FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Backdoors to Tractability of Disjunctive Answer Set Programming

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften

eingereicht von

Johannes Klaus Fichte
Matrikelnummer 0929426

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Prof. Dr. Stefan Szeider
Co-Betreuung: Prof. Dr. Torsten Schaub

Diese Dissertation haben begutachtet:

Stefan Szeider	Mirosław Truszczyński

Wien, 19. August 2015

Johannes Klaus Fichte

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Backdoors to Tractability of Disjunctive Answer Set Programming

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Naturwissenschaften

by

Johannes Klaus Fichte
Registration Number 0929426

to the Faculty of Informatics

at the Vienna University of Technology

Advisor: Prof. Dr. Stefan Szeider
Co-Advisor: Prof. Dr. Torsten Schaub

The dissertation has been reviewed by:

Stefan Szeider

Mirosław Truszczyński

Vienna, 19th August, 2015

Johannes Klaus Fichte

Erklärung zur Verfassung der Arbeit

Johannes Klaus Fichte
Favoritenstraße, 9/11, 1040, Vienna, Austria

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 19. August 2015

Johannes Klaus Fichte

Acknowledgements

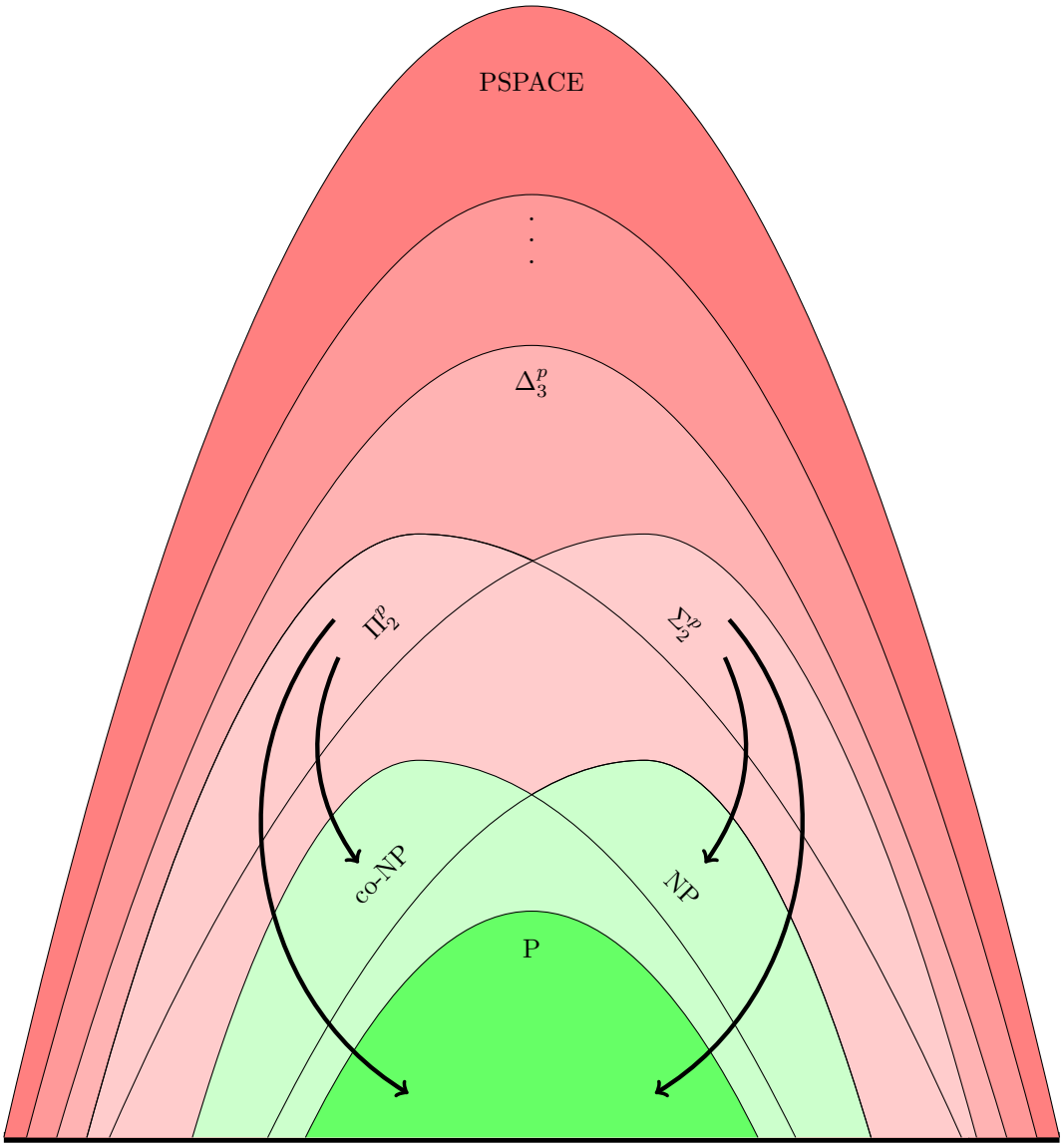
First of all, I would like to thank my principal advisor Stefan Szeider. I am very grateful for his constant advice, for his patient guidance, for the many things he has taught me about parameterized complexity, writing, and academic activities, as well as for many critical comments on my research. I highly appreciate that he granted me considerable freedom and encouraged me during my research.

I would also like to thank my co-advisor Torsten Schaub for providing me with great insights into Asp solving in practice, reasoning, and artificial intelligence. He gave me the chance to work in Potsdam, had always advice when I needed it, and provided highly enthusiastic encouragement. I am very grateful to Martin Gebser for frequent discussions, for providing me with detailed information on Asp modelling and solving, and for critical comments on my work.

Many thanks to all my current and former colleagues and friends in Vienna, in particular, Andreas, Dasha, Eduard, Friedrich, Neha, Patrick, Robert, Ronald, Sebastian, Serge, Simone, Stefan, and Thomas, for interesting discussions and a great time. My special thanks go to my colleagues and friends in Potsdam, in particular, Benjamin, Holger, Javier, Max, Orkunt, Philipp, Roland, and Simon, for providing me with insights into their incredible solvers, modelling problems, for interesting discussions, as well as for great social and sport events.

I am very grateful to my family, my parents Klaus-Dieter and Helga, my brothers Friedrich and Lorenz who supported me with encouragement and inspiration. I am very glad that we sticked together during the hard times over the last years.

Finally, I would like to thank my good friends, who I constantly kept bugging with spelling, typesetting, and all the other questions in daily live. Thanks Anne, Annett, Dirk, Francie, Georg, Jonad, Martin, Michael, Min, Moritz, Ruslan, Silia, and Tina; very special thanks to Ruslan for his friendly and constant hospitality and to Francie who always looks after me and was very caring and supportive while I worked in Berlin.


PSPACE
Δ_3^p
Π_2^p
Σ_2^p
co-NP
NP
P

Abstract

Answer set programming (Asp) is an increasingly popular framework for declarative programming that admits the description of problems by means of atoms, rules, and constraints that form a logic program. The solutions of an answer set program are called answer sets. Many problems in artificial intelligence such as non-monotonic reasoning can be directly formulated in Asp. Reasoning problems for propositional disjunctive programs, which is the focus of this thesis, are of high computational complexity. For instance, the problems of deciding whether a program has at least one answer set or whether a given atom is contained in at least one answer set, are complete for the second level of the Polynomial Hierarchy.

In this thesis we tackle these hard problems using backdoors in problem instances, which are sets of atoms that can be used as clever reasoning shortcuts through the search space. The concept of backdoors has widely been used in theoretical investigations in the areas of propositional satisfiability and constraint satisfaction, we show that backdoors can be fruitfully adapted to Asp.

We develop a rigorous theory of backdoors for Asp and carry out a fine-grained asymptotic computational complexity analysis that takes backdoors into account. We establish new algorithms that can detect and take advantage of small backdoors to solve or to significantly simplify problem instances. More precisely, certain backdoors allow us to solve Asp reasoning problems efficiently for instances with small backdoors (fixed-parameter tractability), other backdoors allow us to significantly simplify the problem instance (complexity barrier breaking reduction), and some backdoors cannot even be used to simplify the problem instance (intractability). Particularly, our simplifications break the complexity barrier between the second level of the Polynomial Hierarchy and the first level by means of reductions that work efficiently for instances with small backdoors. Further, we elaborate upon a detailed comparison where we compare the size of certain types of backdoors with each other. We show that backdoors can serve as a unifying framework for restrictions that have been identified in the literature under which Asp problems significantly simplify and become tractable or NP-complete.

Kurzfassung

Antwortmengen-Programmierung (Asp) ist ein zunehmend populäres deklarative Programmierkonzept, welches die Modellierung von Problemen mit Hilfe logischer Programme erlaubt. Logische Programme bestehen aus Atomen, Regeln und Bedingungen, ihre Lösungen werden Antwortmengen genannt. Viele Probleme der künstlichen Intelligenz, wie beispielsweise nicht-monotones Schließen, können direkt in Asp formuliert werden. In dieser Arbeit beschäftigen wir uns vorrangig mit Schlussfolgerungsproblemen von Asp, beispielsweise die Fragestellung ob ein Programm mindestens eine Antwortmenge besitzt oder ob ein vorgegebenes Atom in mindestens einer Antwortmenge enthalten ist. Zentraler Gegenstand der gesamten Dissertation sind sogenannte aussagenlogische, disjunktive Programme.

Die meisten Asp-Probleme sind schwer zu lösen, in der Komplexitätstheorie sind diese Probleme bekannt als vollständig für die zweite Stufe der polynomiellen Hierarchie. In dieser Arbeit lösen wir diese schweren Probleme mit Hilfe von sogenannten Hintertüren (Backdoors), die sich in Probleminstanzen finden lassen. Dabei kann eine günstige Hintertür für das Schlussfolgern als eine geschickte Abkürzung durch den Lösungsraum angesehen werden, die es ermöglicht das Problem in der Praxis schnell zu lösen. Die Größe der kleinsten Hintertür liefert uns darüber hinaus ein theoretisches Maß für die Schwierigkeit oder Einfachheit einer Probleminstanz. Das Grundprinzip einer Hintertür wurde bereits vielfältig in theoretischen Untersuchungen für das aussagenlogischen Erfüllbarkeitsproblem und das Bedingungserfüllungsproblem verwendet.

In dieser Arbeit zeigen wir, dass das Prinzip einer Hintertür erfolgreich auf Asp übertragen werden kann. Wir entwickeln eine umfassende Theorie für Asp-Hintertüren. Wir führen eine detaillierte asymptotische Komplexitätsanalyse durch, die Asp-Hintertüren mit einbezieht. Wir stellen neue Algorithmen vor, mit denen man kleine Hintertüren in Probleminstanzen erkennen und ausnutzen kann. Unsere Algorithmen erlauben es Probleminstanzen, die eine kleiner Hintertür besitzen, schnell zu lösen oder signifikant zu vereinfachen. Genauer, gewisse Hintertüren erlauben es uns Schlussfolgerungsprobleme von Asp effizient für Probleminstanzen, die eine kleine Hintertür besitzen, zu lösen (Parametrisierbarkeit); andere Hintertüren erlauben es uns dagegen Probleminstanzen, die eine kleine Hintertür besitzen, signifikant zu vereinfachen (komplexitätsbarrierebrechende Reduktionen); wiederum andere Hintertüren können aus theoretischer Sicht nicht genutzt werden, um Probleminstanzen effizient zu lösen oder zu vereinfachen (Schwie-

rigkeit). Unsere komplexitätsbarrierebrechende Reduktionen liefern insbesondere für Probleminstanzen, die eine kleiner Hintertür besitzen, einen Ansatz die Grenze zwischen der zweiten Stufe der polynomiellen Hierarchy und der ersten Stufe zu durchbrechen und die betrachteten Probleme mit etablierten Methoden effektiver zu lösen. Darüber hinaus erarbeiten wir einen detaillierten Vergleich, in dem wir die Größe verschiedener Arten von Hintertüren miteinander in Beziehung setzen. Wir zeigen, dass das Prinzip einer Hintertür als vereinheitlichendes System für aus der Literatur bekannte Restriktionen, unter denen ASP-Probleme signifikant vereinfacht werden können, dienen kann.

