf{NNF}(\neg\varphi)\rangle| = 2$.

if $\varphi \stackrel{\text{def}}{=} \neg\varphi_1$: we assume by induction that $|\langle \mathsf{NNF}(\varphi_1), \mathsf{NNF}(\neg\varphi_1)\rangle|$ is linear in $|\varphi_1|$. Then $\langle \mathsf{NNF}(\varphi), \mathsf{NNF}(\neg\varphi)\rangle = \langle \mathsf{NNF}(\neg\varphi_1), \mathsf{NNF}(\varphi_1)\rangle$ (i.e., we just invert the order),

so that

$$|\langle \mathsf{NNF}(\varphi), \mathsf{NNF}(\neg\varphi)\rangle| = |\langle \mathsf{NNF}(\neg\varphi_1), \mathsf{NNF}(\varphi_1)\rangle| = |\langle \mathsf{NNF}(\varphi_1), \mathsf{NNF}(\neg\varphi_1)\rangle| \tag{A.1}$$

if $\varphi \stackrel{\text{def}}{=} (\varphi_1 \bowtie \varphi_2)$ **s.t.** $\bowtie \in \{\wedge, \leftrightarrow\}$**:** we assume by induction that $|\langle \mathsf{NNF}(\varphi_1), \mathsf{NNF}(\neg\varphi_1)\rangle|$ and $|\langle \mathsf{NNF}(\varphi_2), \mathsf{NNF}(\neg\varphi_2)\rangle|$ are linear in $|\varphi_1|$ and $|\varphi_2|$, respectively. (See Figure A.1):

if $\bowtie$ **is** $\wedge$**:** the DAGs for $\mathsf{NNF}(\varphi), \mathsf{NNF}(\neg\varphi)$ add 2 "$\wedge/\vee$" nodes and $2+2$ arcs:
$\mathsf{NNF}(\quad \varphi_1 \wedge \varphi_2\ ) \stackrel{\text{def}}{=} \mathsf{NNF}(\quad \varphi_1) \wedge \mathsf{NNF}(\quad \varphi_2)$ and
$\mathsf{NNF}(\neg(\varphi_1 \wedge \varphi_2)) \stackrel{\text{def}}{=} \mathsf{NNF}(\neg\varphi_1) \vee \mathsf{NNF}(\neg\varphi_2)$, thus

$$|\langle \mathsf{NNF}(\varphi), \mathsf{NNF}(\neg\varphi)\rangle| = 6{+}|\langle \mathsf{NNF}(\varphi_1), \mathsf{NNF}(\neg\varphi_1)\rangle|{+}|\langle \mathsf{NNF}(\varphi_2), \mathsf{NNF}(\neg\varphi_2)\rangle| \tag{A.2}$$

if $\bowtie$ **is** $\leftrightarrow$**:** the DAGs for $\mathsf{NNF}(\varphi), \mathsf{NNF}(\neg\varphi)$ add 3+3 "$\wedge$"/"$\vee$" nodes and $6+6$ arcs:
$\mathsf{NNF}(\quad \varphi_1 \leftrightarrow \varphi_2\ ) \stackrel{\text{def}}{=} (\mathsf{NNF}(\neg\varphi_1) \vee \mathsf{NNF}(\varphi_2)) \wedge (\mathsf{NNF}(\quad \varphi_1) \vee \mathsf{NNF}(\neg\varphi_2))$ and
$\mathsf{NNF}(\neg(\varphi_1 \leftrightarrow \varphi_2)) \stackrel{\text{def}}{=} (\mathsf{NNF}(\quad \varphi_1) \vee \mathsf{NNF}(\varphi_2)) \wedge (\mathsf{NNF}(\neg\varphi_1) \vee \mathsf{NNF}(\neg\varphi_2))$, thus

$$|\langle \mathsf{NNF}(\varphi), \mathsf{NNF}(\neg\varphi)\rangle| = 18{+}|\langle \mathsf{NNF}(\varphi_1), \mathsf{NNF}(\neg\varphi_1)\rangle|{+}|\langle \mathsf{NNF}(\varphi_2), \mathsf{NNF}(\neg\varphi_2)\rangle| \tag{A.3}$$

if $\varphi \stackrel{\text{def}}{=} (\varphi_1 \bowtie \varphi_2)$ **s.t.** $\bowtie \in \{\vee, \rightarrow\}$**:** these cases can be reduced to the previous cases, since

$$\mathsf{NNF}(\varphi_1 \vee \varphi_2) = \mathsf{NNF}(\neg(\neg\varphi_1 \wedge \neg\varphi_2)) \text{ and } \mathsf{NNF}(\varphi_1 \rightarrow \varphi_2) = \mathsf{NNF}(\neg(\varphi_1 \wedge \neg\varphi_2)).$$

Therefore, from Equations (A.1) to (A.3), we have that $|\langle \mathsf{NNF}(\varphi), \mathsf{NNF}(\neg\varphi)\rangle|$ is $O(|\varphi_|)$. □

A.1.2 Proof of Lemma 3.2.1

$\varphi_1\|_\mu$	$\top$	$\top$	$\top$	$*$	$*$	$*$	$\bot$	$\bot$	$\bot$
$\varphi_2\|_\mu$	$\top$	$*$	$\bot$	$\top$	$*$	$\bot$	$\top$	$*$	$\bot$
$\neg(\varphi_1\|_\mu)$	$\bot$	$\bot$	$\bot$	$*$	$*$	$*$	$\top$	$\top$	$\top$
$\varphi_1\|_\mu \wedge \varphi_2\|_\mu$	$\top$	$*$	$\bot$	$*$	$*$	$\bot$	$\bot$	$\bot$	$\bot$
$\varphi_1\|_\mu \vee \varphi_2\|_\mu$	$\top$	$\top$	$\top$	$\top$	$*$	$*$	$\top$	$*$	$\bot$
$\varphi_1\|_\mu \rightarrow \varphi_2\|_\mu$	$\top$	$*$	$\bot$	$\top$	$*$	$*$	$\top$	$\top$	$\top$
$\varphi_1\|_\mu \leftrightarrow \varphi_2\|_\mu$	$\top$	$*$	$\bot$	$*$	$*$	$*$	$\bot$	$*$	$\top$

Figure A.2: Three-valued semantics of $\varphi|_\mu$ in terms of $\{\top, \bot, *\}$ ("true", "false", "unknown").

In the following, the symbol "$*$" denotes any formula that is not in $\{\top, \bot\}$. Following [193, 194], we adopt a three-valued semantics for residuals $\varphi|_\mu \in \{\top, \bot, *\}$, so that "$\varphi|_\mu = *$" means "$\varphi|_\mu \notin \{\top, \bot\}$" and "$\varphi_1|_\mu = \varphi_2|_\mu$" means that the two residuals $\varphi_1|_\mu$ and $\varphi_2|_\mu$ are either both $\top$, both $\bot$, or neither is in $\{\top, \bot\}$. (Notice that, in the latter case, $\varphi_1|_\mu = \varphi_2|_\mu$ even if $\varphi_1|_\mu$ and $\varphi_2|_\mu$ are different formulas.) We extend the definition to tuples in an obvious way: $\langle \varphi_1|_\mu, \ldots, \varphi_n|_\mu \rangle = \langle \psi_1|_\mu, \ldots, \psi_n|_\mu \rangle$ iff $\varphi_i|_\mu = \psi_i|_\mu$ for each $i \in [1 \ldots n]$.

The three-valued semantics of the Boolean operators is reported for convenience in Figure A.2. As a straightforward consequence of the above semantics, we have:

$$(\neg\varphi)|_\mu = \neg(\varphi|_\mu) \quad \text{(hereafter simply "}\neg\varphi|_\mu\text{")}$$
$$(\varphi_1 \bowtie \varphi_2)|_\mu = \varphi_1|_\mu \bowtie \varphi_2|_\mu \quad \text{for } \bowtie \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}.$$

Also, the usual transformations apply: $\neg(\varphi_1 \wedge \varphi_2)|_\mu = \neg\varphi_1|_\mu \vee \neg\varphi_2|_\mu$, $\neg(\varphi_1 \vee \varphi_2)|_\mu = \neg\varphi_1|_\mu \wedge \neg\varphi_2|_\mu$, $(\varphi_1 \leftrightarrow \varphi_2)|_\mu = (\neg\varphi_1|_\mu \vee \varphi_2|_\mu) \wedge (\varphi_1|_\mu \vee \neg\varphi_2|_\mu)$ and $\neg(\varphi_1 \leftrightarrow \varphi_2)|_\mu = (\varphi_1|_\mu \vee \varphi_2|_\mu) \wedge (\neg\varphi_1|_\mu \vee \neg\varphi_2|_\mu)$. For convenience, we sometimes denote the complement of $v \in \{\top, \bot, *\}$ by $\bar{v}$, i.e., $\bar{v} \stackrel{\text{def}}{=} \neg v$, so that $\bar{\top} = \bot, \bar{\bot} = \top, \bar{*} = *$.

We prove the following lemma, from which Lemma 3.2.1 follows directly.

Lemma A.1.1. *Consider a formula φ, and a partial assignment μ. Then:*

$$\langle \varphi|_\mu, \neg\varphi|_\mu \rangle = \langle \mathsf{NNF}(\varphi)|_\mu, \mathsf{NNF}(\neg\varphi)|_\mu \rangle. \tag{A.4}$$

Proof. As in Appendix A.1.1, we prove this fact by reasoning on the 2-root DAG for the pair $\langle \mathsf{NNF}(\varphi), \mathsf{NNF}(\neg\varphi) \rangle$. Specifically, we prove Equation (A.4) by induction on the structure of φ.

if φ is an atom: then $\mathsf{NNF}(\varphi) = \varphi$ and $\mathsf{NNF}(\neg\varphi) = \neg\varphi$, so that $\varphi|_\mu = \mathsf{NNF}(\varphi)|_\mu$ and $\neg\varphi|_\mu = \mathsf{NNF}(\neg\varphi)|_\mu$.

if $\varphi \stackrel{\text{def}}{=} \neg\varphi_1$**:** we assume by induction that $\langle \varphi_1|_\mu, \neg\varphi_1|_\mu\rangle = \langle \mathsf{NNF}(\varphi_1)|_\mu, \mathsf{NNF}(\neg\varphi_1)|_\mu\rangle$. Let v be s.t. $\langle \varphi|_\mu, \neg\varphi|_\mu\rangle = \langle v, \bar{v}\rangle$. Then:

$$\begin{aligned}
\langle \neg\varphi_1|_\mu,\ \ \varphi_1|_\mu\rangle = \langle v, \bar{v}\rangle &\Leftrightarrow \\
\langle\ \ \varphi_1|_\mu, \neg\varphi_1|_\mu\rangle = \langle \bar{v}, v\rangle &\stackrel{\text{ind}}{\Leftrightarrow} \\
\langle \mathsf{NNF}(\ \ \varphi_1)|_\mu, \mathsf{NNF}(\neg\varphi_1)|_\mu\rangle = \langle \bar{v}, v\rangle &\Leftrightarrow \\
\langle \mathsf{NNF}(\neg\varphi_1)|_\mu, \mathsf{NNF}(\ \ \varphi_1)|_\mu\rangle = \langle v, \bar{v}\rangle &
\end{aligned}$$

if $\varphi \stackrel{\text{def}}{=} (\varphi_1 \bowtie \varphi_2)$ **s.t.** $\bowtie \in \{\wedge, \leftrightarrow\}$**:** we assume by induction that

$$\begin{aligned}
\langle \varphi_1|_\mu, \neg\varphi_1|_\mu\rangle &= \langle \mathsf{NNF}(\varphi_1)|_\mu, \mathsf{NNF}(\neg\varphi_1)|_\mu\rangle, \\
\langle \varphi_2|_\mu, \neg\varphi_2|_\mu\rangle &= \langle \mathsf{NNF}(\varphi_2)|_\mu, \mathsf{NNF}(\neg\varphi_2)|_\mu\rangle.
\end{aligned}$$

Then,

if $\bowtie$ **is** $\wedge$**:**

$$\begin{aligned}
\langle (\varphi_1 \wedge \varphi_2)|_\mu, \neg(\varphi_1 \wedge \varphi_2)|_\mu\rangle &= \\
\langle \varphi_1|_\mu \wedge \varphi_2|_\mu, \neg\varphi_1|_\mu \vee \neg\varphi_2|_\mu\rangle &\stackrel{\text{ind}}{=} \\
\langle \mathsf{NNF}(\varphi_1)|_\mu \wedge \mathsf{NNF}(\varphi_2)|_\mu, \mathsf{NNF}(\neg\varphi_1)|_\mu \vee \mathsf{NNF}(\neg\varphi_2)|_\mu\rangle &= \\
\langle (\mathsf{NNF}(\varphi_1) \wedge \mathsf{NNF}(\varphi_2))|_\mu, (\mathsf{NNF}(\neg\varphi_1) \vee \mathsf{NNF}(\neg\varphi_2))|_\mu\rangle &= \\
\langle \mathsf{NNF}(\varphi_1 \wedge \varphi_2)|_\mu, \mathsf{NNF}(\neg(\varphi_1 \wedge \varphi_2))|_\mu\rangle &
\end{aligned}$$

if $\bowtie$ **is** $\leftrightarrow$**:**

$$\begin{aligned}
\langle (\varphi_1 \leftrightarrow \varphi_2)|_\mu, \neg(\varphi_1 \leftrightarrow \varphi_2)|_\mu\rangle &= \\
\langle ((\neg\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \neg\varphi_2))|_\mu, ((\varphi_1 \vee \varphi_2) \wedge (\neg\varphi_1 \vee \neg\varphi_2))|_\mu\rangle &= \\
\langle (\neg\varphi_1|_\mu \vee \varphi_2|_\mu) \wedge (\varphi_1|_\mu \vee \neg\varphi_2|_\mu), (\varphi_1|_\mu \vee \varphi_2|_\mu) \wedge (\neg\varphi_1|_\mu \vee \neg\varphi_2|_\mu)\rangle &\stackrel{\text{ind}}{=} \\
\langle (\mathsf{NNF}(\neg\varphi_1)|_\mu \vee \mathsf{NNF}(\varphi_2)|_\mu) \wedge (\mathsf{NNF}(\varphi_1)|_\mu \vee \mathsf{NNF}(\neg\varphi_2)|_\mu), & \\
(\mathsf{NNF}(\varphi_1)|_\mu \vee \mathsf{NNF}(\varphi_2)|_\mu) \wedge (\mathsf{NNF}(\neg\varphi_1)|_\mu \vee \mathsf{NNF}(\neg\varphi_2)|_\mu)\rangle &= \\
\langle ((\mathsf{NNF}(\neg\varphi_1) \vee \mathsf{NNF}(\varphi_2)) \wedge (\mathsf{NNF}(\varphi_1) \vee \mathsf{NNF}(\neg\varphi_2)))|_\mu, & \\
((\mathsf{NNF}(\varphi_1) \vee \mathsf{NNF}(\varphi_2)) \wedge (\mathsf{NNF}(\neg\varphi_1) \vee \mathsf{NNF}(\neg\varphi_2)))|_\mu\rangle &= \\
\langle \mathsf{NNF}(\varphi_1 \leftrightarrow \varphi_2)|_\mu, \mathsf{NNF}(\neg(\varphi_1 \leftrightarrow \varphi_2))|_\mu\rangle. &
\end{aligned}$$

if $\varphi \stackrel{\text{def}}{=} (\varphi_1 \bowtie \varphi_2)$ **s.t.** $\bowtie \in \{\vee, \rightarrow\}$**:** these cases can be reduced to the previous cases, since $\mathsf{NNF}(\varphi_1 \vee \varphi_2) = \mathsf{NNF}(\neg(\neg\varphi_1 \wedge \neg\varphi_2))$ and $\mathsf{NNF}(\varphi_1 \rightarrow \varphi_2) = \mathsf{NNF}(\neg(\varphi_1 \wedge \neg\varphi_2))$.

□

Thus, $\langle \varphi|_\mu, \neg\varphi|_\mu\rangle = \langle \mathsf{NNF}(\varphi)|_\mu, \mathsf{NNF}(\neg\varphi)|_\mu\rangle$, so $\varphi|_\mu = v$ iff $\mathsf{NNF}(\varphi)|_\mu = v$ for every $v \in \{\top, \bot\}$; hence, Lemma 3.2.1 holds.

A.2 Additional Experimental Results

A.2.1 An Analysis of Candidate Solvers

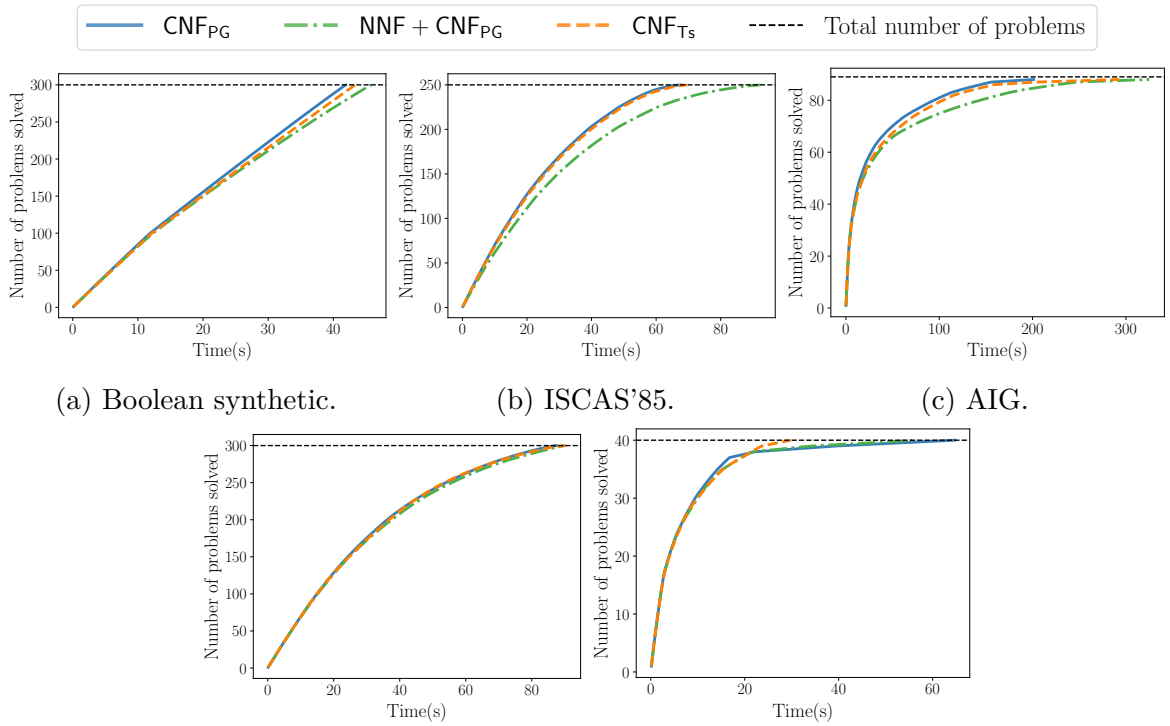
In Table A.1, we report an analysis of the features of every candidate AllSAT and $\mathcal{T}$-AllSAT solver, as discussed in Section 3.3.1.

		Required features					Mode		
	Solver	Available	CNF input	Projected	Partial	Minimal	Disjoint	Non-disjoint	Notes
AllSAT	RELSAT	✓	✓	✗	✗	✗	✓	✗	
	GRUMBERG	✗	✓	✓	✓	✓	✓	✓	Ex-post minim.
	SOLALL	?	✓	✓	✓	✗	✓	✗	Store in OBDD
	JIN	✗	✗	✗	✓	✓	✗	✓	AIG input
	CLASP	✓	✓	✓	✗	✗	✓	✗	ASP solver
	PICOSAT	✓	✓	✗	✗	✗	✓	✗	
	YU	?	✓	✗	✓	✓	✓	✓	
	BC	✓	✓	✗	✓	✓	✓	✓	
	NBC	✓	✓	✗	✗	✗	✓	✗	
	BDD	✓	✓	✗	✓	✗	✗	✓	Store in OBDD
	DEPBDD	✗	✓	✓	✓	✗	✓	✗	Store in OBDD
	DUALIZA	✓	✓	✓	✓	✗	✓	✓	Dual reasoning
	BASOLVER	✗	✓	✗	✓	✓	✗	✓	
	ALLSATCC	✗	✓	✗	✓	✓	✓	✗	
	HALL	✓	✗	✗	✓	✓	✓	✓	AIG input
	TABULARALLSAT	✓	✓	✓	✓	✗	✓	✗	
	DECDNNF	✓	✗	✗	✓	✗	✓	✗	d-DNNF input
$\mathcal{T}$-AllSAT	MATHSAT	✓	✓	✓	✓	✓	✓	✓	
	AZ3	✓	✓	✓	✗	✗	✓	✗	
	TABULARALLSMT	✓	✓	✓	✓	✗	✓	✗	

Table A.1: List of AllSAT and $\mathcal{T}$-AllSAT solvers (rows) and supported features (columns). (In the "Available" column, "✗" indicates that the authors confirmed to us the unavailability of their tool, whereas "?" indicates that they did not reply to our inquiry.)

A.2.2 Experimental Results on Plain SAT and SMT Solving

In this section, we report the results of the experiments on plain SAT and SMT solving with MathSAT using the different CNF-izations. The plots in Figure A.3 show that $\mathsf{NNF}{+}\mathsf{CNF}_{\mathsf{PG}}$ does not bring any advantage for plain SAT solving, supporting the analysis in Section 3.3.5.



(a) Boolean synthetic. (b) ISCAS'85. (c) AIG.

(d) SMT($\mathcal{LRA}$) synthetic. (e) WMI.

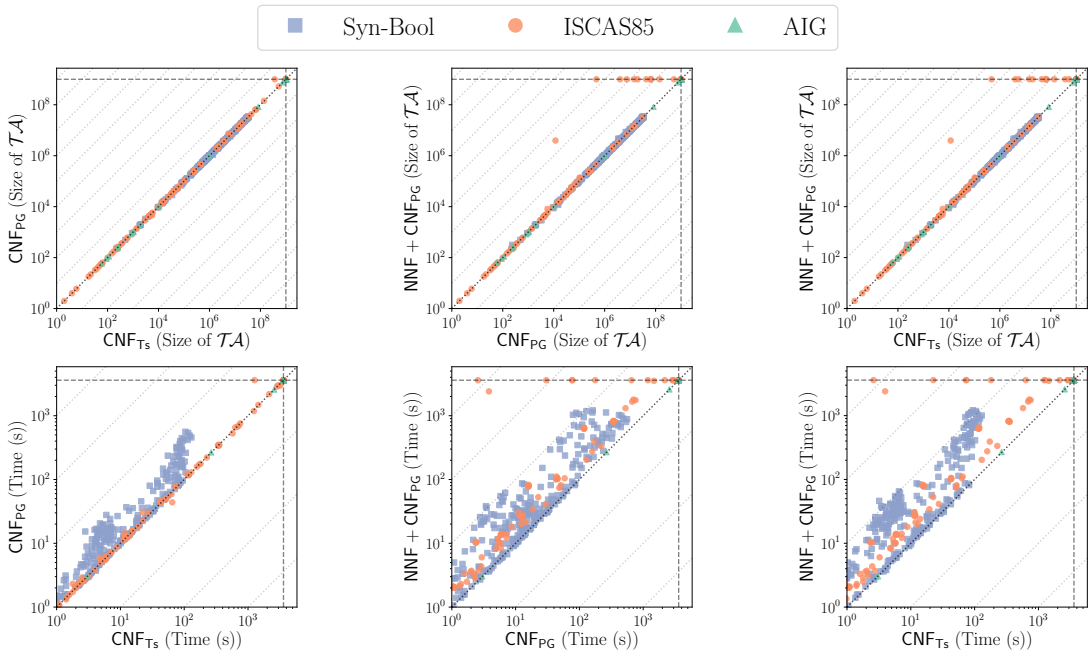
Figure A.3: Time taken for plain SAT and SMT solving using the different CNF transformations. The y-axis reports the number of instances for which the solver finished within the cumulative time on the x-axis.