Contents

1 Preface **1**
- 1.1 Motivation . . . 1
- 1.2 Methodology and Research Questions . . . 2
- 1.3 Contribution . . . 3
- 1.4 Publications . . . 5

2 Preliminaries **7**
- 2.1 Answer Set Programming . . . 7
- 2.2 Propositional Logic and Quantified Boolean Formulas . . . 12
- 2.3 Classical Computational Complexity . . . 15
- 2.4 Parameterized Complexity . . . 17
- 2.5 Graphs . . . 21

3 Backdoors **23**
- 3.1 Satisfiability Backdoors . . . 24
- 3.2 Backdoors of Answer Set Programs . . . 25
- 3.3 Backdoor Evaluation . . . 26
- 3.4 Backdoor Detection . . . 32

4 Tractability Backdoors **37**
- 4.1 Target Class Horn . . . 38
- 4.2 Target Classes Based on Acyclicity . . . 40

5 Lifting Parameters **53**
- 5.1 ASP Parameters . . . 53
- 5.2 Lifting . . . 54
- 5.3 Lifting Theorem . . . 55

6 Polynomial-Time Preprocessing **59**
- 6.1 Kernelization . . . 60
- 6.2 Backdoor Detection . . . 61
- 6.3 Backdoor Evaluation . . . 62
- 6.4 Background and Related Work . . . 63
- 6.5 Contribution and Discussion . . . 64

7 Backdoor Trees **67**
7.1 Backdoor Trees of Answer Set Programs . . . 68
7.2 Backdoor Tree Evaluation . . . 72
7.3 Relation to Backdoors . . . 75
7.4 Backdoor Tree Detection . . . 77

8 Theoretical Comparison of Asp Parameters **83**
8.1 Parameter Comparison . . . 84
8.2 Asp Parameters Based on Backdoor Size . . . 85
8.3 Asp Parameters Based on the Distance from Horn . . . 87
8.4 Asp Parameters Based on the Distance from Being Stratified . . . 90
8.5 Incidence Treewidth . . . 96
8.6 Dependency Treewidth . . . 98
8.7 Interaction Treewidth . . . 100
8.8 Number of Bad Even Cycles . . . 102
8.9 Number of Positive Cycles (Loop Formulas) . . . 104
8.10 Head-Cycles . . . 105

9 Complexity Barrier Breaking Reductions **107**
9.1 Backdoors to Normality . . . 108
9.2 Backdoors to Tightness . . . 120

10 Practical Considerations **125**
10.1 Backdoor Detection . . . 125
10.2 Backdoor Evaluation . . . 129

11 Summary **133**
11.1 Technical Results . . . 133
11.2 Overall Results . . . 134
11.3 Future Research . . . 134

List of Figures **137**

List of Tables **138**

List of Algorithms **139**

Index **141**

Bibliography **149**

Curriculum Vitae **181**

CHAPTER 1

Preface

1.1 Motivation

Answer set programming (ASP) is an increasingly popular framework for declarative programming [MT99; Nie99; Geb+12]. ASP admits the description of problems by means of rules and constraints that form a logic program. Solutions to the program are so-called stable models or answer sets. Many important problems in artificial intelligence and reasoning can be represented in terms of ASP, i.e., the search for answer sets of logic programs. Throughout the thesis we consider so-called disjunctive programs. Various ASP solvers (see e.g., [Geb+11c; Leo+06]) have been designed and used to solve large industrial applications, e.g., social networks [JSS12], match making [Geb+13b], planning at a seaport [Ric+12], optimization of packaging of Linux distributions [Geb+11b]. Moreover, recent ASP competitions [Den+09; CIR14; Alv+13] have demonstrated that modern solvers work efficiently on a wide variety of instances.

However, computational problems for disjunctive propositional ASP (such as deciding whether a program has a solution, or whether a certain atom is contained in at least one or in all solutions) are complete for the second level of the Polynomial Hierarchy [EG95]; thus, propositional ASP problems are "harder than NP" and have a higher worst-case complexity than CSP and SAT. In the literature, several restrictions have been identified that make ASP tractable, most prominently the **Horn** fragment and the stratified fragment [GL88; ABW88; BD94].

Unfortunately, there is little theoretical knowledge why modern solvers work efficiently on many real-world instances (see e.g., [Var14] for SAT), but it is widely believed that they exploit the presence of a "hidden structure" (see e.g., [Bie+09]) which is usually accomplished by means of various solving techniques including decision heuristics. A variety of research results have been established to improve on the theoretical understanding of the effectiveness of modern SAT solvers, e.g., [WGS03a; WGS03b; Ans+08; Gom+08;

Bie+09; PD11; AFT11; GS12b]. However, in the field of Asp only a few theoretical results, in particular in computational complexity [JPW09; GPW10; Pic+14], have been carried out to diminish the huge gap between efficiency in practical solving and worst case bounds of classical complexity theory.

1.2 Methodology and Research Questions

In this thesis we exploit hidden structure in terms of backdoors in problem instances, to establish a more fine granted analysis of the main reasoning problems of disjunctive propositional Asp in dependency of the existing structure. The notion of a backdoor *makes* the *vague notion of a hidden structure* theoretically more *precise*. The idea is the following: Quite often we can solve a computationally hard problem efficiently when we restrict the input to a certain *subclass* of the originally allowed input. If we can identify a *backdoor into* such a subclass (*target class*) and exploit the backdoor, we can solve the computational problem faster. We usually take a small part of the input as a backdoor, in the context of Asp, a *small set of atoms*, which then represents a "clever reasoning shortcut" through the search space. The size of the backdoor can be seen as a distance measure that indicates how far the instance is from the target class. Backdoors were originally introduced by Williams, Gomes, and Selman [WGS03a; WGS03b] as a method for the theoretical analysis of decision heuristics in propositional satisfiability (cf. for a recent survey [GS12b]). Since then, backdoors have also been used in theoretical investigations in constraint satisfaction [GS08], abductive reasoning [PRS13], argumentation [DOS12], and quantified Boolean formulas [SS09a].

Exemplarily, we consider the propositional *satisfiability* problem (Sat). The problem is to decide whether a (propositional) formula is satisfiable. For a class $\mathcal{C}$ of formulas and a formula F, *membership* refers to the check whether F belongs to $\mathcal{C}$. We take a class $\mathcal{C}$ of formulas where membership and satisfiability are decidable in polynomial time, e.g., Horn formulas. Then, a set X of variables of a formula F is a *backdoor* into $\mathcal{C}$ if all formulas, that can be obtained from F by assigning the truth values 0 and 1 to the variables in X, yield simplified formulas which belong to the class $\mathcal{C}$. Once we have found a backdoor X, the formula F can be evaluated by considering all $2^{|X|}$ truth assignments to the variables in X. A main research interest in this thesis is to adapt the backdoor approach to the domain of Asp which seems most reasonable as backdoors have successfully been used to theoretically analyze the performance of Sat algorithms [WGS03a] and Asp and Sat are closely related (see e.g., [BD94; Fag94; LZ04a; LZ03; Jan06]). Backdoors allow us to develop exact algorithms which work efficiently for real-world instances with small backdoors. Since the backdoor approach is only useful on instances that have small backdoors and where we can find the backdoors significantly faster than by brute force search, we are also interested in potential target classes and the problem of determining backdoors efficiently. Another main research interest of this thesis is to exploit backdoors for problems in answer set programming beyond NP without making the problem itself tractable.

Unfortunately, classical complexity theory is not very useful to formally analyze ASP reasoning problems depending on certain backdoors of problem instances. A main reason is that classical complexity considers only the amount of a resource (e.g., time or space) in a function of the size of the input instance and does not distinguish whether an instance has a certain (hidden) structure. Therefore, we use for our asymptotic complexity analysis the framework of Parameterized Complexity [DF13], which takes the input size of an instance along with the structural properties (*the parameter*) of the input instance into account. A main concept of parameterized complexity theory is *fixed-parameter tractability*, which relaxes classical polynomial-time tractability in such a way that all non-polynomial parts depend only on the size of the parameter and not on the size of the input. It offers a framework for a more detailed theoretical analysis of the worst-case complexity of algorithms on real-world instances including hard algorithmic performance guarantees. Moreover, it explains gaps between classical theory and practice if the parameter is comparatively small.

1.3 Contribution

Our main contributions can be summarized as follows:

1. We develop a *rigorous theory of backdoors for answer set programming* (Theorem 3.12). We show that the concept of backdoors can be fruitfully adapted for this setting and that backdoors can serve as a *unifying framework* that accommodates several tractable restrictions of propositional ASP known from the literature.

2. We apply parameterized complexity theory for reasoning problems beyond NP and establish a fixed-parameter tractable reduction that reduces disjunctive ASP to normal ASP (Theorem 9.1); in other words, a reduction from the second level of the Polynomial Hierarchy to the first level (*complexity barrier breaking reductions*), using the size of a smallest backdoor into a normal program as the parameter.

More specifically, we provide the following novel contributions (after each chapter outline, we indicate where the results have been published):

1. In Chapter 3 we develop definitions and concepts for answer set programming backdoors. We point out major differences to propositional satisfiability. We introduce a backdoor approach for answer set programming, which consists of the steps backdoor detection and backdoor evaluation. We show that the most *important computational problems of propositional answer set programming*, including brave and skeptical reasoning, and even counting all answer sets, are *fixed-parameter tractable* when parameterized by the size of the backdoor into a tractable target class (where we can enumerate the set of all answer sets in polynomial time). [FS11; FS15b]

2. In Chapter 4 we show that *detecting backdoors is fixed-parameter tractable* for various target classes, including the class of all Horn programs and classes based on various notions of acyclicity. This way we make recent results of fixed-parameter algorithmics accessible to the field of answer set programming. [FS11; Fic12; FS15b]

3. In Chapter 5 we introduce a general method for *lifting* parameters, which are defined for disjunction-free (i.e., normal) programs only, to disjunctive programs. We apply our method to various parameters considered in the literature and obtain also fixed-parameter tractability under certain conditions. [FS15b]

4. In Chapter 6 we establish preprocessing methods in terms of *kernelization* for backdoor detection and establish kernel lower bounds for backdoor evaluation. These bounds provide worst case guarantees and limits for polynomial-time preprocessing for the considered problems. [FS15b]

5. In Chapter 7 we provide a more refined view on backdoors by means of search trees (*backdoor trees*) which allows us to significantly reduce the exponential blowup in the parameter in certain cases.

6. In Chapter 8 we compare the backdoor size with respect to various target classes with each other and with recently studied parameters, and we demonstrate that several structural restrictions considered in the literature can be stated in terms of backdoors. [FS15b]

7. In Chapter 9 we consider target classes where the problem of determining an answer set is already NP-hard. We establish a fixed-parameter tractable (fpt) reduction that reduces disjunctive Asp to Qbf-Sat where the universally quantified variables are bounded by the size of the backdoor which in turn can be fpt-reduced to Sat. In other words, we take advantage of the distance of a disjunctive program from being normal and we develop a reduction from the second level of the Polynomial Hierarchy to the first level (*complexity barrier breaking reductions*). This however, does not provide a reduction of the Asp reasoning problems for disjunctive logic programs into normal Asp in polynomial time, unless the (unlikely) collapse of the Polynomial Hierarchy. [FS13; FS15a]

8. In Chapter 10 we present some first experimental results where we consider the size of backdoors into Horn programs, normal programs, or acyclic programs based on acyclicity for typical benchmark instances (including structured and random programs of varied density). We also sketch ideas how backdoors could be used within a practical setting, in particular heuristics of an Asp solver. [FS11; FS13; FS15b; FS15a]

We conclude each chapter with a background on concepts, references, related work, insights and conclusions. The remaining chapters are organized as follows: In Chapter 2 we provide basic definitions, notations, and basic concepts used throughout this thesis.

We briefly introduce the reader to parameterized complexity, propositional satisfiability, answer set programming, and quantified Boolean formulas which are the main frameworks and problems of interest. In Chapter 11 we summarize the results, contextualize our contribution more broadly, and outline future research topics.

1.4 Publications

The following publications provide the basis for this thesis:

[FS15b] Johannes K. Fichte and Stefan Szeider. Backdoors to tractable answer-set programming. *Artificial Intelligence*, 220(0):64–103, 2015. ISSN 0004-3702.
Extended and updated version of a paper that appeared in Proceedings of the 22nd International Conference on Artificial Intelligence (IJCAI'11).