A.2.3 Experimental Results on d-DNNF-based Enumeration

The authors of [125] recently presented a tool named DECDNNF to enumerate all models of a d-DNNF formula [62].

Following the D4+DECDNNF approach [125], we first compile the CNF-ization of the input formula into a d-DNNF using the D4 compiler [126], and then enumerate its models using DECDNNF. To avoid enumerating assignments to CNF labels, we perform projected d-DNNF compilation onto the atoms $\mathbf{A}$ of the input formula, and then use DECDNNF to enumerate the models of the projected d-DNNF.

The results of the experiments on the Boolean benchmarks are shown in Figure A.4. We see that $\mathsf{CNF}_{\mathsf{Ts}}$ is uniformly the best-performing CNF-ization, supporting the analysis in Section 3.3.5.



(a) Results for disjoint enumeration.

Bench.	Instances	T.O. for disjoint AllSAT		
		$\mathsf{CNF}_{\mathsf{Ts}}$	$\mathsf{CNF}_{\mathsf{PG}}$	NNF+$\mathsf{CNF}_{\mathsf{PG}}$
Syn-Bool	300	**0**	**0**	**0**
ISCAS85	250	**15**	16	27
AIG	89	**76**	**76**	**76**

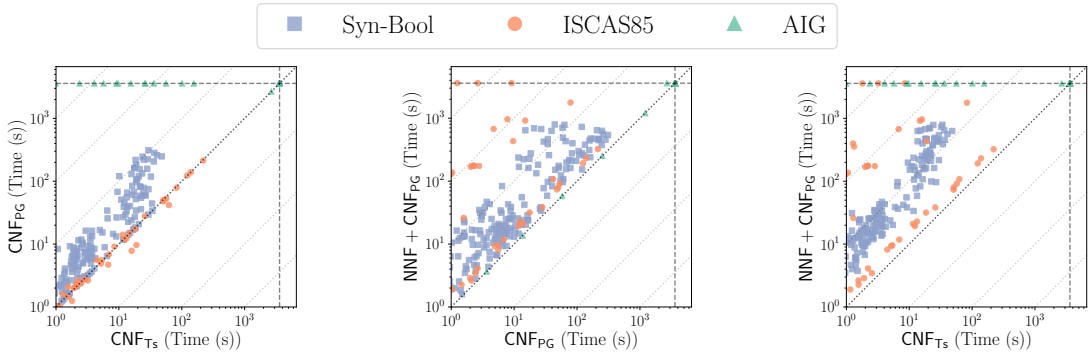
(b) Number of timeouts.

Figure A.4: Results on the Boolean benchmarks using D4+DECDNNF. Plots in A.4a compare CNF-izations by $\mathcal{TA}$ size (first row) and execution time (second row). Points on dashed lines represent timeouts, shown in A.4b. All axes use a logarithmic scale.

A.2.4 Experimental Results on d-DNNF-based Model Counting

We also tested the different CNF-izations for model counting using D4. Since $\mathsf{CNF}_{\mathsf{PG}}$ and $\mathsf{NNF}{+}\mathsf{CNF}_{\mathsf{PG}}$ do not preserve the model count, we perform projected model counting on the original atoms of the input formula. Even though this would not be necessary for $\mathsf{CNF}_{\mathsf{Ts}}$, we apply the same procedure to ensure a fair comparison.

The results of the experiments on the Boolean benchmarks are shown in Figure A.5. We also observe that, for model counting, $\mathsf{CNF}_{\mathsf{Ts}}$ is uniformly the best-performing CNF-ization, supporting the analysis in Section 3.3.5.



(a) Results for model counting.

Bench.	Instances	T.O. for model counting $\mathsf{CNF}_{\mathsf{Ts}}$	$\mathsf{CNF}_{\mathsf{PG}}$	$\mathsf{NNF}{+}\mathsf{CNF}_{\mathsf{PG}}$
Syn-Bool	300	**0**	**0**	**0**
ISCAS85	250	**0**	**0**	4
AIG	89	**50**	67	68

(b) Number of timeouts.

Figure A.5: Results on the Boolean benchmarks using D4 for model counting. Plots in A.5a compare CNF-izations by execution time. Points on dashed lines represent timeouts, shown in A.5b. All axes use a logarithmic scale.

Appendix B

SMT-Enumeration Techniques for Structure-Aware WMI

B.1 Description of the datasets for the DET experiments

Table B.1 describes the datasets used to train the DETs considered in Section 4.3.2. For each dataset, we report the number of Boolean and continuous variables, the sizes of the training and validation splits, and the size of the resulting DET in terms of the number of internal nodes (conditions).

B.2 Details on the program fairness verification task

The population model pop (Algorithm 17) is a probabilistic program with Gaussian priors, modeling a distribution over real-valued variables: (i) *ethnicity*; (ii) *colRank* (rank of the college attended by the person, where lower values are better); (iii) *yExp* (years of work experience). In this population model, *colRank* is influenced by *ethnicity*. The decision program dec (Algorithm 18) takes *colRank* and *yExp* as input and outputs hire. Notice that, while dec does not have access to the sensitive condition minority $=$ *ethnicity* > 10, the program is still unfair (Figure 4.8) due to the high importance attributed to *colRank*. This can be amended by modifying the second condition to $(5 \cdot yExp - colRank > -5)$, putting more emphasis on the years of work experience. This modification indeed makes the program satisfy the fairness criterion with $\varepsilon = 0.1$ given pop.

Dataset	$\lvert\mathbf{A}\rvert$	$\lvert\mathbf{x}\rvert$	# Train	# Valid	Size
balance-scale	3	4	1875	205	5
iris	3	4	450	50	3
cars	33	7	2115	234	14
diabetes	1	8	4149	459	28
breast-cancer	12	4	1650	180	13
glass2	1	9	970	100	7
glass	7	9	1280	140	7
breast	1	10	4521	495	31
solar	25	3	2522	273	13
cleve	17	6	2492	266	18
hepatitis	14	6	940	100	5
heart	3	11	2268	252	16
australian	34	6	6210	690	46
crx	38	6	6688	736	46
german	41	10	12600	1386	93
german-org	13	12	15000	1650	108
auto	56	16	2522	260	18
anneal-U	74	9	21021	2301	144

Table B.1: UCI datasets considered in our experiments. For each dataset, we report the number of Boolean and continuous variables, the numbers of training and validation instances, and the size of the resulting density estimation tree in terms of the number of conditions.

Algorithm 17 `pop` ()
Output: *colRank*: college rank, *yExp*: years of experience

ethnicity $\leftarrow \mathcal{N}(0, 10)$
colRank $\leftarrow \mathcal{N}(25, 10)$
yExp $\leftarrow \mathcal{N}(10, 5)$
if *ethnicity* > 10 **then**
  colRank $\leftarrow$ *colRank* $+ 5$
end if
return *colRank*, *yExp*

Algorithm 18 dec (*colRank*, *yExp*)
Input: *colRank*: college rank, *yExp*: years of experience
Output: hire: hiring decision

```
1: if (colRank ≤ 5) then
2:     hire ← True
3: else if (yExp − colRank > −5) then
4:     hire ← True
5: else
6:     hire ← False
7: end if
```

Appendix C

SMT Enumeration for MAP Inference with Non-Convex Constraints

C.1 Particle, Constraint-Aware Adam Optimizer

We provide details on the particle-based, constraint-aware version of Adam [118], dubbed PCAdam, that we present as a side contribution. We use PCAdam both as a baseline to compare with, as well as a convex-polytope optimizer combined with PaMap in the data imputation experiments in Section 6.3.3. Given the constraints φ, the density p, and an initial batch of points $\mathcal{X}_0 = \{\mathbf{x}_1^{\mathcal{M}}, \ldots, \mathbf{x}_n^{\mathcal{M}}\}$, PCAdam keeps track of the best feasible solution found so far (lines 1-2). Then, for a fixed number of iterations, it updates the batch of points using Adam (line 5). Next, it checks which of the new points satisfy the constraints (line 7) and updates the best solution found so far accordingly (lines 8-9). Finally, it returns the best solution found (line 13).

Algorithm 19 PCAdam(φ, p, *it*, $\mathcal{X}_0$)
Input: φ: $\mathcal{LRA}$-formula; p: density function; *it*: iterations; $\mathcal{X}_0$: initial batch of points.
Output: lb: max. density; $\mathcal{M}$: the model that achieves it.

1: $\mathsf{lb} \leftarrow -\infty$
2: $\mathcal{M}^\star \leftarrow \emptyset$
3: $\mathcal{X} \leftarrow \mathcal{X}_0$
4: **for** $i = 1$ to *it* **do**
5: $\quad \mathcal{X} \leftarrow \mathsf{update}(\mathcal{X}, \nabla p)$ {Gradient-based step}
6: $\quad$ **for each** $\mathbf{x}^{\mathcal{M}} \in \mathcal{X}$ **do**
7: $\quad\quad$ **if** $\mathbf{x}^{\mathcal{M}} \models \varphi$ **and** $p(\mathbf{x}^{\mathcal{M}}) > \mathsf{lb}$ **then**
8: $\quad\quad\quad \mathsf{lb} \leftarrow p(\mathbf{x}^{\mathcal{M}})$
9: $\quad\quad\quad \mathcal{M}^\star \leftarrow \mathbf{x}^{\mathcal{M}}$
10: $\quad\quad$ **end if**
11: $\quad$ **end for**
12: **end for**
13: **return** $\mathcal{M}^\star, \mathsf{lb}$

C.2 Details for SDD Experiments

We provide additional experiments on the SDD dataset and on the configuration of the optimization algorithms used in our experiments. We recall that for this dataset, we learn the densities with PAL [124]. PAL densities are non-negative piecewise polynomials, each piece being an axis-aligned box

$$\Phi = [l_1, u_1] \times \cdots \times [l_N, u_N] \subseteq \mathbb{R}^N \tag{C.1}$$

over which the density is defined as

$$p_\Phi(\mathbf{x}) = \sum_{i=1}^{d} \alpha_i \prod_{j=1}^{N} s_j(x_j)^2, \tag{C.2}$$

where each $s_j(x_j)$ is a cubic polynomial.

C.3 PaMap

C.3.1 Enumeration and Upper Bounds

For these densities, we implement the enumerator as follows. First, we compute the upper bound for each spline piece (see below). Then we sort the pieces in decreasing order of their upper bounds to increase the chances of pruning suboptimal pieces early. Finally, we enumerate polytopes corresponding to each piece in turn, terminating when the upper bound of the current piece is lower than the current best value.

An upper bound on the maximum of Equation (C.2) over a piece Φ can be computed as follows:

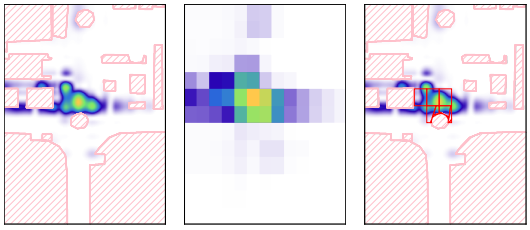
$$\max_{\mathbf{x}\in\Phi} p_\Phi(\mathbf{x}) \leq \sum_{i=1}^{d} \alpha_i \cdot \max_{\mathbf{x}\in\Phi} \prod_{j=1}^{N} s_j(x_j)^2 \tag{C.3}$$

$$= \sum_{i=1}^{d} \alpha_i \prod_{j=1}^{N} \max_{x_j\in[l_j,u_j]} s_j(x_j)^2 \tag{C.4}$$

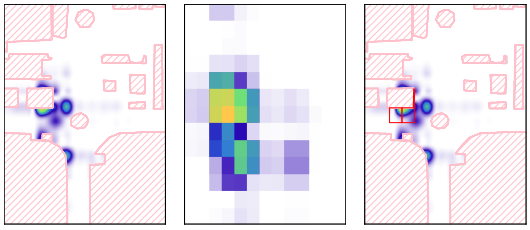
The critical points of $s_j(x_j)^2$ are the same as those of $s_j(x_j)$, which can be computed in closed form. Thus, we evaluate $s_j(x_j)$ at its critical points within $[l_j, u_j]$, as well as at the interval boundaries l_j and u_j. The maximum squared value among these points provides the desired bound. Figure C.1 shows how pruning works in practice on 10 sample trajectories from the SDD dataset.

C.3.2 SciPy's Optimizers

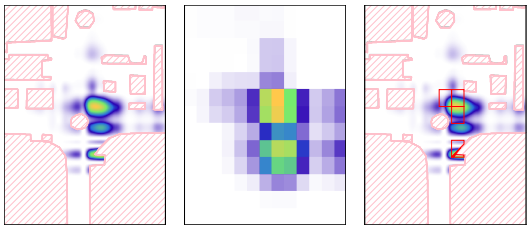
For PaMap(SciPy), we tried all combinations of global and local constrained optimizers available in SciPy's optimize library. The results are shown in Figure C.2a.



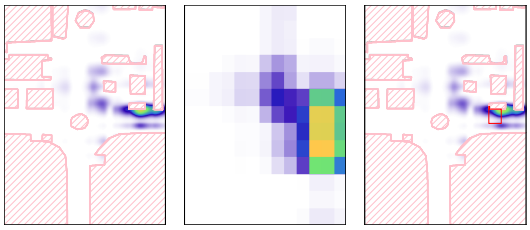
Traj. 2000. Enum. polytopes: 12/257



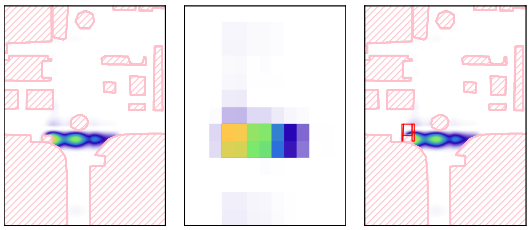
Traj. 3200. Enum. polytopes: 4/257



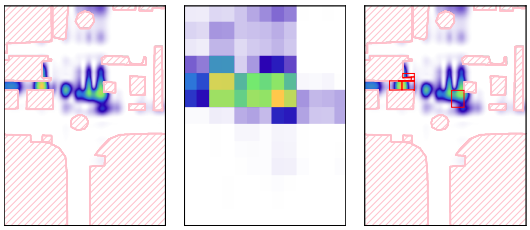
Traj. 6000. Enum. polytopes: 6/257



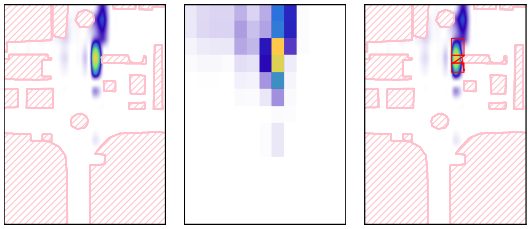
Traj. 9000. Enum. polytopes: 2/257



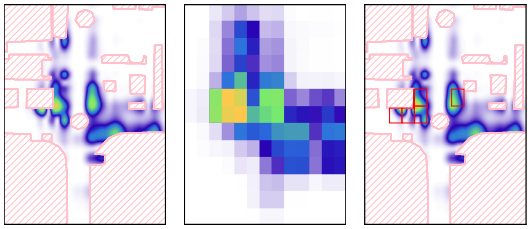
Traj. 13000. Enum. polytopes: 3/257



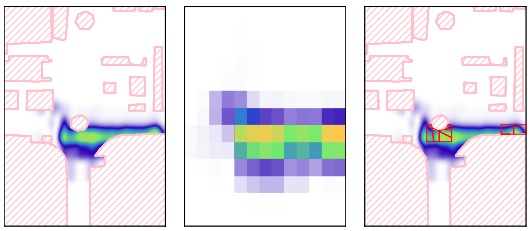
Traj. 17000. Enum. polytopes: 9/257



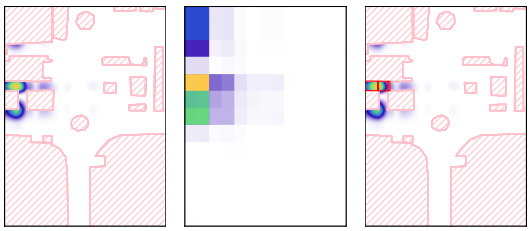
Traj. 22000. Enum. polytopes: 6/257



Traj. 29300. Enum. polytopes: 7/257



Traj. 35200. Enum. polytopes: 10/257



Traj. 41600. Enum. polytopes: 4/257

Figure C.1: Visualization of the PaMap pruning process across 10 sample trajectories. Each group shows the density (left), the computed upper bounds used for pruning (center), and the polytopes analyzed by PaMap(SHGO) thanks to pruning (right).

From the plot, we can see that the fastest configuration combines SHGO as the global optimizer with SLSQP as the local optimizer while achieving a relative error comparable to that of the other configurations. Hence, this is the configuration we used in our experiments.

C.4 PCAdam

To better understand the performance of PCAdam, we conduct experiments across different hyperparameter configurations. We fix the learning rate to 0.1 and vary:

- N: number of parallel particles used in the optimization;
- *it*: number of iterations for each particle;

The results are shown in Figure C.2b. As expected, all configurations report feasible solutions, but the performance varies significantly. Since the SDD dataset is low-dimensional (2D) and the feasible region is broad, PCAdam can find good solutions if enough particles and iterations are used. However, this comes at the cost of increased computation time.

C.5 OMT Encoding

OMT($\mathcal{NRA}$) solvers require the optimization problem to be encoded as a pair of a logical formula and an objective function encoded as an $\mathcal{NRA}$-term. While the logical formula is directly given by the problem, multiple choices are possible for encoding the objective function (C.2).

A first possibility is to encode it as a nested if-then-else expression:

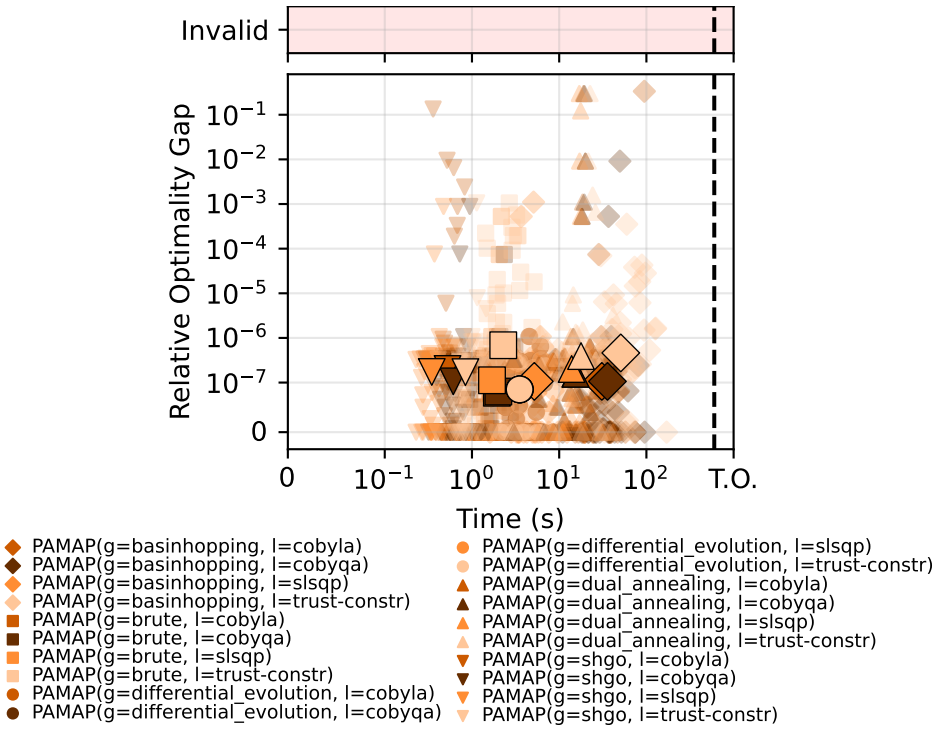
$$\mathrm{Ite}(\Phi_1, p_{\Phi_1}(\mathbf{x}^{\mathcal{M}}), \mathrm{Ite}(\Phi_2, p_{\Phi_2}(\mathbf{x}^{\mathcal{M}}), \ldots)) \tag{C.5}$$

where each Φ_i is a piece of the PAL density. An alternative is to encode the objective as a sum of if-then-else expressions:

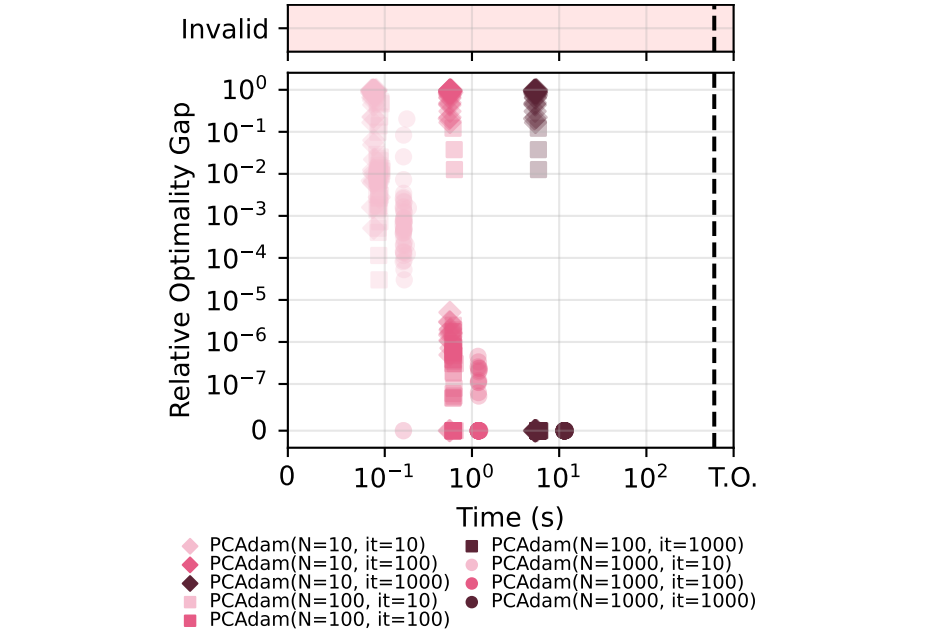
$$\sum_{\Phi_i} \mathrm{Ite}(\Phi_i, p_{\Phi_i}(\mathbf{x}^{\mathcal{M}}), 0). \tag{C.6}$$

To ensure that this encoding is well defined, the regions Φ_i must be mutually exclusive. We enforce this by defining each region using left-closed, right-open intervals, except for the final region along each dimension, which is closed on the right as well.