[FS15a] Johannes K. Fichte and Stefan Szeider. Backdoors to normality for disjunctive logic programs. 2015. *Submitted.*
Extended and updated version of a paper that appeared in Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI'13).

These publications combine and extend earlier work presented at conferences.

[FS11] Johannes K. Fichte and Stefan Szeider. Backdoors to tractable answer-set programming. In Toby Walsh, editor, *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, pages 863—868, Barcelona, Catalonia, Spain, July 2011. AAAI Press/IJCAI.

[Fic12] Johannes K. Fichte. The good, the bad, and the odd: Cycles in answer-set programs. In Daniel Lassiter and Marija Slavkovik, editors, Proceedings of the 23rd European Summer School in Logic, Language and Information (ESSLLI'11) and in New Directions in Logic, Language and Computation (ESSLLI'10 and ESSLLI'11 Student Sessions, Selected Papers Series), volume 7415 of *Lecture Notes in Computer Science*, pages 78–90. Springer Verlag, 2012. ISBN 978-3-642-31466-7.

[FS12] Johannes K. Fichte and Stefan Szeider. Backdoors to normality for disjunctive logic programs. *Proceedings of the 5th International Workshop Answer Set Programming and Other Computing Paradigms (ASPOCP'12)*, CoRR:abs/cs/1301.1391v2:99–114, September 2012. Preliminary version of the AAAI'13 paper below.

[FS13] Johannes K. Fichte and Stefan Szeider. Backdoors to normality for disjunctive logic programs. In Marie des Jardins and Michael Littman, editors, *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI'13)*, pages 320–327, Bellevue, WA, USA, July 2013. The AAAI Press. ISBN 978-1-57735-615-8.

The author has also presented summaries of is research at the following doctoral consortiums:

[Fic13a] Johannes K. Fichte. Backdoors to tractability of answer-set programming. In Peter McBurney and Ayanna Howard, editors, *Proceedings of the 27th AAAI Conference on Artificial Intelligence (Proceedings of the 18th AAAI-SIGART Doctoral Consortium) (AAAI DC'13)*, pages 1662—1663, 2013.

[Fic13b] Johannes K. Fichte. Backdoors to tractability of answer-set programming. In Martin Gebser and Marco Gavanelli, editors, *9th ICLP Doctoral Consortium (ICLP DC) — ICLP Technical Communications online supplement of Theory and Practice of Logic Programming (TPLP)*, volume 13, pages 1–10, 2013.

The author is a coauthor of the following publications which are not part of this thesis:

[AFT11] Albert Atserias, Johannes K. Fichte, and Marc Thurley. Clause-learning algorithms with many restarts and bounded-width resolution. *J. Artif. Intell. Res.*, 40:353–373, 2011. Extended and updated version of a paper that appeared in the Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT'09).

[FTW15] Johannes K. Fichte, Mirosław Truszczyński, Stefan Woltran. Dual-normal programs – The forgotten class. *Theory Pract. Log. Program.*. 2015. Proceedings of the 31st International Conference on Logic Programming (ICLP 2015). To appear.

CHAPTER 2

Preliminaries

In this chapter we give definitions and abbreviations that are used throughout the thesis. At the end of each section we provide some background on concepts, references, and related work. We start with an introduction to answer set programming in Section 2.1, followed by an introduction to classical complexity theory in Section 2.3 and parameterized complexity theory in Section 2.4. Finally, we present in Section 2.5 basic graph theoretical terminology including some subtle points that are important for the thesis.

2.1 Answer Set Programming

We consider a universe U of propositional *atoms*. A *literal* is an atom $a \in U$ or its negation $\neg a$. A *disjunctive logic program* (or simply a *program*) P is a set of *rules* of the form

$$a_1 \vee \ldots \vee a_l \leftarrow b_1, \ldots, b_n, \neg c_1, \ldots, \neg c_m$$

where $a_1, \ldots, a_l, b_1, \ldots, b_n, c_1, \ldots, c_m$ are atoms and l, n, m are non-negative integers. We write $H(r) = \{a_1, \ldots, a_l\}$ (the *head* of r), $B^+(r) = \{b_1, \ldots, b_n\}$ (the *positive body* of r), and $B^-(r) = \{c_1, \ldots, c_m\}$ (the *negative body* of r). We denote the sets of atoms occurring in a rule r or in a program P by $\mathrm{at}(r) = H(r) \cup B^+(r) \cup B^-(r)$ and $\mathrm{at}(P) = \bigcup_{r \in P} \mathrm{at}(r)$, respectively. We denote the number of rules of P by $|P| = |\{\, r : r \in P \,\}|$. The *size* $\|P\|$ of a program P is defined as $\sum_{r \in P} |H(r)| + |B^+(r)| + |B^-(r)|$.

A rule r is *positive* (basic/negation-free) if $B^-(r) = \emptyset$, r is *normal* if $|H(r)| \leq 1$, r is a *constraint* (integrity rule) if $|H(r)| = 0$, r is *constraint-free* if $|H(r)| > 0$, r is *Horn* if it is positive and normal or a constraint, r is *definite Horn* if it is Horn and constraint-free, and r is *tautological* if $B^+(r) \cap (H(r) \cup B^-(r)) \neq \emptyset$, and *non-tautological* if it is not tautological. We say that a program has a certain property if all its rules have the property. **Horn** refers to the class of all Horn programs. We denote the class

of all normal programs by **Normal**. Let P and P' be programs. We say that P' is a *subprogram* of P (in symbols $P' \subseteq P$) if for each rule $r' \in P'$ there is some rule $r \in P$ with $H(r') \subseteq H(r)$, $B^+(r') \subseteq B^+(r)$, $B^-(r') \subseteq B^-(r)$. Let $P \in$ **Horn**, we write $\mathrm{Constr}(P)$ for the set of constrains of P and $\mathrm{DH}(P) = P \setminus \mathrm{Constr}(P)$.

A set M of atoms *satisfies* a rule r if $(H(r) \cup B^-(r)) \cap M \neq \emptyset$ or $B^+(r) \setminus M \neq \emptyset$. M is a *model* of P if it satisfies all rules of P. The *Gelfond-Lifschitz (GL) reduct* of a program P under a set M of atoms is the program P^M obtained from P by first removing all rules r with $B^-(r) \cap M \neq \emptyset$ and then removing all $\neg z$ where $z \in B^-(r)$ from the remaining rules r [GL91]. M is an *answer set* (or *stable model*) of a program P if M is a minimal model of P^M. In other words, we consider a subset M of atoms of P as a "candidate" for an answer set. By default we interpret all atoms in M as "positive literals" and all others as "negative literals". The GL reduct establishes a semantics which threats the negative body of a rule in such a way that the positive literals naturally behave as exceptions for the implication, e.g., the rule $a \leftarrow b, \neg c$ reads as b implies a unless c belongs to M. The negative literals that occur in negative bodies of a rule have no effect on the rule (negative atoms are simply removed to construct P^M). The positive literals that occur in a literal of the negative body of a rule, however, effect the entire rule as an occurring exception (the entire rule is removed to construct P^M). M is an answer set if M is a minimal model after considering exceptions. We denote by $\mathrm{AS}(P)$ the set of all answer sets of P.

Example 2.1. Consider the disjunctive program

$$P = \left\{ \begin{array}{llllll} d & \leftarrow a, e; & a & \leftarrow d, \neg b, \neg c; & e \vee c & \leftarrow f; \\ f & \leftarrow d, c; & c & \leftarrow f, e, \neg b; & c & \leftarrow d; \\ b & \leftarrow c; & f & & & \end{array} \right\}.$$

The set $M = \{b, c, f\}$ is an answer set of P, since

$$P^M = \left\{ \begin{array}{llll} d & \leftarrow a, e; & e \vee c & \leftarrow f; \\ f & \leftarrow d, c; & c & \leftarrow d; \\ b & \leftarrow c; & f & \end{array} \right\}$$

and the minimal models of P^M are $\{b, c, f\}$ and $\{e, f\}$. ⊣

Example 2.2. Consider another disjunctive program

$$R = \left\{ \begin{array}{llllll} a \vee c & \leftarrow b; & b & \leftarrow c, \neg g; & c & \leftarrow a; \\ b \vee c & \leftarrow e; & h \vee i & \leftarrow g, \neg c; & a \vee b & ; \\ g & \leftarrow \neg i; & c & & & \end{array} \right\}.$$

The set $N = \{b, c, g\}$ is an answer set of R since

$$R^N = \left\{ \begin{array}{llll} a \vee c & \leftarrow b; & c & \leftarrow a; \\ b \vee c & \leftarrow e; & a \vee b & ; \\ g; & c & & \end{array} \right\}$$

and the minimal models of R^N are $\{b, c, g\}$ and $\{a, c, g\}$. ⊣

In the following, we restrict ourselves for simplicity of exposition to programs that do not contain any tautological rules. This restriction is not significant as tautological rules can be omitted from a program without changing its answer sets [BD98], i.e., $\mathrm{AS}(P) = \mathrm{AS}(P')$ where P' is a program obtained from P' by removing all tautological rules [BD98; Eit+04]. In one case we allow for tautological rules and state that explicitly (Proposition 4.15). Moreover, we generally assume that programs do not contain any rules r where $H(r) \cap B^-(r) \neq \emptyset$ since one can simply remove from such rules the head atoms in $H(r) \cap B^-(r)$ without effecting the answer sets. That is, $\mathrm{AS}(P) = \mathrm{AS}(P')$ where P' is a program obtained from P' by setting $H(r) := H(r) \setminus B^-(r)$ for every rule r where $H(r) \cap B^-(r) \neq \emptyset$ [Eit+04].

It is well known that normal Horn programs have a unique answer set or no answer set and that this set can be found in linear time. Van Emden and Kowalski [VK76] have shown that every constraint-free Horn program has a unique minimal model. The Emden-Kowalski operator of a program P and a subset A of atoms of P is the set $T_P(A) = \{\, a : a \in H(r), B^+(r) \subseteq A, r \in P \,\}$. The *least model* $LM(P)$ is the least fixed point of $T_P(A)$ [VK76]. Note that every definite Horn program P has a unique minimal model which equals the least model $LM(P)$ [GL88]. Dowling and Gallier [DG84] have established a linear-time algorithm for testing the satisfiability of propositional Horn formulas which easily extends to Horn programs. In the following, we state the well-known linear-time result.

Lemma 2.1. *Every Horn program has at most one minimal model which can be found in linear time.*

Asp Problems

We consider the following fundamental Asp problems.

Checking
Given: A program P and a set $M \subseteq \mathrm{at}(P)$.
Task: Decide whether M is an answer set of P.

Consistency
Given: A program P.
Task: Decide whether P has an answer set.

Brave Reasoning
Given: A program P and an atom $a \in \mathrm{at}(P)$.
Task: Decide whether a belongs to *some* answer set of P.

Skeptical Reasoning
Given: A program P and an atom $a \in \mathrm{at}(P)$.
Task: Decide whether a belongs to *all* answer sets of P.

Counting
Given: A program P.
Task: Compute the number of answer sets of P.

Enum
Given: A program P.
Task: List all answer sets of P.

We denote by *AspReason* the family of the reasoning problems Checking, Consistency, Brave Reasoning, and Skeptical Reasoning. We denote by *AspFull* the family of all the problems defined above. *AspReason* consists of decision problems, and *AspFull* adds to it a counting and an enumeration problem.

Background and Related Work

For an extensive introduction to answer set programming we refer to other sources [Geb+12; BET11; EIK09; Bar03; MT99]. Various definitions of answer set programming have been considered by Lifschitz [Lif10].