We tested both encodings using OptiMathSAT and CDCL-OCAC. Both OMT solvers timed out on the first encoding without finding a solution. With the second encoding, OptiMathSAT in anytime mode occasionally found a solution within the time limit and exited with an error on the remaining runs. In contrast, CDCL-OCAC timed out without finding any solution.



(a) PAMAP with different numerical optimizers from SciPy on the SDD dataset. Since the best configuration is not eye-catching, we also show the median point for each configuration.



(b) PCAdam configurations varying the number of particles and iterations on SDD.

Figure C.2: Experiments on the SDD dataset varying the numerical optimizers used by PAMAP (top) and the PCAdam hyperparameters (bottom).

Appendix D

A Theory-Agnostic Approach to Theory-Lemma Enumeration in SMT

D.1 Proofs

D.1.1 Proof of Theorem 7.2.1

Proof. Since $BC_{\boldsymbol{\alpha}^{\mathcal{T}}}(\varphi) \cup Cl_{\boldsymbol{\alpha}}$ rules out $P_{\boldsymbol{\alpha}}(\varphi)$, and since $BC_{\boldsymbol{\alpha}^{\mathcal{T}}}(\varphi)$ rules out only $\mathcal{T}$-satisfiable assignments on $\boldsymbol{\alpha}^{\mathcal{T}}$, every $\rho^{\boldsymbol{\alpha}} \stackrel{\text{def}}{=} \rho^{\mathcal{T}}[\boldsymbol{\alpha}^{\mathcal{T}}] \wedge \rho^{\mathbb{B}}[\boldsymbol{\alpha}^{\mathbb{B}}] \in P_{\boldsymbol{\alpha}}(\varphi)$ such that $\rho^{\mathcal{T}}[\boldsymbol{\alpha}^{\mathcal{T}}]$ is $\mathcal{T}$-unsatisfiable is ruled out by some $\mathcal{T}$-lemma in $Cl_{\boldsymbol{\alpha}}$.

Since $\rho^{\mathcal{T}}[\boldsymbol{\alpha}^{\mathcal{T}}] \wedge \rho^{\mathbb{B}}[\boldsymbol{\alpha}^{\mathbb{B}}]$ is $\mathcal{T}$-unsatisfiable iff $\rho^{\mathcal{T}}[\boldsymbol{\alpha}^{\mathcal{T}}]$ is $\mathcal{T}$-unsatisfiable, every $\rho^{\boldsymbol{\alpha}} \in P_{\boldsymbol{\alpha}}(\varphi)$ is ruled out by some $\mathcal{T}$-lemma in $Cl_{\boldsymbol{\alpha}}$.

□

D.1.2 Proof of Theorem 7.2.2

Proof. Since $BC_{\boldsymbol{\alpha}_1}(\varphi) \cup Cl_{\boldsymbol{\alpha}}$ rules out $P_{\boldsymbol{\alpha}}(\varphi)$, and since $BC_{\boldsymbol{\alpha}_1}(\varphi)$ rules out only $\mathcal{T}$-satisfiable assignments on $\boldsymbol{\alpha}_1$, every $\rho^{\boldsymbol{\alpha}} \stackrel{\text{def}}{=} \rho_1[\boldsymbol{\alpha}_1] \wedge \rho_2[\boldsymbol{\alpha}_2] \in P_{\boldsymbol{\alpha}}(\varphi)$ such that $\rho_1[\boldsymbol{\alpha}_1]$ is $\mathcal{T}$-unsatisfiable is ruled out by some $\mathcal{T}$-lemma in $Cl_{\boldsymbol{\alpha}}$.

Symmetrically, we have that every $\rho^{\boldsymbol{\alpha}} \stackrel{\text{def}}{=} \rho_1[\boldsymbol{\alpha}_1] \wedge \rho_2[\boldsymbol{\alpha}_2] \in P_{\boldsymbol{\alpha}}(\varphi)$ such that $\rho_2[\boldsymbol{\alpha}_2]$ is $\mathcal{T}$-unsatisfiable is ruled out by some $\mathcal{T}$-lemma in $Cl'_{\boldsymbol{\alpha}}$.

Thus, since $\boldsymbol{\alpha}_1$ and $\boldsymbol{\alpha}_2$ do not share variables or uninterpreted symbols, by Fact 7.2.1, $Cl_{\boldsymbol{\alpha}} \cup Cl'_{\boldsymbol{\alpha}}$ rules out $P_{\boldsymbol{\alpha}}(\varphi)$. □

D.2 Experimental Evaluation: Additional Details

D.2.1 Encoding of Numeric Planning Problems

In the original encoding [173], each action at a given time step is represented by a Boolean atom indicating whether the action is applied, together with additional atoms encoding the values of its parameters. However, when an action is not applied, the values assigned to its parameters are semantically irrelevant. As a consequence, two

truth assignments may differ only in the parameter values of an inactive action while still representing the same actual plan.

This can lead to overcounting in plan counting and enumeration. For example, if $a_{move_block_left,1}$ is false, then assignments that differ only in the value of $p_{move_block_left,1}$ should not be considered distinct plans, since the corresponding action is not executed.

To remove these spurious distinctions, we modify the encoding by forcing the parameters of inactive actions to take a fixed default value. In particular, for each action and each of its parameters, we add constraints of the following form:

$$\neg a_{move_block_left,1} \rightarrow (p_{move_block_left,1} = 0)$$

where 0 is the chosen default value for the parameter. This modification does not affect satisfiability because the parameter values of inactive actions do not influence the semantics of the plan. Therefore, every plan admitted by the original encoding is still admitted by the modified encoding, and vice versa. The only effect is that assignments that previously differed only in irrelevant parameter values are collapsed into a single canonical representative.

This change is mainly motivated by plan counting and enumeration, where we want the number of models to correspond more closely to the number of semantically distinct plans. In addition, the same constraints reduce the search space for $\mathcal{T}$-lemma enumeration.

Appendix E

d-DNNF Modulo Theories: A General Framework for Polytime SMT Transformations and Queries

E.1 A Survey of SMT-Level KC Compilers

In Table E.1, we summarize the SMT-level compilers analyzed in Section 2.7. Most tools are theory-specific, and potentially $\mathcal{T}$-canonical DDs are available only for $\mathcal{EUF}$, though their canonicity has not been proved. Except for LDDs [36], none of the analyzed tools is directly available.

	Compiler	**Theory**	**Avail.**	**$\mathcal{T}$-consistent**	**Canonical**
BDD	EQ-BDDs [84]	$\mathcal{E}$	*	✓	✗
	Goel-FM [96, 95]	$\mathcal{E}$	✗	✓	?
	Goel-e_{ij} [96, 95]	$\mathcal{E}$	✗	✗	✗
	Bryant [34]	$\mathcal{E}$	✗	✓	?
	EUF-BDDs [225]	$\mathcal{EUF}$	*	✓	✗
	(0,S,=)-BDDs [8]	$\{0, S, =\}$	✗	✓	✗
	DDD [158]	$\mathcal{DL}$	✗	✓	✗
	LDD [36]	$\mathcal{LIRA}$	✓	✗	✗
	Chan [40]	$\mathcal{NRA}$	✗	✓	✗
	haRVey [65]	any	✗	✓	✗
	Fontaine [81]	any	✗	✓	✗
	BDD+SMT [35, 46]	any	*	✓	✗
SDD	XSDD [72, 120]	$\mathcal{LRA}$	*	✗	✗

Table E.1: SMT-level KC compilers. In **Avail.**, ✓, ✗, and * mean, respectively, that the tool is publicly available, unavailable, or implemented within another tool and not usable. In **Canonical**, "?" means that the DDs may be canonical, but it was not proven.

E.2 Proofs

In what follows "Proposition" denotes facts that follow directly from the definitions, whereas "Lemma" and "Theorem" denote facts for which we provide a proof explicitly.

E.2.1 Some Auxiliary Propositions and Lemmas

Proposition E.2.1. *Let $\varphi[\boldsymbol{\alpha}]$, $\varphi_1[\boldsymbol{\alpha}]$, and $\varphi_2[\boldsymbol{\alpha}]$ be some $\mathcal{T}$-formulas. We have that:*

(a) $H_{\boldsymbol{\alpha}}(\varphi)$, $H_{\boldsymbol{\alpha}}(\neg\varphi)$, $P_{\boldsymbol{\alpha}}(\varphi)$, $P_{\boldsymbol{\alpha}}(\neg\varphi)$ *are all pairwise disjoint;*

(b) $H_{\boldsymbol{\alpha}}(\varphi) \cup H_{\boldsymbol{\alpha}}(\neg\varphi) \cup P_{\boldsymbol{\alpha}}(\varphi) \cup P_{\boldsymbol{\alpha}}(\neg\varphi) = H_{\boldsymbol{\alpha}}(\top) \cup P_{\boldsymbol{\alpha}}(\top) = 2^{\boldsymbol{\alpha}}$;

(c) $H_{\boldsymbol{\alpha}}(\varphi_1 \wedge \varphi_2) = H_{\boldsymbol{\alpha}}(\varphi_1) \cap H_{\boldsymbol{\alpha}}(\varphi_2)$; $P_{\boldsymbol{\alpha}}(\varphi_1 \wedge \varphi_2) = P_{\boldsymbol{\alpha}}(\varphi_1) \cap P_{\boldsymbol{\alpha}}(\varphi_2)$;
$H_{\boldsymbol{\alpha}}(\varphi_1 \vee \varphi_2) = H_{\boldsymbol{\alpha}}(\varphi_1) \cup H_{\boldsymbol{\alpha}}(\varphi_2)$; $P_{\boldsymbol{\alpha}}(\varphi_1 \vee \varphi_2) = P_{\boldsymbol{\alpha}}(\varphi_1) \cup P_{\boldsymbol{\alpha}}(\varphi_2)$;
$H_{\boldsymbol{\alpha}}(\neg\varphi) = H_{\boldsymbol{\alpha}}(\top) \setminus H_{\boldsymbol{\alpha}}(\varphi)$; $P_{\boldsymbol{\alpha}}(\neg\varphi) = P_{\boldsymbol{\alpha}}(\top) \setminus P_{\boldsymbol{\alpha}}(\varphi)$;

(d) φ *is $\mathcal{T}$-unsatisfiable if and only if* $H_{\boldsymbol{\alpha}}(\varphi) = \emptyset$;

(e) φ *is $\mathbb{B}$-unsatisfiable if and only if* $H_{\boldsymbol{\alpha}}(\varphi) = \emptyset$ *and* $P_{\boldsymbol{\alpha}}(\varphi) = \emptyset$;

(f) φ *is $\mathcal{T}$-valid if and only if* $H_{\boldsymbol{\alpha}}(\neg\varphi) = \emptyset$;

(g) φ *is $\mathbb{B}$-valid if and only if* $H_{\boldsymbol{\alpha}}(\neg\varphi) = \emptyset$ *and* $P_{\boldsymbol{\alpha}}(\neg\varphi) = \emptyset$;

(h) $\varphi_1 \equiv_{\mathcal{T}} \varphi_2$ *if and only if* $H_{\boldsymbol{\alpha}}(\varphi_1) = H_{\boldsymbol{\alpha}}(\varphi_2)$;

(i) $\varphi_1 \equiv_{\mathbb{B}} \varphi_2$ *if and only if* $H_{\boldsymbol{\alpha}}(\varphi_1) = H_{\boldsymbol{\alpha}}(\varphi_2)$ *and* $P_{\boldsymbol{\alpha}}(\varphi_1) = P_{\boldsymbol{\alpha}}(\varphi_2)$;

(j) $\varphi_1 \models_{\mathcal{T}} \varphi_2$ *if and only if* $H_{\boldsymbol{\alpha}}(\varphi_1) \subseteq H_{\boldsymbol{\alpha}}(\varphi_2)$;

(k) $\varphi_1 \models \varphi_2$ *if and only if* $H_{\boldsymbol{\alpha}}(\varphi_1) \subseteq H_{\boldsymbol{\alpha}}(\varphi_2)$ *and* $P_{\boldsymbol{\alpha}}(\varphi_1) \subseteq P_{\boldsymbol{\alpha}}(\varphi_2)$.