Answer set programming has its roots in logic programming such as Prolog [CR93], which attempts to develop a programming language where human knowledge is declaratively expressed by means of mathematical logic. There questions are posed in terms of queries to a system that executes the given program. A succeeding query yields an instantiation of the variables from which the solution of the query is extracted. Apt, Blair, and Walker [ABW88] and Gelder [Gel89] have established concepts for logic programs to handle exceptions and incomplete information in form of negations in the rule bodies without certain types of cycles in the graph representation of the program (see stratified programs in Section 4.2). Gelfond and Lifschitz [GL88] have proposed the stable model semantics for logic programming with negation and extended negation to non-stratified programs. The seemingly natural but, at the time, fundamentally novel condition of a stable model is minimality with respect to the GL reduct (see above) that is obtained from a given program by applying the current "knowledge" to negations. Since the stable model semantics is based on models and additional properties, a common solver obtains a solution by computing stable models that satisfy the program, in contrast to Prolog where the programming system interprets the program and extracts a solution from a successful run of the program. The stable model semantics forms the basis of answer set programming. The term answer set has originally been established by Gelfond and Lifschitz [GL91] to distinguish stable model semantics extended by classical negation from stable model semantics. Note that a program which additionally contains classical negation can be transformed into a program without classical negation (see, e.g., [EIK09; Geb+12]). The term *answer set programming* itself was coined by Marek and Truszczyński [MT99], Lifschitz [Lif99], and Niemelä [Nie99].

An aspect of answer set programming, which has contributed to its popularity as a declarative problem solving language, is the rich modelling language which includes extended and first-order programs. Extended programs contain extensions to propositional programs such as choice rules, cardinality rules, and weighted constraints [NSS99]. First-order answer set programs (also known as non-ground programs) allow the use of a restricted form of first-order variables. We would like to point out that we restrict ourselves in this thesis strictly to propositional answer set programming. However, extended rules (choice, cardinality, weighted constraints) can be transformed into normal rules [NSS99; JN11; BGJ14] and first-order variables are usually systematically instantiated by means of grounding techniques within common ASP solvers and hence a program that contains first-order variables is usually transformed into a program that contains only propositional variables [AHV95; GST07; Syr09]. Hence, the techniques presented in this thesis also apply when solving extended and first-order programs after transforming a given program into a propositional program.

Over the last years, various ASP solvers have been developed and tremendous gains have been made in the efficiency of solvers crucially conducted by making various techniques from SAT [SNS02; LZ04b; Leo+06; GKS12b] and CSP [Lie14] available to ASP solving. Many solvers utilize SAT solvers as black boxes or search techniques from SAT. There are solvers that deal with one or more fragments of disjunctive programs (normal, tight, or head-cycle-free), e.g., Smodels [SNS02; Sim08], Assat [LZ04b; Zha04], Cmodels2 [GLM06], and the original version of Clasp [Geb+11c; Kau+15]. There are also solvers that deal with the full set of disjunctive programs, e.g., Clasp [Dre+08a; GKS12b; GKS13; Kau+15], Cmodels3 [Lie05; Lie11], DLV [Leo+06; DLV12], GnT [Jan+06; Jan13b], and WASP [Alv+13]. Compilations to other problem domains and respective solvers have been considered for normal programs, e.g., propositional satisfiability [JN11; Jan13d], mixed integer programming [LJN12; Jan13a], and satisfiability modulo theories [JNS09; Jan13c; GJR14; Jan14].

ASP solvers and solving procedures have been considered from an abstract perspective. Lierler and Truszczyński [LT11] have characterized the core techniques of the ASP solvers Cmodels and Clasp (when restricted to normal programs) by means of transition systems and compared them to a representation of a solver for another declarative programming and knowledge representation approach, more specifically, the solver MiniSAT(ID)[1]. Further, they have established that the abstract representations of Clasp and MiniSAT(ID) are identical. Brochenin, Lierler, and Maratea [BLM14] have extended the transition system based approach to characterize ASP solvers that deal with the full set of disjunctive programs. Gebser and Schaub [GS13] have used tableaux-based proof systems to characterize solving techniques and strategies of ASP solvers by means of tableaux systems, compared the tableaux systems with each other, and established best-case proof complexity results. However, backjumping and learning schemes are not considered. Still, the effectiveness of the solvers is theoretically not well understood. Common solvers

[1] MiniSAT(ID) [Mar+08] is a solver for the logic PC(ID) logic [Den00], which extends propositional satisfiability by concepts of generalized inductive definitions.

use heuristics without non-trivial worst-case performance guarantees. We provide for the problems in *AspFull* and *AspReason*, respectively, theoretical worst-case time bounds that take certain hidden structures in disjunctive programs into account.

Many important problems of AI and reasoning can be succinctly represented and have successfully been solved within the Asp framework. Asp has been applied to several large industrial applications, e.g., match making [Geb+13b], planning [Ric+12; GKS12a; Pon+12], optimization of packaging of Linux distributions [Geb+11b], optimization [And+12; Geb+11b], robotics [APE12], scheduling [Ric+12], semantic web [Eit+12], social networks [JSS12], verification [Fal+12], and several other fields.

2.2 Propositional Logic and Quantified Boolean Formulas

We provide some notions from propositional logic and give basic background on quantified Boolean formulas. We consider a universe U of propositional variables. Note that we usually say *variable* instead of atom in the context of formulas. A literal is a variable x or its negation $\neg x$. A *clause* is a finite set of literals, interpreted as the disjunction of these literals. A propositional formula in *conjunctive normal form (CNF)* is a finite set of clauses, interpreted as the conjunction of its clauses. A *truth assignment* (or simply an *assignment*) is a mapping $\tau : X \to \{0, 1\}$ defined for a set $X \subseteq U$ of variables (atoms). For $x \in X$, we define $\tau(\neg x) = 1 - \tau(x)$. By 2^X we denote the set of all truth assignments $\tau : X \to \{0, 1\}$. By $\tau^{-1}(b)$ we denote the preimage $\tau^{-1}(b) := \{\, a : a \in X, \tau(a) = b \,\}$ of the truth assignment τ for some truth value $b \in \{0, 1\}$. The *truth assignment reduct* of a CNF formula F with respect to $\tau \in 2^X$ is the CNF formula F_τ obtained from F by first removing all clauses c that contain a literal set to 1 by τ, and then removing from the remaining clauses all literals set to 0 by τ. A truth assignment τ *satisfies* a given CNF formula F if $F_\tau = \emptyset$. Moreover, F is *satisfiable* if it is satisfied by some truth assignment τ. The definitions lead to the following fundamental problems of propositional logic:

Sat (propositional satisfiability)

Given: A propositional formula F.
Task: Decide whether F is satisfiable.

UnSat (propositional unsatisfiability)

Given: A propositional formula F.
Task: Decide whether F is unsatisfiable.

A *(prenex) quantified Boolean formula* Q is a formula of the form

$$Q_1 V_1 . Q_2 V_2 . \ldots Q_m V_m . F$$

where $Q_i \in \{\forall, \exists\}$, V_i are disjoint sets of propositional variables, and F is a propositional formula that contains only the variables in $\bigcup_{i=1}^{m} V_i$. We call the set V_i of variables a

quantifier block and the quantifier-free part F the *matrix* of Q. We say that Q is in *(prenex) conjunctive normal form* if its matrix is in conjunctive normal form (PCNF). The truth (evaluation) of quantified Boolean formulas is defined in the standard way [KL99]. These definitions lead to the following problem:

QBF-SAT (EVALUATION PROBLEM OF QUANTIFIED BOOLEAN FORMULAS)
Given: A quantified Boolean formula Q.
Task: Decide whether Q evaluates to true.

Well known fragments of QBF-SAT are $\forall\exists$-QBF-SAT and $\exists\forall$-QBF-SAT where the input is restricted to quantified Boolean formulas of the form $\forall V_1.\exists V_2.F$ and $\exists V_1.\forall V_2.F$, respectively, where F is a propositional formula that contains only the variables in $V_1 \cup V_2$.

Background and Related Work

For a comprehensive introduction and more detailed information we refer to other sources [KL99; Bie+09]. SAT and QBF-SAT have been intensively studied as theoretical problems [DP60; DLL62; Coo71; Sav70] and have been of strong practical interest since solving techniques significantly improved [MS99; ZM02; Mar99; LEV13] the effectiveness of modern SAT and QBF-SAT solvers. There has been wide research on the theoretical understanding of the effectiveness of modern SAT solving, e.g., from the perspective of computational and proof complexity [WGS03a; WGS03b; Nor06; Ans+08; BN08; Gom+08; Bie+09; PD11; AFT11; GS12b; Jär+12; BS14; Nor14; Fil+14], to understand the effectiveness of solving techniques [KSM11], and to understand correlations between characteristics of formulas and solving time [Hut+14; Ans+14; New+14]. However, there is still no clear understanding regarding the connection between theory and practice, in particular, why heuristics in modern solvers are so effective in practice [Var14].

Propositional ASP and propositional satisfiability are closely related. As mentioned in the previous section, a wide variety of problem solving techniques from SAT are utilized in ASP solving. However, the techniques have additionally been enhanced to handle the specific semantics of ASP. A key difference from the semantic point of view is that an atom in an ASP model needs a justification according to the definition of a stable model or, in other words, an atom is considered to be false unless otherwise proved.

Transformations of fragments of disjunctive programs into SAT have been considered by Ben-Eliyahu and Dechter [BD94] and Fages [Fag94]. Ben-Eliyahu and Dechter [BD94] have introduced head-cycle-free programs where the problems in *AspReason* are NP-complete and co-NP-complete, respectively. Fages [Fag94] has shown that the answer sets of a tight program (programs where certain cycles on a graph representation of the given program are forbidden) are exactly the models of Clark's completion [Cla78]. The definition of tight programs has been generalized to disjunctive programs by means of the concept of loop formulas by Lee and Lifschitz [LL03]. Gebser and Schaub [GS05] have introduced elementary loops, which improve loop formulas for normal programs.

Gebser, Lee, and Lierler [GLL11] have extended elementary loops to disjunctive programs, considered connections between maximal elementary loops and minimal unfounded sets, and introduced head-elementary-loop-free programs and a transformation of these programs into normal programs. Ji et al. [Ji+14] have established for normal programs an alternative formulation for elementary loops and the concept of proper loops, which improves the concept of elementary loops. Then, Ji, Wan, and Xiao [JWX15] have presented a relaxed notion for elementary and proper loops for disjunctive programs. Lin and Zhao [LZ03] have established a transformation of normal programs into propositional formulas that is based on the characterization of inherent tightness[2] and may produce additional new variables. Janhunen [Jan06] has suggested a transformation from normal programs into propositional formulas. Janhunen et al. [Jan+06] have given a transformation from disjunctive programs into propositional formulas for programs where the number of disjunctions in the heads of rules is bounded. The transformation provides a Sat encoding that consists of a guess and check approach. Transformations into related problem domains have consequently been designed, e.g., Sat modulo theories [JNS09; JLN11; GJR14] and mixed integer linear programming [LJN12].

The transformations mentioned in the previous paragraph have also been considered under the aspect of compactness and modularity, in particular, to analyze the expressiveness of answer set programming from the modelling perspective. A transformation t is called *modular* if $t(F) = P$ and $t(F') = P'$, then $t(F \cup F') = P \cup P'$ where F and F' are propositional CNF formulas (programs) and P and P' are the resulting programs (formulas). Niemelä [Nie99] has presented a modular transformation which transforms propositional formulas into normal programs and introduces a linear number of fresh atoms and rules. The transformation by Ben-Eliyahu and Dechter [BD94], which transforms normal programs into propositional formulas, produces a quadratic number of fresh atoms and a cubic number of fresh rules. The transformation by Lin and Zhao [LZ04a] introduces no fresh atom but in worst case an exponential number of new rules. Another transformation by Lin and Zhao [LZ03] is modular and introduces a quadratic number of fresh atoms and rules. The transformation by Janhunen [Jan06] produces a sub-quadratic number of fresh atoms. However, it is currently an open question whether there is also a linear transformation of normal programs into propositional formulas. Further, Lifschitz and Razborov [LR06] have established that under the widely believed complexity theoretical assumption $\mathrm{P} \not\subseteq \mathrm{NC}^1/\mathrm{poly}$ [Pap94; AB09] either the transformation needs to introduce new atoms or the size of the resulting propositional formula will be significantly larger than the input program. We will discuss the computational complexity of the main Asp reasoning problems in the following Sections 2.3 and 2.4.

[2] A set X of atoms is inherent tight in a program P if there is a tight subprogram P' of P where the set X is supported by P'.

2.3 Classical Computational Complexity

We assume that the reader is familiar with the main concepts of computational complexity theory, in particular, algorithms, (decision) problems, and complexity classes [Pap94; AB09]. In the following, we briefly recall some notations and definitions.