Proposition E.2.2. *Consider a $\mathcal{T}$-formula φ and two supersets of Atoms(φ), $\boldsymbol{\alpha}, \boldsymbol{\alpha}'$, such that $\boldsymbol{\alpha} \neq \boldsymbol{\alpha}'$. We have that:*

(a) $H_{\boldsymbol{\alpha}}(\varphi) \neq H_{\boldsymbol{\alpha}'}(\varphi)$ *unless* $H_{\boldsymbol{\alpha}}(\varphi) = H_{\boldsymbol{\alpha}'}(\varphi) = \emptyset$, *and*
$P_{\boldsymbol{\alpha}}(\varphi) \neq P_{\boldsymbol{\alpha}'}(\varphi)$ *unless* $P_{\boldsymbol{\alpha}}(\varphi) = P_{\boldsymbol{\alpha}'}(\varphi) = \emptyset$;

(b) $\bigvee_{\eta \in H_{\boldsymbol{\alpha}}(\varphi)} \eta \not\equiv_{\mathbb{B}} \bigvee_{\eta' \in H_{\boldsymbol{\alpha}'}(\varphi)} \eta'$ *unless* $H_{\boldsymbol{\alpha}}(\varphi) = H_{\boldsymbol{\alpha}'}(\varphi) = \emptyset$, *and*
$\bigvee_{\rho \in P_{\boldsymbol{\alpha}}(\varphi)} \rho \not\equiv_{\mathbb{B}} \bigvee_{\rho' \in P_{\boldsymbol{\alpha}'}(\varphi)} \rho'$ *unless* $P_{\boldsymbol{\alpha}}(\varphi) = P_{\boldsymbol{\alpha}'}(\varphi) = \emptyset$;

(c) $\bigvee_{\eta \in H_{\boldsymbol{\alpha}}(\varphi)} \eta \equiv_{\mathcal{T}} \bigvee_{\eta' \in H_{\boldsymbol{\alpha}'}(\varphi)} \eta'$,
$\bigvee_{\rho \in P_{\boldsymbol{\alpha}}(\varphi)} \rho \equiv_{\mathcal{T}} \bigvee_{\rho' \in P_{\boldsymbol{\alpha}'}(\varphi)} \rho' \equiv_{\mathcal{T}} \bot$.

(d) $H_{\boldsymbol{\alpha}}(\top) \neq H_{\boldsymbol{\alpha}'}(\top)$, $P_{\boldsymbol{\alpha}}(\top) \neq P_{\boldsymbol{\alpha}'}(\top)$.

Lemma E.2.1 ($\mathcal{T}$-reduction of $\varphi|_\ell \wedge \ell$). *Let $\varphi[\boldsymbol{\alpha}]$ be a $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ $\mathcal{T}$-formula, and let ℓ be a literal on $\boldsymbol{\alpha}$. Then $\varphi|_\ell \wedge \ell$ is $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$.*

Proof. $\varphi|_\ell \wedge \ell \equiv_{\mathbb{B}} \varphi \wedge \ell$. Then $P_{\boldsymbol{\alpha}}(\varphi|_\ell \wedge \ell) = P_{\boldsymbol{\alpha}}(\varphi \wedge \ell) \subseteq P_{\boldsymbol{\alpha}}(\varphi) = \emptyset$. Thus, $\varphi|_\ell \wedge \ell$ is $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$. □

Lemma E.2.2 ($\mathcal{T}$-extension of $\neg\ell \vee \varphi|_\ell$ and $\neg\ell \vee (\varphi|_\ell \wedge \ell)$). *Let $\varphi[\boldsymbol{\alpha}]$ be a $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$ $\mathcal{T}$-formula, and let ℓ be a literal on $\boldsymbol{\alpha}$. Then $\neg\ell \vee \varphi|_\ell$ and $\neg\ell \vee (\varphi|_\ell \wedge \ell)$ are $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$.*

Proof. Let φ be $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$. Then $\neg\varphi$ is $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$. By Lemma E.2.1, $(\neg\varphi)|_\ell \wedge \ell$ is $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$. Thus, $\neg(\neg(\varphi|_\ell) \wedge \ell)$ is $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$. Since $\neg(\neg(\varphi|_\ell) \wedge \ell) \equiv_{\mathbb{B}} \varphi|_\ell \vee \neg\ell \equiv_{\mathbb{B}} (\varphi|_\ell \wedge \ell) \vee \neg\ell$, both $\neg\ell \vee \varphi|_\ell$ and $\neg\ell \vee (\varphi|_\ell \wedge \ell)$ are $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$. □

Lemma E.2.3 ($\mathcal{T}$-literal implication check with a $\mathcal{T}$-extended$_\alpha$ $\mathcal{T}$-formula). *Let $\varphi[\boldsymbol{\alpha}]$ be a $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$ $\mathcal{T}$-formula. Let ℓ be a literal in $\boldsymbol{\alpha}$. Then $\ell \models_{\mathcal{T}} \varphi$ if and only if $\ell \models \varphi$.*

Proof.
$\ell \models_{\mathcal{T}} \varphi$ if and only if $\neg\ell \vee \varphi$ is $\mathcal{T}$-valid, that is, if and only if $\neg\ell \vee \varphi|_\ell$ is $\mathcal{T}$-valid.
$\ell \models \varphi$ if and only if $\neg\ell \vee \varphi$ is $\mathbb{B}$-valid, that is, if and only if $\neg\ell \vee \varphi|_\ell$ is $\mathbb{B}$-valid.
By applying Lemma E.2.2, we have that $\neg\ell \vee \varphi|_\ell$ is also $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$.
Thus, by Lemma E.2.12, $\neg\ell \vee \varphi|_\ell$ is $\mathcal{T}$-valid if and only if it is $\mathbb{B}$-valid.
Combining the three facts above, $\ell \models_{\mathcal{T}} \varphi$ if and only if $\ell \models \varphi$. □

E.2.2 Proof of Theorem 8.2.1

The item CD follows straightforwardly from Definition 8.2.1. The items ∧BC, ∨BC, and ¬C follow from Definitions 8.2.1 and 8.2.2 and Proposition E.2.1 Item (c).

We prove SFO in the following lemma.

Lemma E.2.4 (SFO: forgetting on $\mathcal{T}$-reduced$_\alpha$ formulas). *Let $\varphi[\boldsymbol{\alpha}]$ be a $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ $\mathcal{T}$-formula, and let α be a $\mathcal{T}$-atom in $\boldsymbol{\alpha}$. Then $\exists\alpha.\varphi$ is $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}\setminus\{\alpha\}}$.*

Proof. By definition of existential quantification, $\exists\alpha.\varphi \equiv_{\mathbb{B}} \varphi|_\alpha \vee \varphi|_{\neg\alpha}$.
By item CD in the theorem, $\varphi|_\alpha$ and $\varphi|_{\neg\alpha}$ are both $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}\setminus\{\alpha\}}$.
Then, by item ∨BC, $\varphi|_\alpha \vee \varphi|_{\neg\alpha}$ is $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}\setminus\{\alpha\}}$. □

E.2.3 Proof of Theorem 8.2.2

The items CO, CE, EQ, SE, CT, and ME of Theorem 8.2.2 are proven by Lemmas E.2.5, E.2.6, E.2.7, E.2.8, E.2.9 and E.2.10 respectively.

Lemma E.2.5 (CO: $\mathcal{T}$-satisfiability of $\mathcal{T}$-reduced$_\alpha$ formulas). *Let $\varphi[\boldsymbol{\alpha}]$ be a $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ $\mathcal{T}$-formula. Then φ is $\mathcal{T}$-satisfiable if and only if it is $\mathbb{B}$-satisfiable.*

Proof. By Proposition E.2.1 Item (d), φ is $\mathcal{T}$-satisfiable if and only if $H_{\boldsymbol{\alpha}}(\varphi) \neq \emptyset$.
By Proposition E.2.1 Item (e), φ is $\mathbb{B}$-satisfiable if and only if $H_{\boldsymbol{\alpha}}(\varphi) \cup P_{\boldsymbol{\alpha}}(\varphi) \neq \emptyset$.
$P_{\boldsymbol{\alpha}}(\varphi) = \emptyset$ because φ is $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$, thus φ is $\mathcal{T}$-satisfiable if and only if it is $\mathbb{B}$-satisfiable. □

Lemma E.2.6 (CE: $\mathcal{T}$-entailment with a $\mathcal{T}$-reduced$_\alpha$ $\mathcal{T}$-formula). *Let $\varphi[\boldsymbol{\alpha}]$ be a $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ $\mathcal{T}$-formula. Let C be a clause on $\boldsymbol{\alpha}$. Then $\varphi \models_{\mathcal{T}} C$ iff $\varphi \models C$.*

Proof. Let μ be the cube such that $\mu \equiv_{\mathbb{B}} \neg C$. Then:
$\varphi \models_{\mathcal{T}} C$ if and only if $\varphi \wedge \mu$ is $\mathcal{T}$-unsatisfiable, that is, if and only if $\varphi|_\mu \wedge \mu$ is $\mathcal{T}$-unsatisfiable.
$\varphi \models C$ if and only if $\varphi \wedge \mu$ is $\mathbb{B}$-unsatisfiable, that is, if and only if $\varphi|_\mu \wedge \mu$ is $\mathbb{B}$-unsatisfiable.
By iteratively applying Lemma E.2.1 to all literals in μ, we have that $\varphi|_\mu \wedge \mu$ is also $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$. Thus, by Lemma E.2.5, $\varphi|_\mu \wedge \mu$ is $\mathcal{T}$-satisfiable if and only if it is $\mathbb{B}$-satisfiable.
Combining the three facts above, $\varphi \models_{\mathcal{T}} C$ if and only if $\varphi \models C$. □

Lemma E.2.7 (EQ: $\mathcal{T}$-equivalence of $\mathcal{T}$-reduced$_\alpha$ formulas). *Let $\varphi_1[\boldsymbol{\alpha}]$ and $\varphi_2[\boldsymbol{\alpha}]$ be $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ $\mathcal{T}$-formulas. Then $\varphi_1 \equiv_{\mathcal{T}} \varphi_2$ if and only if $\varphi_1 \equiv_{\mathbb{B}} \varphi_2$.*

Proof. By Proposition E.2.1 Item (h), $\varphi_1 \equiv_{\mathcal{T}} \varphi_2$ if and only if $H_{\boldsymbol{\alpha}}(\varphi_1) = H_{\boldsymbol{\alpha}}(\varphi_2)$.
By Proposition E.2.1 Item (i), $\varphi_1 \equiv_{\mathbb{B}} \varphi_2$ if and only if $H_{\boldsymbol{\alpha}}(\varphi_1) = H_{\boldsymbol{\alpha}}(\varphi_2)$ and $P_{\boldsymbol{\alpha}}(\varphi_1) =$

$P_{\boldsymbol{\alpha}}(\varphi_2)$.
$P_{\boldsymbol{\alpha}}(\varphi_1) = P_{\boldsymbol{\alpha}}(\varphi_2) = \emptyset$ because φ_1 and φ_2 are $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$.
Thus, $\varphi_1 \equiv_{\mathcal{T}} \varphi_2$ if and only if $\varphi_1 \equiv_{\mathbb{B}} \varphi_2$. □