We use the asymptotic notation $\mathcal{O}(\cdot)$ and the big omega notation $\Omega(\cdot)$ in the standard way. Let Σ and Σ' be some finite alphabets. We call $I \in \Sigma^*$ an *instance* and $\|I\|$ denotes the size of I. Let $L \subseteq \Sigma^*$ and $L' \subseteq \Sigma'^*$ be problems. We sometimes call an instance $I \in L$ a *yes-instance* and an instance $I \notin L$ a *no-instance*. A *(non-deterministic) polynomial-time Turing reduction* from L to L' is an (non-deterministic) algorithm that decides in time $\mathcal{O}(\|I\|^c)$ for some constant c whether $I \in L$ using L' as an oracle. Polynomial-time Turing reductions are also known as Cook reductions. Let $L \subseteq \Sigma^*$ and $L' \subseteq \Sigma'^*$ be decision problems. A *polynomial-time many-to-one reduction (or simply polynomial-time reduction)* from L to L' is a function $r : \Sigma^* \to \Sigma'^*$ such that for all $I \in \Sigma^*$ we have $I \in L$ if and only if $r(I) \in L'$ and r is computable in time $\mathcal{O}(\|I\|^c)$ for some constant c. In other words, a polynomial-time many-to-one reduction transforms instances of decision problem L into instances of decision problem L' in polynomial time. Polynomial-time many-to-one reductions are also known as *Karp reductions*. Note that such reductions are in fact stronger than polynomial-time Turing reductions since an oracle can be used only once considering L' as an oracle.

A problem L is *(non-deterministically) polynomial-time solvable* if there exists a constant c such that we can decide by an (non-deterministic) algorithm whether $I \in L$ in time $\mathcal{O}(\|I\|^c)$. Such an algorithm is called a *(non-deterministic) polynomial-time algorithm*. If L is a decision problem, we identify L with the set of all yes-instances I.

- P is the class of all polynomial-time solvable decision problems.
- NP is the class of all non-deterministically polynomial-time solvable decision problems.

Let C be a complexity class, e.g., NP. Then co-C denotes the class of all decision problems whose complement (the same problem with yes and no answers swapped) is in C. We say that a problem L is C-hard if there is a polynomial-time reduction for every problem $L' \in$ C to L. If in addition $L \in$ C, then L is C-complete. For instance, a problem is NP-complete if it belongs to NP and all problems in NP have polynomial-time reductions to it. We are also interested in the Polynomial Hierarchy [SM73; Sto76; Wra76; Pap94] up to the second level. The Polynomial Hierarchy consists of complexity classes Σ_i^p for $i \geq 0$ based on the following definitions: $\Sigma_0^p := P$ and $\Sigma_{i+1}^p = \mathrm{NP}^{\Sigma_i^p}$ for all $i \geq 0$ where NP^{C} denotes the class of all decision problems such that there is a polynomial-time Turing reduction to any decision problem $L \in$ C, i.e., a decision problem $L' \in \mathrm{NP}^{\mathrm{C}}$ is non-deterministically polynomial-time solvable using any problem $L \in$ C as an oracle. Moreover, $\Pi_k^p :=$ co-Σ_k^p. Note that NP $= \Sigma_1^p$, co-NP $= \Pi_1^p$,

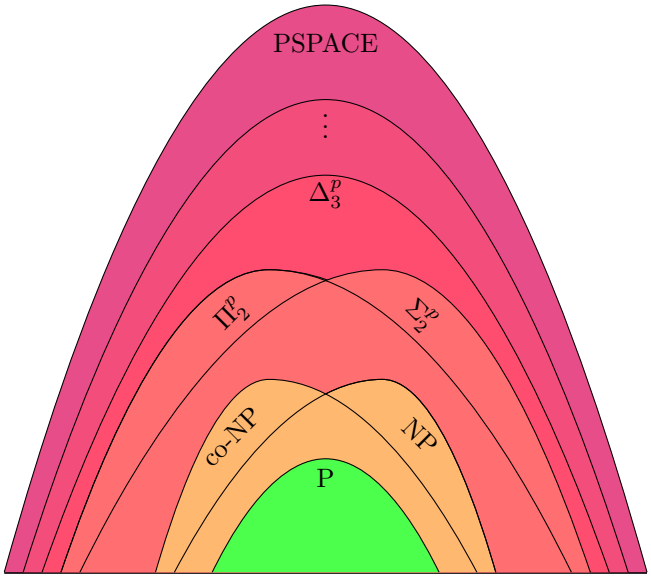


Figure 2.1: Illustration of the Polynomial Hierarchy.

$\Sigma_2^p = \text{NP}^{\text{NP}}$, and $\Pi_2^p = \text{co-NP}^{\text{NP}}$. Figure 2.1 illustrates the relationship between the complexity classes of the Polynomial Hierarchy.

$\#P$ is the complexity class consisting of all counting problems associated with decision problems in NP [Val79]. The (non-uniform) complexity class NP/poly consists of all problems for which there exists a problem $L' \in \text{NP}$ and a function $a : \mathbb{N} \to \Sigma^*$ with $|a(n)| \in \mathcal{O}(n^c)$ for some constant c such that for all $I \in \Sigma^*$, $I \in L$ if and only if $(I, a(\|I\|)) \in L'$. In other words, the complexity class NP/poly consists of all problems that can be solved non-deterministically in non-uniform polynomial time, i.e., by a non-deterministic polynomial-time algorithm with an additional polynomial-bounded input that depends only on the length of the input but not on the input itself (advice function). Note that NP/poly consists of the problems that can be decided by non-deterministic Boolean circuits of polynomial size. For details on the non-uniform complexity classes, we refer to other sources [KL80].

Answer Set Programming

The computational complexity of various problems arising in answer set programming has been the subject of extensive studies. Eiter and Gottlob [EG95] have established that the main decision problems of disjunctive answer set programming are complete for the second level of the Polynomial Hierarchy, i.e., Consistency and Brave Reasoning are Σ_2^p-complete and Skeptical Reasoning is Π_2^p-complete [EG95]. Checking is co-NP-complete in general [EG95]. Counting is easily seen to be $\#P$-hard. Several fragments of programs where Asp problems are non-deterministically polynomial-time solvable or even polynomial-time solvable have been identified. When the input is re-

stricted to normal programs Consistency and Brave Reasoning are NP-complete, Skeptical Reasoning is co-NP-complete [BF91; MT91a], and Checking is polynomial-time solvable [CL94]. Several fragments of programs where problems in *AspFull* are polynomial-time tractable have been identified, e.g., Horn programs [GL88], stratified programs [ABW88] and programs without even cycles [Zha02]. Lonc and Truszczyński [LT06] have introduced algorithms and complexity results for finding all stable models of disjunctive programs.

Dantsin et al. [Dan+01] have provided a survey for classical complexity of the main reasoning problems for various semantics of logic programming, including fragments of propositional answer set programs and first-order answer set programs. Minker [Min93] has provided an overview on various semantics of non-monotonic logic programming and its computational complexity including stable models. Truszczyński [Tru11] has classified the computational complexity of the main Asp reasoning problems, when the input is restricted to certain classes of programs, in a trichotomy (P, first level, and second level of PH) similar to dichotomy results in propositional satisfiability (see results by Schaefer [Sch78]).

Sat and Qbf-Sat

Cook [Coo71] and Levin [Lev73] have established that the Sat problem is NP-complete. The problem Qbf-Sat is PSPACE[3]-complete and is therefore believed to be computationally harder than Sat [KL99; Pap94; SM73; Wra76]. The problem $\exists\forall$-Qbf-Sat is Σ_2^p-complete and the complement $\forall\exists$-Qbf-Sat is Π_2^P-complete [Pap94].

2.4 Parameterized Complexity

Problem instances that originate in practical applications are often structured in a way that facilitates obtaining a solution relatively fast. Such instances seem to be harder in theory than they are in practice, in particular, since classical complexity theory mainly considers worst-case bounds as a function of the size of the input instance. Downey and Fellows [DF99] introduced in a series of papers *parameterized complexity*, which also takes structural properties of problem instances in form of a parameter into account. In consequence, parameterized complexity offers a framework for a more detailed theoretical analysis that can be closer to the practical hardness of problems. Since there are many ways of defining and capturing structure in a problem instance, there are also various ways to parameterize a problem. A main concept of parameterized complexity theory is *fixed-parameter tractability*, which relaxes classical polynomial-time tractability in such a way that all non-polynomial parts depend only on the size of the parameter and not on the size of the input.

We briefly give some background on parameterized complexity. Figure 2.2 (right side) illustrates the Hierarchy of the parameterized complexity classes as defined in the

[3]PSPACE contains all polynomial-space solvable decision problems. Note that $\bigcup_{i=0}^{\infty} \Pi_i^p \subseteq$ PSPACE.

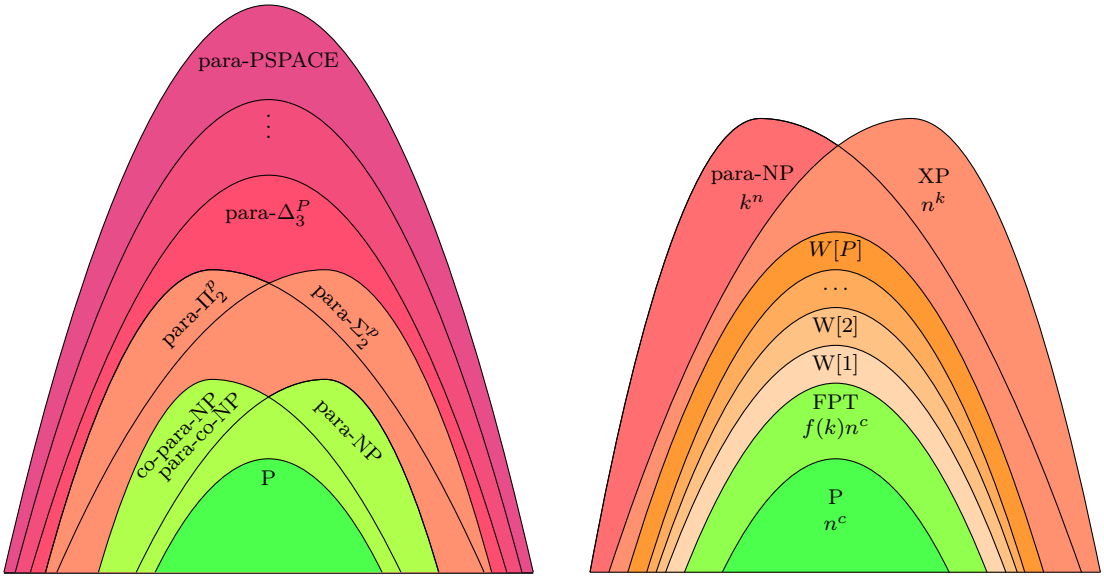


Figure 2.2: Illustration of the relationship of parameterized complexity classes. para-C complexity classes based on the Polynomial Hierarchy (left), which are interesting for problems that are located beyond NP, and complexity classes up to para-NP and XP (right), which are interesting for NP-complete problems. n refers to the size of the input and k to the parameter.

following. For more detailed information we refer to other sources [DF99; DF13; FG06; GS08; Nie06]. An instance of a *parameterized problem* L is a pair $(I, k) \in \Sigma^* \times \mathbb{N}$ for some finite alphabet Σ. For an instance $(I, k) \in \Sigma^* \times \mathbb{N}$ we call I the *main part* and k the *parameter*. $\|I\|$ denotes the size of I. L is *fixed-parameter tractable* if there exist a computable function f and a constant c such that we can decide by an algorithm whether $(I, k) \in L$ in time $\mathcal{O}(f(k)\|I\|^c)$. Such an algorithm is called an *fpt-algorithm*. If L is a decision problem, then we identify L with the set of all yes-instances (I, k). FPT is the class of all fixed-parameter tractable decision problems.

Let $L \subseteq \Sigma^* \times \mathbb{N}$ and $L' \subseteq \Sigma'^* \times \mathbb{N}$ be two parameterized decision problems for some finite alphabets Σ and Σ'. An *fpt-reduction* r from L to L' is a many-to-one reduction from $\Sigma^* \times \mathbb{N}$ to $\Sigma'^* \times \mathbb{N}$ such that for all $I \in \Sigma^*$ we have $(I, k) \in L$ if and only if $r(I, k) = (I', k') \in L'$ such that $k' \leq g(k)$ for a fixed computable function $g : \mathbb{N} \to \mathbb{N}$ and there is a computable function f and a constant c such that r is computable in time $O(f(k)\|I\|^c)$ [FG06]. Thus, an fpt-reduction is, in particular, an fpt-algorithm. It is easy to see that the class FPT is closed under fpt-reductions. It is clear for parameterized problems L_1, and L_2 that if $L_1 \in$ FPT and there is an fpt-reduction from L_2 to L_1, then $L_2 \in$ FPT. We would like to note that the theory of fixed-parameter intractability is based on fpt-reductions [DF99; DF13; FG06].

The *Weft Hierarchy* consists of parameterized complexity classes $\mathrm{W}[1] \subseteq \mathrm{W}[2] \subseteq \cdots$ which are defined as the closure of certain parameterized problems under parameterized reductions. There is strong theoretical evidence that parameterized problems that are hard for classes $\mathrm{W}[i]$ are not fixed-parameter tractable. A prominent $W[2]$-complete problem is Hitting Set [DF99; DF13] defined as follows:

Hitting Set

Given:	A family of sets (S, k) where $\mathsf{S} = \{S_1, \ldots, S_m\}$ and an integer k.
Parameter:	The integer k.
Task:	Decide whether there exists a set H of size at most k which intersects with all the S_i (H is a *hitting set* of S).

The class XP of *non-uniform* polynomial-time tractable problems consists of all parameterized decision problems that can be solved in polynomial time if the parameter is considered constant. That is, $(I, k) \in L$ can be decided in time $\mathcal{O}(\|I\|^{f(k)})$ for some computable function f.

Parameterized complexity theory also offers complexity classes for problems that lie higher in the polynomial hierarchy. Let C be a classical complexity class, e.g., NP. The parameterized complexity class para-C is then defined as the class of all parameterized problems $L \subseteq \Sigma^* \times \mathbb{N}$, for some finite alphabet Σ, for which there exist an alphabet Π, a computable function $f : \mathbb{N} \to \Pi^*$, and a problem $P \subseteq \Sigma^* \times \Pi^*$ such that $P \in C$ and for all instances $(x, k) \in \Sigma^* \times \mathbb{N}$ of L we have that $(x, k) \in L$ if and only if $(x, f(k)) \in P$. Intuitively, the class para-C consists of all problems that are in C after a precomputation that only involves the parameter [FG03]. The class para-NP can also be defined via non-deterministic fpt-algorithms, i.e., para-NP contains all parameterized decision problems L such that $(I, k) \in L$ can be decided *non-deterministically* in time $\mathcal{O}(f(k)\|I\|^c)$ for some computable function f and constant c [FG06]. A parameterized decision problem is para-NP-complete if it is in NP and NP-complete when restricted to finitely many parameter values [FG06]. co-para-NP denotes the class of all parameterized decision problems whose complement (the same problem with yes and no answers swapped) is in para-NP. Using the concepts and terminology of Flum and Grohe [FG06], co-para-NP = para-co-NP. Figure 2.2 (left side) illustrates the relationship between the para-$\mathcal{C}$ complexity classes for classes $\mathcal{C}$ in the Polynomial Hierarchy.

An example for a para-NP-complete problem is the satisfiability problem, where the parameter value is ignored (or simply associating with every formula the parameter 0).

Sat

Given:	A propositional formula F and an integer k.
Parameter:	The integer k.
Task:	Is F satisfiable?

Similarly we get a co-para-NP-complete problem.

UnSat
Given: A propositional formula F and an integer k.
Parameter: The integer k.
Task: Is F unsatisfiable?

Another para-NP-complete problem is the satisfiability problem, where the parameter is the maximum number of literals in a clause [FG06].

Max-Lits-Sat
Given: A propositional formula F.
Parameter: The maximum number of literals in a clause of F.
Task: Is F satisfiable?

We observe hardness from the fact that the propositional satisfiability problem is already NP-complete for input formulas that contain at most 3 literals in each clause (3-CNFs), and hence Max-Lits-Sat is hard for the parameter value 3.

The complexity class para-NP can be seen as an analogue of NP in parameterized complexity. Since a parameterized decision problem is para-NP-complete if it is in NP and NP-complete when restricted to finitely many parameter values, the complexity class para-NP is not very interesting for parameterizations of NP-complete problems [FG06]. However, for parameterizations of problems that are harder than NP (like the main reasoning problems of propositional disjunctive Asp) para-NP-completeness is a desirable property as it allows us to exploit the parameter to solve the problem for small parameter values more efficiently.

An example of a para-Σ_2^P-complete problem is the consistency problem of answer set programming, which is to decide whether a given disjunctive program P has an answer set, where the parameter is the maximum number of occurrences of an atom in the program P [DS14b].

Max-Occurrence-Consistency
Given: A disjunctive program P.
Parameter: The maximum number of occurrences of an atom P.
Task: Does P have an answer set?

Given a parameterized problem L on some finite alphabet Σ. The *unparameterized version* of L is the classical problem $\{ I\#u^k : (I,k) \in L \}$ where u denotes an arbitrary symbol from Σ and $\#$ is a new symbol not in Σ.

Background and Related Work

Research on parameterized complexity theory has been initiated by Downey et al. [DFS99; DF99; DF13] and extended, among others, by Flum and Grohe [FG06], Niedermeier [Nie06], and Cygan et al. [Cyg+15]. Various parameters have been suggested, e.g.,

treewidth [RS84; Bod93], size of the solution, cliquewidth [CO00], size of a smallest backdoor [WGS03a; WGS03b], Boolean width [Adl+10]. Gottlob, Scarcello, and Sideri [GSS02] have provided fixed-parameter tractability results of several problems in artificial intelligence and non-monotonic reasoning. Gottlob and Szeider [GS08] presented a survey on parameterized complexity of problems in artificial intelligence, database theory and automated reasoning. Gaspers and Szeider [GS14] have established results on polynomial-time preprocessing in terms of kernelization lower bounds for problems in artificial intelligence. Tools to establish lower bounds for kernel size have been developed by Fortnow and Santhanam [FS11]. A general theoretical framework to classify parameterized problems on whether they admit an fpt-reduction to Sat or not has lately been introduced by DeHaan and Szeider [DS14b].

So far there has been no rigorous study of disjunctive Asp within the framework of parameterized complexity. However, several results known from the literature can be stated in terms of parameterized complexity. Some of these results already provide fixed-parameter tractability. The considered parameters include the number of atoms of a normal program that occur in negative rule bodies [Ben96], the number of non-Horn rules of a normal program [Ben96], the size of a smallest feedback vertex set in the dependency digraph of a normal program [GSS02], the number of cycles of even length in the dependency digraph of a normal program [LZ04a], the treewidth of the incidence graph of a normal program [JPW09; Mor+10], and a combination of two parameters: the length of the longest cycle in the dependency digraph and the treewidth of the interaction graph of a head-cycle-free program [BD94].

2.5 Graphs

We recall some graph theoretical notations. We consider undirected and directed graphs. An *undirected graph* or simply a *graph* is a pair $G = (V, E)$ where $V \neq \emptyset$ is a set of *vertices* and $E \subseteq \{\, \{u, v\} \subseteq V : u \neq v \,\}$ is a set of *edges*. We denote an edge $\{u, v\}$ by uv or vu. A graph $G' = (V', E')$ is a *subgraph* of G if $V' \subseteq V$ and $E' \subseteq E$ and an *induced subgraph* if additionally for any $u, v \in V'$ and $uv \in E$ also $uv \in E'$. A *path of length* k is a graph with $k + 1$ pairwise distinct vertices $v_1, \ldots, v_{k+1}$, and k distinct edges $v_i v_{i+1}$ where $1 \leq i \leq k$ (possibly $k = 0$). A *cycle of length* k is a graph that consists of k distinct vertices $v_1, v_2, \ldots, v_k$ and k distinct edges $v_1 v_2, \ldots, v_{k-1} v_k, v_k v_1$. Let $G = (V, E)$ be a graph. G is *bipartite* if the set V of vertices can be divided into two disjoint sets U and W such that there is no edge $uv \in E$ with $u, v \in U$ or $u, v \in W$. G is *complete* if for any two vertices $u, v \in V$ there is an edge $uv \in E$. G contains a *clique* on $V' \subseteq V$ if the induced subgraph (V', E') of G is a complete graph. A *connected component* C of G is an inclusion-maximal subgraph $C = (V_C, E_C)$ of G such that for any two vertices $u, v \in V_C$ there is a path in C from u to v. We say G is a *tree* if it is a connected component $C = G$ and G contains no cycles. We usually call the vertices of a tree *nodes*.

A *directed graph* or simply a *digraph* is a pair $G = (V, E)$ where $V \neq \emptyset$ is a set of vertices and $E \subseteq \{\, (u, v) \in V \times V : u \neq v \,\}$ is a set of *directed edges*. A digraph $G' = (V', E')$

is a *subdigraph* of G if $V' \subseteq V$ and $E' \subseteq E$ and an *induced subdigraph* if additionally for any $u, v \in V'$ and $(u, v) \in E$ also $(u, v) \in E'$. For a vertex $v \in V$ we call a vertex $v \in \{ w : (w, u) \in E \}$ a *predecessor of* u and a vertex $u \in \{ w : (v, w) \in E \}$ a *successor of* v. A *directed path of length* k is a digraph with $k + 1$ pairwise distinct vertices $v_1, \ldots, v_{k+1}$, and k distinct edges (v_i, v_{i+1}) where $1 \leq i \leq k$ (possibly $k = 0$). A *directed cycle of length* k is a digraph that consists of k distinct vertices $v_1, v_2, \ldots, v_k$ and k distinct edges $(v_1, v_2), \ldots, (v_{k-1}, v_k), (v_k, v_1)$.

We sometimes denote a (directed) path or (directed) cycle as a sequence of vertices. We would like to point out that according to the above definitions, the length of an undirected cycle is at least 3, whereas the length of a directed cycle is at least 2.

A *strongly connected component* C of a digraph $G = (V, E)$ is an inclusion-maximal directed subgraph $C = (V_C, E_C)$ of G such that for any two vertices $u, v \in V_C$ there are paths in C from u to v and from v to u. The strongly connected components of G form a partition of the set V of vertices, we denote this partition by $\mathrm{SCC}(G)$.

An *underlying graph* of a digraph $G = (V, E)$ is the graph G' that is obtained by replacing each edge $(u, v) \in E$ with an edge uv. A *binary tree* $T = (V, E, r)$ consists of (i) a digraph (V, E) whose underlying graph is a tree, (ii) a designated vertex r which has no predecessor, and (iii) each vertex $v \in V$ has either no or two successors. We call a vertex $v \in V$ *node of* T, a successor of a node *child*, the vertex r the *root of* T, and a vertex v that has no child a *leaf of* T.

Background and Related Work

For further basic terminology on graphs and digraphs, we refer to standard texts [Die12; BM08; BG09].

CHAPTER 3

Backdoors

An interesting "hidden structure" in answer set programs are certain atoms that provide a *backdoor* into some fixed *target class of programs* for which the computational problem under consideration is computationally easier, mostly polynomial-time tractable in previous work. In this chapter we develop definitions and concepts for answer set programming backdoors. We show that the most important computational problems of propositional answer set programming are fixed-parameter tractable when parameterized by the size of the backdoor into a fixed tractable target class. Sometimes target classes are also referred to as *"islands of tractability"* within the classes where problems are intractable [GS12b]. By means of a backdoor one tries to identify structural hard parts of a problem instance to reach an island of tractability. Hence, the size of the backdoor can be seen as a distance measure that indicates how far the instance is from the target class. When we exploit backdoors to solve a problem we have to find a backdoor of the given instance (*backdoor detection*) and then apply the found backdoor to the instance and determine the solution (*backdoor evaluation*).

In Section 3.1 we exemplarily consider backdoors for the propositional satisfiability problem where backdoors originate from [WGS03a; WGS03b]. In Section 3.2 we present basic concepts like truth assignment reduct, strong backdoors, and deletion backdoors for answer set programs. Subsequently, we develop a method to solve the main ASP reasoning problems using backdoors (backdoor evaluation for ASP) in Section 3.3. It turns out that the evaluation problem is more complicated than for propositional satisfiability. We show, however, that the most important computational problems of propositional ASP, including brave and skeptical reasoning, and even counting all answer sets, are fixed-parameter tractable when parameterized by the size of the backdoor with respect to various target classes. In Section 3.4 we introduce basic terminology for ASP backdoor detection. We conclude the chapter with background and related work, a summary of our contribution, and a discussion. This chapter is based on published work [FS15b].