Lemma E.2.8 (SE: $\mathcal{T}$-entailment of $\mathcal{T}$-reduced$_{\alpha}$ formulas). *Let $\varphi_1[\boldsymbol{\alpha}]$ and $\varphi_2[\boldsymbol{\alpha}]$ be $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ $\mathcal{T}$-formulas. Then $\varphi_1 \models_{\mathcal{T}} \varphi_2$ if and only if $\varphi_1 \models \varphi_2$.*

Proof. By Proposition E.2.1 Item (j), $\varphi_1 \models_{\mathcal{T}} \varphi_2$ if and only if $H_{\boldsymbol{\alpha}}(\varphi_1) \subseteq H_{\boldsymbol{\alpha}}(\varphi_2)$.
By Proposition E.2.1 Item (k), $\varphi_1 \models \varphi_2$ if and only if $H_{\boldsymbol{\alpha}}(\varphi_1) \subseteq H_{\boldsymbol{\alpha}}(\varphi_2)$ and $P_{\boldsymbol{\alpha}}(\varphi_1) \subseteq P_{\boldsymbol{\alpha}}(\varphi_2)$.
$P_{\boldsymbol{\alpha}}(\varphi_1) = P_{\boldsymbol{\alpha}}(\varphi_2) = \emptyset$ because φ_1 and φ_2 are $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$.
Thus, $\varphi_1 \models_{\mathcal{T}} \varphi_2$ if and only if $\varphi_1 \models \varphi_2$. □

Lemma E.2.9 (CT: $\mathcal{T}$-#SAT of $\mathcal{T}$-reduced$_{\alpha}$ $\mathcal{T}$-formula). *Let $\varphi[\boldsymbol{\alpha}]$ be a $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ $\mathcal{T}$-formula. Then $\mathcal{T}$-$\#\mathsf{SAT}(\varphi) = \#\mathsf{SAT}(\varphi^p)$.*

Proof. By Definition 7.1.1, for an arbitrary $\mathcal{T}$-formula φ, $\#\mathsf{SAT}(\varphi^p[\boldsymbol{\alpha}]) = |H_{\boldsymbol{\alpha}}(\varphi)| + |P_{\boldsymbol{\alpha}}(\varphi)|$. If φ is $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$, then $|P_{\boldsymbol{\alpha}}(\varphi)| = 0$. Thus $\mathcal{T}$-$\#\mathsf{SAT}(\varphi) = |H_{\boldsymbol{\alpha}}(\varphi)| = \#\mathsf{SAT}(\varphi^p)$. □

Lemma E.2.10 (ME: $\mathcal{T}$-consistent truth assignment enumeration for $\mathcal{T}$-reduced$_{\alpha}$ $\mathcal{T}$-formulas). *Let $\varphi[\boldsymbol{\alpha}]$ be a $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ $\mathcal{T}$-formula. Then $\mathcal{T}$-$\mathsf{AllSAT}(\varphi)$ is the refinement of $\mathsf{AllSAT}(\varphi^p)$.*

Proof. For an arbitrary formula φ, the refinement of $\mathsf{AllSAT}(\varphi^p)$ is $H_{\boldsymbol{\alpha}}(\varphi) \cup P_{\boldsymbol{\alpha}}(\varphi)$. Since φ is $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ so that $P_{\boldsymbol{\alpha}}(\varphi) = \emptyset$ by Definition 8.2.1, it reduces to $H_{\boldsymbol{\alpha}}(\varphi) = \mathcal{T}$-$\mathsf{AllSAT}(\varphi)$. □

E.2.4 Proof of Theorem 8.2.3

The item CD follows straightforwardly from Definition 8.2.2. The items ∧BC, ∨BC, and ¬C follow from Definitions 8.2.1 and 8.2.2 and Proposition E.2.1 Item (c).

We prove SFO in the following lemma.

Lemma E.2.11 (SFO: forgetting on $\mathcal{T}$-extended$_\alpha$ formulas). *Let $\varphi[\boldsymbol{\alpha}]$ be a $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$ $\mathcal{T}$-formula, and let α be a $\mathcal{T}$-atom in $\boldsymbol{\alpha}$. Then $\exists\alpha.\varphi$ is $\mathcal{T}$-extended$_{\boldsymbol{\alpha}\setminus\{\alpha\}}$.*

Proof. By definition of existential quantification, $\exists\alpha.\varphi \equiv_{\mathbb{B}} \varphi|_\alpha \vee \varphi|_{\neg\alpha}$.
By item CD in the theorem, $\varphi|_\alpha$ and $\varphi|_{\neg\alpha}$ are both $\mathcal{T}$-extended$_{\boldsymbol{\alpha}\setminus\{\alpha\}}$.
Then, by item ∨BC, $\varphi|_\alpha \vee \varphi|_{\neg\alpha}$ is $\mathcal{T}$-extended$_{\boldsymbol{\alpha}\setminus\{\alpha\}}$. □

E.2.5 Proof of Theorem 8.2.4

The items VA, IM, EQ, and SE of Theorem 8.2.4 are proven by Lemmas E.2.12, E.2.13, E.2.14 and E.2.15 respectively.

Lemma E.2.12 (VA: $\mathcal{T}$-validity of $\mathcal{T}$-extended$_\alpha$ $\mathcal{T}$-formulas). *A $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$ $\mathcal{T}$-formula $\varphi[\boldsymbol{\alpha}]$ is $\mathcal{T}$-valid if and only if it is $\mathbb{B}$-valid.*

Proof. (If case) A $\mathbb{B}$-valid $\mathcal{T}$-formula φ is obviously also $\mathcal{T}$-valid by definition.
(Only if case) $H_{\boldsymbol{\alpha}}(\neg\varphi) = \emptyset$ because φ is $\mathcal{T}$-valid (Proposition E.2.1, Item (f)). $P_{\boldsymbol{\alpha}}(\neg\varphi) = \emptyset$ because φ is $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$ (Definition 8.2.2). Thus $\neg\varphi$ is $\mathbb{B}$-unsatisfiable, so that φ is $\mathbb{B}$-valid. □

Lemma E.2.13 (IM: $\mathcal{T}$-implicant check with a $\mathcal{T}$-extended$_\alpha$ $\mathcal{T}$-formula). *Let $\varphi[\boldsymbol{\alpha}]$ be a $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$ $\mathcal{T}$-formula. Let μ be a cube on $\boldsymbol{\alpha}$. Then $\mu \models_{\mathcal{T}} \varphi$ if and only if $\mu \models \varphi$.*

Proof. Let $\mu \stackrel{\text{def}}{=} \bigwedge_{i=1}^{n} \ell_i$ for some $n \geq 1$. We show that $\bigwedge_{i=1}^{n} \ell_i \models_{\mathcal{T}} \varphi$ if and only if $\bigwedge_{i=1}^{n} \ell_i \models_{\mathbb{B}} \varphi$ by induction on n:
(base): By Lemma E.2.3, if $n = 1$, then $\ell_1 \models_{\mathcal{T}} \varphi$ if and only if $\ell_1 \models \varphi$.
(step): We recall that $\neg\ell \vee \varphi \equiv_{\mathbb{B}} \neg\ell \vee (\ell \wedge \varphi) \equiv_{\mathbb{B}} \neg\ell \vee (\ell \wedge \varphi|_\ell) \equiv_{\mathbb{B}} \neg\ell \vee \varphi|_\ell$ (Lemma A.7 in [62]).
$\bigwedge_{i=1}^{n} \ell_i \models_{\mathcal{T}} \varphi$ if and only if $\bigwedge_{i=1}^{n-1} \ell_i \models_{\mathcal{T}} (\neg\ell_n \vee \varphi)$, if and only if $\bigwedge_{i=1}^{n-1} \ell_i \models_{\mathcal{T}} (\neg\ell_n \vee \varphi|_{\ell_n})$.
$\bigwedge_{i=1}^{n} \ell_i \models \varphi$ if and only if $\bigwedge_{i=1}^{n-1} \ell_i \models (\neg\ell_n \vee \varphi)$, if and only if $\bigwedge_{i=1}^{n-1} \ell_i \models (\neg\ell_n \vee \varphi|_{\ell_n})$.
By Lemma E.2.2, $(\neg\ell_n \vee \varphi|_{\ell_n})$ is $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$.
Thus, by induction, $\bigwedge_{i=1}^{n-1} \ell_i \models_{\mathcal{T}} (\neg\ell_n \vee \varphi|_{\ell_n})$ if and only if $\bigwedge_{i=1}^{n-1} \ell_i \models (\neg\ell_n \vee \varphi|_{\ell_n})$, that is, $\bigwedge_{i=1}^{n} \ell_i \models_{\mathcal{T}} \varphi$ if and only if $\bigwedge_{i=1}^{n} \ell_i \models \varphi$. □

Lemma E.2.14 (EQ: $\mathcal{T}$-equivalence of $\mathcal{T}$-extended$_\alpha$ formulas). *Let $\varphi_1[\boldsymbol{\alpha}]$ and $\varphi_2[\boldsymbol{\alpha}]$ be $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$ $\mathcal{T}$-formulas. Then $\varphi_1 \equiv_{\mathcal{T}} \varphi_2$ if and only if $\varphi_1 \equiv_{\mathbb{B}} \varphi_2$.*

Proof. By Definition 8.2.2, $\neg\varphi_1$ and $\neg\varphi_2$ are $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$.
Thus, by Lemma E.2.7, $\neg\varphi_1 \equiv_{\mathcal{T}} \neg\varphi_2$ if and only if $\neg\varphi_1 \equiv_{\mathbb{B}} \neg\varphi_2$.
Therefore, $\varphi_1 \equiv_{\mathcal{T}} \varphi_2$ if and only if $\varphi_1 \equiv_{\mathbb{B}} \varphi_2$. □

Lemma E.2.15 (SE: $\mathcal{T}$-entailment of $\mathcal{T}$-extended$_\alpha$ formulas). *Let $\varphi_1[\boldsymbol{\alpha}]$ and $\varphi_2[\boldsymbol{\alpha}]$ be $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$ $\mathcal{T}$-formulas. Then $\varphi_1 \models_{\mathcal{T}} \varphi_2$ if and only if $\varphi_1 \models_{\mathbb{B}} \varphi_2$.*

Proof. By Definition 8.2.2, $\neg\varphi_1$ and $\neg\varphi_2$ are $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$.
Thus, by Lemma E.2.8, $\neg\varphi_2 \models_{\mathcal{T}} \neg\varphi_1$ if and only if $\neg\varphi_2 \models_{\mathbb{B}} \neg\varphi_1$.
Therefore, $\varphi_1 \models_{\mathcal{T}} \varphi_2$ if and only if $\varphi_1 \models_{\mathbb{B}} \varphi_2$. □

Comparing Lemmas E.2.14 and E.2.15 with Lemmas E.2.7 and E.2.8 respectively, we note that the same property holds for both $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$ and $\mathcal{T}$-extended$_{\boldsymbol{\alpha}}$ pairs of $\mathcal{T}$-formulas. This is unsurprising because $\varphi_1 \equiv \varphi_2$ if and only if $\neg\varphi_1 \equiv \neg\varphi_2$ and that $\varphi_1 \models \varphi_2$ if and only if $\neg\varphi_2 \models \neg\varphi_1$.

E.2.6 Proof of Theorem 8.2.5

Proof. By construction, all C_l's in $Cl_{\boldsymbol{\alpha}}(\varphi)$ are $\mathcal{T}$-valid, so that $\bigwedge_{C_l \in Cl_{\boldsymbol{\alpha}}(\varphi)} C_l$ is $\mathcal{T}$-valid. Thus:

(i) By (8.2), $\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi) \equiv_{\mathcal{T}} \varphi$ because all C_l's are $\mathcal{T}$-valid.

(ii) By (8.2), $\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi) \models_{\mathbb{B}} \varphi$.

(iii) By Item (i), $H_{\boldsymbol{\alpha}}(\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi)) = H_{\boldsymbol{\alpha}}(\varphi)$; by Item (ii), $P_{\boldsymbol{\alpha}}(\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi)) \subseteq P_{\boldsymbol{\alpha}}(\varphi)$. Consider any $\rho_j \in P_{\boldsymbol{\alpha}}(\varphi)$. By Definition 7.1.2, $\rho_j \models_{\mathbb{B}} \neg C_l$ for some $C_l \in Cl_{\boldsymbol{\alpha}}(\varphi)$. Since ρ_j is a total assignment on $\boldsymbol{\alpha}$, $\rho_j \not\models_{\mathbb{B}} C_l$, that is, $\rho_j \not\models_{\mathbb{B}} \mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi)$, that is, $\rho_j \notin P_{\boldsymbol{\alpha}}(\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi))$. Thus $P_{\boldsymbol{\alpha}}(\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi)) = \emptyset$, that is, $\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi)$ is $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$.

□

E.2.7 Proof of Theorem 8.2.6

Proof. Comparing (8.2) and (8.3), we see that $\mathcal{T}\mathsf{ext}_{\boldsymbol{\alpha}}(\varphi) \equiv_{\mathbb{B}} \neg\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\neg\varphi)$. Thus, Items (i), (ii) and (iii) follow directly from Theorem 8.2.5. □

E.2.8 Proof of Theorem 8.2.7

We first prove the following lemma.

Lemma E.2.16. *Let $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ be disjoint sets of atoms and let $\mathbf{A}$ and $\mathbf{B}$ denote their Boolean abstraction. Let $\varphi[\boldsymbol{\alpha}]$ be a $\mathcal{T}$-formula. Let $Cl_{\boldsymbol{\alpha},\boldsymbol{\beta}}(\varphi) \stackrel{def}{=} \{C_1, \ldots, C_K\}$ be a set of $\mathcal{T}$-lemmas on $\boldsymbol{\alpha} \cup \boldsymbol{\beta}$ that rules out $P_{\boldsymbol{\alpha}}(\varphi)$. Then we have that:*

$$\varphi^p \wedge \exists \mathbf{B}. \bigwedge_{C_l \in Cl_{\boldsymbol{\alpha},\boldsymbol{\beta}}(\varphi)} C_l^p \equiv \bigvee_{\eta_i \in H_{\boldsymbol{\alpha}}(\varphi)} \eta_i^p. \tag{E.1}$$

Proof. Since $Cl_{\boldsymbol{\alpha},\boldsymbol{\beta}}(\varphi)$ rules out $P_{\boldsymbol{\alpha}}(\varphi)$, we have

$$\bigvee_{\rho_j \in P_{\boldsymbol{\alpha}}(\varphi)} \rho_j \wedge \bigwedge_{C_l \in Cl_{\boldsymbol{\alpha},\boldsymbol{\beta}}(\varphi)} C_l \equiv_{\mathbb{B}} \bot, \tag{E.2}$$

$$\text{that is:} \quad \bigvee_{\rho_j \in P_{\boldsymbol{\alpha}}(\varphi)} \rho_j^p \wedge \bigwedge_{C_l \in Cl_{\boldsymbol{\alpha},\boldsymbol{\beta}}(\varphi)} C_l^p \equiv \bot, \tag{E.3}$$

$$\text{equivalently:} \quad \bigvee_{\rho_j \in P_{\boldsymbol{\alpha}}(\varphi)} \rho_j^p \wedge \exists \mathbf{B}. \bigwedge_{C_l \in Cl_{\boldsymbol{\alpha},\boldsymbol{\beta}}(\varphi)} C_l^p \equiv \bot, \tag{E.4}$$

Let $\varphi^{*p} \stackrel{\text{def}}{=} \exists \mathbf{B}. \bigwedge_{C_l \in Cl_{\boldsymbol{\alpha},\boldsymbol{\beta}}(\varphi)} C_l^p$. Since the η_i's in $H_{\boldsymbol{\alpha}}(\varphi)$ are all total on $\boldsymbol{\alpha}$, then for each η_i, either $\eta_i^p \models \varphi^{*p}$ or $\eta_i^p \models \neg\varphi^{*p}$. The latter is not possible, because it would mean that $\eta_i^p \wedge \varphi^{*p} \models \bot$, and hence, by (E.4), $\eta_i \in P_{\boldsymbol{\alpha}}(\varphi)$, which would contradict the fact that $H_{\boldsymbol{\alpha}}(\varphi)$ and $P_{\boldsymbol{\alpha}}(\varphi)$ are disjoint. Thus, we have

$$\bigvee_{\eta_i \in H_{\boldsymbol{\alpha}}(\varphi)} \eta_i^p \models \exists \mathbf{B}. \bigwedge_{C_l \in Cl_{\boldsymbol{\alpha},\boldsymbol{\beta}}(\varphi)} C_l^p. \tag{E.5}$$

Hence, we have

$$\begin{aligned}
& \varphi^p \wedge \exists \mathbf{B}. \bigwedge_{C_l \in Cl_{\boldsymbol{\alpha},\boldsymbol{\beta}}(\varphi)} C_l^p \\
\textit{by (7.1):} \quad & \equiv \left(\bigvee_{\eta_i \in H_{\boldsymbol{\alpha}}(\varphi)} \eta_i^p \vee \bigvee_{\rho_j \in P_{\boldsymbol{\alpha}}(\varphi)} \rho_j^p \right) \wedge \exists \mathbf{B}. \bigwedge_{C_l \in Cl_{\boldsymbol{\alpha},\boldsymbol{\beta}}(\varphi)} C_l^p \\
\wedge/\vee: \quad & \equiv \left(\bigvee_{\eta_i \in H_{\boldsymbol{\alpha}}(\varphi)} \eta_i^p \wedge \exists \mathbf{B}. \bigwedge_{C_l \in Cl_{\boldsymbol{\alpha},\boldsymbol{\beta}}(\varphi)} C_l^p \right) \vee \left(\bigvee_{\rho_j \in P_{\boldsymbol{\alpha}}(\varphi)} \rho_j^p \wedge \exists \mathbf{B}. \bigwedge_{C_l \in Cl_{\boldsymbol{\alpha},\boldsymbol{\beta}}(\varphi)} C_l^p \right) \\
\textit{by (E.4):} \quad & \equiv \left(\bigvee_{\eta_i \in H_{\boldsymbol{\alpha}}(\varphi)} \eta_i^p \wedge \exists \mathbf{B}. \bigwedge_{C_l \in Cl_{\boldsymbol{\alpha},\boldsymbol{\beta}}(\varphi)} C_l^p \right) \\
\textit{by (E.5):} \quad & \equiv \bigvee_{\eta_i \in H_{\boldsymbol{\alpha}}(\varphi)} \eta_i^p.
\end{aligned}$$

□

This leads to the proof of Theorem 8.2.7.

Proof. By construction, all C_l's in $Cl_{\boldsymbol{\alpha},\boldsymbol{\beta}}(\varphi)$ are $\mathcal{T}$-valid, so that $\bigwedge_{C_l \in Cl_{\boldsymbol{\alpha},\boldsymbol{\beta}}(\varphi)} C_l$ is $\mathcal{T}$-valid, so that $\exists \boldsymbol{\beta}. \bigwedge_{C_l \in Cl_{\boldsymbol{\alpha},\boldsymbol{\beta}}(\varphi)} C_l$ is $\mathcal{T}$-valid. Thus:

(i) By Lemma E.2.16, $\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi) \equiv_{\mathbb{B}} \bigvee_{\eta_i \in H_{\boldsymbol{\alpha}}(\varphi)} \eta_i \equiv_{\mathcal{T}} \varphi$.

(ii) By (8.4), $\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi) \models_{\mathbb{B}} \varphi$.

(iii) By Lemma E.2.16, $\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi) \equiv_{\mathbb{B}} \bigvee_{\eta_i \in H_{\boldsymbol{\alpha}}(\varphi)} \eta_i$, so that $\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\varphi)$ is $\mathcal{T}$-reduced$_{\boldsymbol{\alpha}}$.

□

E.2.9 Proof of Theorem 8.2.8

Proof. Comparing (8.4) and (8.5), we see that $\mathcal{T}\mathsf{ext}_{\boldsymbol{\alpha}}(\varphi) \equiv_{\mathbb{B}} \neg\mathcal{T}\mathsf{red}_{\boldsymbol{\alpha}}(\neg\varphi)$. Thus, Items (i), (ii) and (iii) follow directly from Theorem 8.2.7. □

E.2.10 Proof of Theorem 8.3.1

Proof.

(a) [CO] By Lemma E.2.5, ψ is $\mathcal{T}$-satisfiable iff it is $\mathbb{B}$-satisfiable, which can be computed in polynomial time w.r.t. the size of ψ for a d-DNNF formula [62].

(b) [CE] By Lemma E.2.6, $\psi \models_{\mathcal{T}} C$ iff $\psi \models C$, which can be computed in polynomial time w.r.t. the size of ψ and C for a d-DNNF formula [62].

(c) [CT] By Lemma E.2.9, $\mathcal{T}\text{-}\#SAT(\psi[\boldsymbol{\alpha}]) = \#\mathsf{SAT}(\psi^p[\boldsymbol{\alpha}])$, which can be computed in polynomial time w.r.t. the size of ψ for a d-DNNF formula [62].

(d) [ME] By Lemma E.2.10, $\mathcal{T}\text{-}AllSAT(\psi[\boldsymbol{\alpha}]) = \mathsf{AllSAT}(\psi^p[\boldsymbol{\alpha}])$, which can be computed in polynomial time w.r.t. the size of ψ and the number of output models for a d-DNNF formula [62].

□

E.2.11 Proof of Theorem 8.3.2

Proof.

(a) [EQ] By Lemma E.2.7, $\psi_1 \equiv_{\mathcal{T}} \psi_2$ if and only if $\psi_1 \equiv_{\mathbb{B}} \psi_2$, which can be computed in polynomial time w.r.t. the size of ψ_1 and ψ_2 for the listed NNF formulas [62, 60].

(b) [SE] By Lemma E.2.8, $\psi_1 \models_{\mathcal{T}} \psi_2$ if and only if $\psi_1 \models_{\mathbb{B}} \psi_2$, which can be computed in polynomial time w.r.t. the size of ψ_1 and ψ_2 for the listed NNF formulas [62, 60].

□

E.2.12 Proof of Theorem 8.3.3

Proof.

(a) [VA] By Lemma E.2.12, ψ is $\mathcal{T}$-valid iff it is $\mathbb{B}$-valid, which can be computed in polynomial time w.r.t. the size of ψ for a d-DNNF formula [62].

(b) [IM] By Lemma E.2.13, $\mu \models_{\mathcal{T}} \psi$ if and only if $\mu \models \psi$, which can be computed in polynomial time w.r.t. the size of ψ and μ for a d-DNNF formula [62].

□

E.2.13 Proof of Theorem 8.3.4

Proof.

(a) [EQ] By Lemma E.2.14, $\psi_1 \equiv_{\mathcal{T}} \psi_2$ if and only if $\psi_1 \equiv_{\mathbb{B}} \psi_2$, which can be computed in polynomial time w.r.t. the size of ψ_1 and ψ_2 for the listed NNF formulas [62, 60].

(b) [SE] By Lemma E.2.15, $\psi_1 \models_{\mathcal{T}} \psi_2$ if and only if $\psi_1 \models_{\mathbb{B}} \psi_2$, which can be computed in polynomial time w.r.t. the size of ψ_1 and ψ_2 for the listed NNF formulas [62, 60].

□