3.1 Satisfiability Backdoors

In this section we provide an introduction to the concept of backdoors based on the propositional satisfiability problem (see e.g., [GS12b]).

The following is obvious from the definitions:

Observation 3.1. *Let F be a CNF formula and X a set of atoms. F is satisfiable if and only if F_τ is satisfiable for at least one truth assignment $\tau \in 2^X$.*

This leads to the definition of a strong backdoor relative to a class $\mathcal{C}$ of polynomially solvable CNF formulas: a set X of atoms is a *strong $\mathcal{C}$-backdoor* of a CNF formula F if $F_\tau \in \mathcal{C}$ for all truth assignments $\tau \in 2^X$. Assume that the satisfiability of formulas $F \in \mathcal{C}$ of size $\|F\| = n$ can be decided in time $\mathcal{O}(n^c)$ for some constant c. Then we can decide the satisfiability of an arbitrary formula F for which we know a strong $\mathcal{C}$-backdoor of size k in time $\mathcal{O}(2^k n^c)$ for some constant c, which is efficient as long as k remains small.

A further variant of backdoors are deletion backdoors defined by removing literals from a CNF formula. $F - X$ denotes the formula obtained from F by removing all literals $x, \neg x$ for $x \in X$ from the clauses of F. Then a set X of atoms is a *deletion $\mathcal{C}$-backdoor* of F if $F - X \in \mathcal{C}$. In general, deletion $\mathcal{C}$-backdoors are not necessarily strong $\mathcal{C}$-backdoors. If all subsets of a formula in $\mathcal{C}$ also belong to $\mathcal{C}$ ($\mathcal{C}$ is clause-induced), then deletion $\mathcal{C}$-backdoors are strong $\mathcal{C}$-backdoors.

Before we can use a backdoor we need to find it first. What we call the *backdoor approach* is a process consisting of the following two phases:

- finding a backdoor (*backdoor detection*) and
- using the backdoor to solve the problem (*backdoor evaluation*).

For most reasonable target classes $\mathcal{C}$ the detection of a strong $\mathcal{C}$-backdoor of size at most k is NP-hard if k is part of the input. However, as we are interested in finding *small* backdoors, it makes sense to parameterize the backdoor search by k and consider the parameterized complexity of backdoor detection. Indeed, with respect to the classes of Horn CNF formulas and 2-CNF formulas, the detection of strong backdoors of size at most k is fixed-parameter tractable [NRS04]. The parameterized complexity of backdoor detection for many further target classes has been investigated [GS12b].

The purpose of the following sections is to develop a backdoor approach for answer set programming. It turns out that the evaluation problem is more complicated than for propositional satisfiability (see Section 3.3). Later in Chapters 4 and 9 we will see that various target classes for answer set programming require new algorithms for backdoor detection (see in particular Section 4.2).

3.2 Backdoors of Answer Set Programs

In order to translate the notion of backdoors to the domain of answer set programming, we first need to come up with a suitable concept of a reduction with respect to a truth assignment. The following is a natural definition which generalizes a concept of Gottlob, Scarcello, and Sideri [GSS02].

Definition 3.2. *Let P be a program, X a set of atoms, and $\tau \in 2^X$. The* truth assignment reduct *of P under τ is the logic program P_τ obtained from P by*

1. *removing all rules r with $H(r) \cap \tau^{-1}(1) \neq \emptyset$ or $H(r) \subseteq X$;*
2. *removing all rules r with $B^+(r) \cap \tau^{-1}(0) \neq \emptyset$;*
3. *removing all rules r with $B^-(r) \cap \tau^{-1}(1) \neq \emptyset$;*
4. *removing from the heads and bodies of the remaining rules all literals $a, \neg a$ with $a \in X$.*

Definition 3.3. *Let $\mathcal{C}$ be a class of programs. A set X of atoms is a* strong $\mathcal{C}$-backdoor *of a program P if $P_\tau \in \mathcal{C}$ for all truth assignments $\tau \in 2^X$.*

By a *minimal* strong $\mathcal{C}$-backdoor of a program P we mean a strong $\mathcal{C}$-backdoor of P that does not properly contain a smaller strong $\mathcal{C}$-backdoor of P; a *smallest* strong $\mathcal{C}$-backdoor of P is one of smallest cardinality.

Example 3.1. Consider program P from Example 2.1. The set $\{b, c\}$ is a strong **Horn**-backdoor since all four truth assignment reducts $P_{\neg b \neg c} = P_{b=0,c=0} = \{d \leftarrow a, e;\ a \leftarrow d;\ e \leftarrow f;\ f\}$, $P_{\neg bc} = \{d \leftarrow a, e;\ f \leftarrow d;\ f\}$, $P_{b\neg c} = \{d \leftarrow a, e;\ e \leftarrow f;\ f\}$, and $P_{bc} = \{d \leftarrow a, e;\ f \leftarrow d;\ f\}$ are in the class **Horn**. ⊣

Example 3.2. Consider the program R from Example 2.2. The set $\{b, c, h\}$ is a strong **Normal**-backdoor since the truth assignment reducts $R_{\neg b \neg c \neg h} = R_{b=0,c=0,h=0} = \{\, i \leftarrow g;\ a;\ g \leftarrow \neg i \,\}$, $R_{\neg b \neg c h} = R_{\neg b c \neg h} = R_{\neg b c h} = R_{b \neg c h} = \{\, a;\ g \leftarrow \neg i \,\}$, $R_{b \neg c \neg h} = \{\, a;\ i \leftarrow g;\ g \leftarrow \neg i \,\}$, and $R_{bc\neg h} = P_{bch} = \{\, g \leftarrow \neg i \,\}$ are in the class **Normal**. ⊣

Next we define a variant of answer set backdoors similar to satisfiability deletion backdoors.

Definition 3.4. *For a program P and a set X of atoms we define $P - X$ as the program obtained from P by deleting $a, \neg a$ for $a \in X$ from the rules of P.*

The definition gives rise to deletion backdoors. We will see that finding deletion backdoors is in some cases easier than finding strong backdoors.

Definition 3.5. *Let $\mathcal{C}$ be a class of programs. A set X of atoms is a* deletion $\mathcal{C}$-backdoor *of a program P if $P - X \in \mathcal{C}$.*

In general, not every strong $\mathcal{C}$-backdoor is a deletion $\mathcal{C}$-backdoor, and not every deletion $\mathcal{C}$-backdoor is a strong $\mathcal{C}$-backdoor. But we can strengthen one direction requiring the target class to satisfy the very mild condition of being hereditary.

Definition 3.6. *Let $\mathcal{C}$ be a class of programs. We call a class $\mathcal{C}$ of programs* hereditary *if for each $P \in \mathcal{C}$ all subprograms of P are in $\mathcal{C}$ as well (see Section 2.1 for definition of subprograms).*

Remark. Note that many natural classes of programs (and all classes considered in this paper) are hereditary.

Lemma 3.7. *If $\mathcal{C}$ is hereditary, then every deletion $\mathcal{C}$-backdoor is a strong $\mathcal{C}$-backdoor.*

Proof. Let P be a program, $X \subseteq \mathrm{at}(P)$, and $\tau \in 2^X$. Let $r' \in P_\tau$. It follows from Definition 3.2 that r' is obtained from some $r \in P$ by deleting $a, \neg a$ for all $a \in X$ from the head and body of r. Consequently $r' \in P - X$. Hence $P_\tau \subseteq P - X$ which establishes the proposition. □

3.3 Backdoor Evaluation

An analogue to Observation 3.1 does not hold for answer set programming, even if we consider the most basic problem Consistency. Take for example the program $P = \{\, x \leftarrow y;\ y \leftarrow x;\ \leftarrow \neg x;\ z \leftarrow \neg x \,\}$ and the set $X = \{x\}$. Both reducts $P_{x=0} = \{\, z \,\}$ and $P_{x=1} = \{\, y \,\}$ have answer sets, but P has no answer set. However, we can show a somewhat weaker asymmetric variant of Observation 3.1, where we can map each answer set of P to an answer set of P_τ for some $\tau \in 2^X$. This is made precise by the following definition and lemma (which are key for a backdoor approach to answer set programming).

Definition 3.8. *Let P be a program and X a set of atoms. We define*

$$\mathrm{AS}(P, X) = \{\, M \cup \tau^{-1}(1) : \tau \in 2^{X \cap \mathrm{at}(P)}, M \in \mathrm{AS}(P_\tau) \,\}.$$

In other words, the sets in $\mathrm{AS}(P, X)$ are answer sets of P_τ for truth assignments τ to $X \cap \mathrm{at}(P)$ extended by those atoms which are set to true by τ. In the following lemma we will see that the elements in $\mathrm{AS}(P, X)$ are "answer set candidates" of the original program P.

Lemma 3.9. $\mathrm{AS}(P) \subseteq \mathrm{AS}(P, X)$ *holds for every program P and every set X of atoms.*

Proof. Let $M \in \mathrm{AS}(P)$ be chosen arbitrarily. We put $X_0 = (X \setminus M) \cap \mathrm{at}(P)$ and $X_1 = X \cap M$ and define a truth assignment $\tau \in 2^{X \cap \mathrm{at}(P)}$ by setting $\tau^{-1}(i) = X_i$ for $i \in \{0, 1\}$. Let $M' = M \setminus X_1$. Observe that $M' \in \mathrm{AS}(P_\tau)$ implies $M \in \mathrm{AS}(P, X)$ since $M = M' \cup \tau^{-1}(1)$ by definition. Hence, to establish the lemma, it suffices to show that $M' \in \mathrm{AS}(P_\tau)$. We have to show that M' is a model of $P_\tau^{M'}$, and that no proper subset of M' is a model of $P_\tau^{M'}$.

In order to show that M' is a model of $P_\tau^{M'}$, choose $r' \in P_\tau^{M'}$ arbitrarily. By construction of $P_\tau^{M'}$ there is a corresponding rule $r \in P$ with $H(r') = H(r) \setminus X_0$ and $B^+(r') = B^+(r) \setminus X_1$ which gives rise to a rule $r'' \in P_\tau$, and in turn, r'' gives rise to $r' \in \mathrm{P}_\tau^{M'}$. Since $B^-(r) \cap X_1 = \emptyset$ (otherwise r would have been deleted forming P_τ) and $B^-(r) \cap M' = \emptyset$ (otherwise r'' would have been deleted forming $P_\tau^{M'}$), it follows that $B^-(r) \cap M = \emptyset$. Thus, r gives rise to a rule $r^* \in P^M$ with $H(r) = H(r^*)$ and $B^+(r) = B^+(r^*)$. Since $M \in \mathrm{AS}(P)$, M satisfies r^*, i.e., $H(r) \cap M \neq \emptyset$ or $B^+(r) \setminus M \neq \emptyset$. However, $H(r) \cap M = H(r') \cap M'$ and $B^+(r) \setminus M = B^+(r') \setminus M'$; thus, M' satisfies r'. Since $r' \in P_\tau^{M'}$ was chosen arbitrarily, we conclude that M' is a model of $P_\tau^{M'}$.

In order to show that no proper subset of M' is a model of $P_\tau^{M'}$ choose arbitrarily a proper subset $N' \subsetneq M'$. Let $N = N' \cup X_1$. Since $M' = M \setminus X_1$ and $X_1 \subseteq M$ it follows that $N \subsetneq M$. Since M is a minimal model of P^M, N cannot be a model of P^M. Consequently, there must be a rule $r \in P$ such that $B^-(r) \cap M = \emptyset$ (i.e., r is not deleted by forming P^M), $B^+(r) \subseteq N$ and $H(r) \cap N = \emptyset$. However, since M satisfies P^M and since $B^+(r) \subseteq N \subseteq M$, $H(r) \cap M \neq \emptyset$. Thus, r is not a constraint. Moreover, since $H(r) \cap M \neq \emptyset$ and $M \cap X_0 = \emptyset$, it follows that $H(r) \setminus X_0 \neq \emptyset$. Thus, since $H(r) \cap X_1 = \emptyset$, $H(r) \setminus X \neq \emptyset$. We conclude that r is not deleted when forming P_τ and giving rise to a rule $r' \in P_\tau$, which in turn is not deleted when forming $P_\tau^{M'}$, giving rise to a rule r'', with $H(r'') = H(r) \setminus X_0$, $B^+(r'') = B^+(r) \setminus X_1$, and $B^-(r'') = \emptyset$. Since $B^+(r'') \subseteq N'$ and $H(r'') \cap N = \emptyset$, N' is not a model of $P_\tau^{M'}$.

Thus, we have established that M' is a stable model of P_τ, and so the lemma follows. □

In view of Lemma 3.9 we shall refer to the elements in $\mathrm{AS}(P, X)$ as "answer set candidates."

Example 3.3. We consider program P of Example 2.1 and the strong **Horn**-backdoor $X = \{b, c\}$ of Example 3.1. The answer sets of P_τ are $\mathrm{AS}(P_{\neg b \neg c}) = \{\{e, f\}\}$, $\mathrm{AS}(P_{\neg bc}) = \{\{f\}\}$, $\mathrm{AS}(P_{b \neg c}) = \{\{e, f\}\}$, and $\mathrm{AS}(P_{bc}) = \{\{f\}\}$ for $\tau \in 2^{\{b,c\}}$. We obtain the set $\mathrm{AS}(P, X) = \{\{e, f\}, \{c, f\}, \{b, e, f\}, \{b, c, f\}\}$. ⊣

In view of Lemma 3.9, we can compute $\mathrm{AS}(P)$ by (i) computing $\mathrm{AS}(P_\tau)$ for all $\tau \in 2^X$ (this produces the set $\mathrm{AS}(P, X)$ of candidates for $\mathrm{AS}(P)$), and (ii) checking for each $M \in \mathrm{AS}(P, X)$ whether it is an answer set of P. The check (ii) entails (ii.a) checking whether $M \in \mathrm{AS}(P, X)$ is a model of P and (ii.b) whether $M \in \mathrm{AS}(P, X)$ is a minimal model of P^M. We would like to note that in particular any constraint contained in P

is removed in the truth assignment reduct P_τ but considered in check (ii.a). Clearly check (ii.a) can be carried out in polynomial time for each M. Check (ii.b), however, is co-NP-complete in general [MT91a], but polynomial for normal programs [CL94].

Fortunately, for our considerations it suffices to perform check (ii.b) for programs that are "close to **Normal**" (or close to a subset), and so the check is fixed-parameter tractable in the size of the given backdoor. More precisely, we consider the following parameterized problem and establish its fixed-parameter tractability in the next lemma.

Strong $\mathcal{C}$-Backdoor Asp Check

Given:	A program P, a strong $\mathcal{C}$-backdoor X of P and a set $M \subseteq \mathrm{at}(P)$.
Parameter:	The size $\lvert X\rvert$ of the backdoor.
Task:	Decide whether M is an answer set of P.

By deciding the status of the atoms in the backdoor, we can reduce a given program to *several* tractable programs belonging to a *target class* of programs. Consequently, the evaluation of the given program is polynomial for fixed backdoor size k where the order of the polynomial is independent of k, i.e., *fixed-parameter tractable* in the size of the backdoor [DF13]. By allowing backdoors of increasing size $k = 1, 2, 3, \ldots$, we can gradually augment a known tractable class of programs.

Lemma 3.10. *Let $\mathcal{C}$ be a class of normal programs. The problem* Strong $\mathcal{C}$-Backdoor Asp Check *is fixed-parameter tractable. More specifically, given a program P of input size n, a strong $\mathcal{C}$-backdoor X of P of size k, and a set $M \subseteq \mathrm{at}(P)$ of atoms, we can check in time $\mathcal{O}(2^k n)$ whether M is an answer set of P.*

Proof. Let $\mathcal{C}$ be a class of normal programs, P a program, and X a strong $\mathcal{C}$-backdoor X of P with $\lvert X\rvert = k$. We can check in polynomial time whether M is a model of P and whether M is a model of P^M. If it is not, we can reject M, and we are done. Hence assume that M is a model of P^M. In order to check whether $M \in \mathrm{AS}(P)$ we still need to decide whether M is a minimal model of P^M. Recall that P contains no tautological rules.

Let $X_1 \subseteq M \cap X$. We construct from P^M a program $P^M_{X_1 \subseteq X}$ by (i) removing all rules r for which $H(r) \cap X_1 \neq \emptyset$, and (ii) replacing for all remaining rules r the head $H(r)$ with $H(r) \setminus X$, and the positive body $B^+(r)$ with $B^+(r) \setminus X_1$.

Claim: $P^M_{X_1 \subseteq X}$ is Horn.

To show the claim, consider some rule $r' \in P^M_{X_1 \subseteq X}$. By construction, there must be a rule $r \in P$ that gives raise to a rule in P^M, which in turn gives raise to r'. Let $\tau \in 2^X$ be the assignment that sets all atoms in $X \cap H(r)$ to 0, and all atoms in $X \setminus H(r)$ to 1. Since r is not tautological, it follows that r is not deleted when we obtain P_τ, and it gives rise to a rule $r^* \in P_\tau$, where $H(r^*) = H(r) \setminus X$. However, since $\mathcal{C}$ is a class of normal programs, r^* is normal. Hence $1 \geq \lvert H(r^*)\rvert = \lvert H(r) \setminus X\rvert = \lvert H(r')\rvert$, and the claim follows.

To test whether M is a minimal model of P^M, we run the following procedure for every set $X_1 \subseteq M \cap X$.

> If $P^M_{X_1 \subseteq X}$ has no model, then stop and return *True*.
>
> Otherwise, compute the unique minimal model L of the Horn program $P^M_{X_1 \subseteq X}$.
> If $L \subseteq M \setminus X$, $L \cup X_1 \subsetneq M$, and $L \cup X_1$ is a model of P^M, then return *False*.
> Otherwise return *True*.

For each set $X_1 \subseteq M \cap X$ the above procedure runs in linear time by Lemma 2.1. As there are $\mathcal{O}(2^k)$ sets X_1 to consider, we have a total running time of $\mathcal{O}(2^k n)$ where n denotes the input size of P and $k = |X|$. It remains to establish the correctness of the above procedure in terms of the following claim.

Claim: M is a minimal model of P^M if and only if the algorithm returns True for each $X_1 \subseteq M \cap X$.

($\Rightarrow$). Assume that M is a minimal model of P^M, and suppose to the contrary that there is some $X_1 \subseteq M \cap X$ for which the algorithm returns *False*. Consequently, $P^M_{X_1 \subseteq X}$ has a unique minimal model L with $L \subseteq M \setminus X$, $L \cup X_1 \subsetneq M$, and where $L \cup X_1$ is a model of P^M. This contradicts the assumption that M is a minimal model of P^M. Hence the only-if direction of the claim is shown.

($\Leftarrow$). Assume that the algorithm returns *True* for each $X_1 \subseteq M \cap X$. We show that M is a minimal model of P^M. Suppose to the contrary that P^M has a model $M' \subsetneq M$.

We run the algorithm for $X_1 := M' \cap X$. By assumption, the algorithm returns *True*. There are two possibilities: (i) $P^M_{X_1 \subseteq X}$ has no model, or (ii) $P^M_{X_1 \subseteq X}$ has a model, and for its unique minimal model L the following holds: (iia) L is not a subset of $M \setminus X$, or (iib) $L \cup X_1$ is not a proper subset of M, or (iic) $L \cup X_1$ is not a model of P^M.

We show that case (i) is not possible by showing that $M' \setminus X$ is a model of $P^M_{X_1 \subseteq X}$.

To see this, consider a rule $r' \in P^M_{X_1 \subseteq X}$, and let $r \in P^M$ such that r' is obtained from r by removing X from $H(r)$ and by removing X_1 from $B^+(r)$. Since M' is a model of P^M, we have (a) $B^+(r) \setminus M' \neq \emptyset$ or (b) $H(r) \cap M' \neq \emptyset$. Moreover, since $B^+(r') = B^+(r) \setminus X_1$ and $X_1 = M' \cap X$, (a) implies $\emptyset \neq B^+(r) \setminus M' = B^+(r) \setminus X_1 \setminus M' = B^+(r') \setminus M' \subseteq B^+(r') \setminus (M' \setminus X)$, and since $H(r) \cap X_1 = \emptyset$, (b) implies $\emptyset \neq H(r) \cap M' = H(r) \cap (M' \setminus X_1) = H(r) \cap (M' \setminus X) = (H(r) \setminus X) \cap (M' \setminus X) = H(r') \cap (M' \setminus X)$. Hence $M' \setminus X$ satisfies r'. Since $r' \in P^M_{X_1 \subseteq X}$ was chosen arbitrarily, we conclude that $M' \setminus X$ is a model of $P^M_{X_1 \subseteq X}$.

Case (ii) is not possible either, as we can see as follows. Assume $P^M_{X_1 \subseteq X}$ has a model, and let L be its unique minimal model. Since $M' \setminus X$ is a model of $P^M_{X_1 \subseteq X}$, as shown above, we have $L \subseteq M' \setminus X$. Case (iia): We have $L \subseteq M \setminus X$ since $L \subseteq M' \setminus X$ and $M' \setminus X \subseteq M \setminus X$. Case (iib): Further we have $L \cup X_1 \subsetneq M$ since $L \cup X_1 \subseteq (M' \setminus X) \cup X_1 = (M' \setminus X) \cup (M' \cap X) = M' \subsetneq M$. Case (iic): And finally $L \cup X_1$ is a model of P^M,

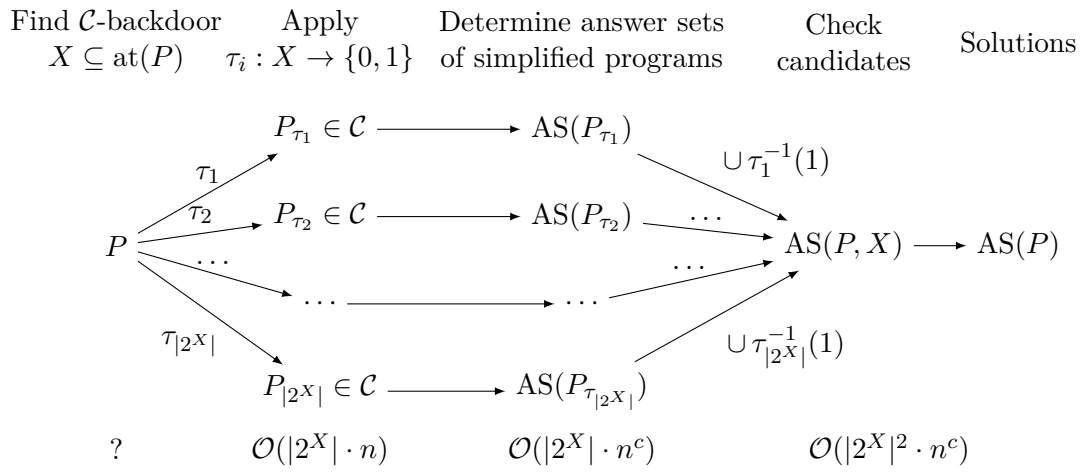


Figure 3.1: Exploit pattern of Asp backdoors if the target class $\mathcal{C}$ is normal and enumerable where n denotes the input size of P.

as can be seen as follows. Consider a rule $r \in P^M$. If $X_1 \cap H(r) \neq \emptyset$, then $L \cup X_1$ satisfies r; thus, it remains to consider the case $X_1 \cap H(r) = \emptyset$. In this case there is a rule $r' \in P^M_{X_1 \subseteq X}$ with $H(r') = H(r) \setminus X$ and $B^+(r') = B^+(r) \setminus X_1$. Since L is a model of $P^M_{X_1 \subseteq X}$, L satisfies r'. Hence (a) $B^+(r') \setminus L \neq \emptyset$ or (b) $H(r') \cap L \neq \emptyset$. Since $B^+(r') = B^+(r) \setminus X_1$, (a) implies that $B^+(r) \setminus (L \cup X_1) \neq \emptyset$; and since $H(r') \subseteq H(r)$, (b) implies that $H(r) \cap (L \cup X_1) \neq \emptyset$. Thus, $L \cup X_1$ satisfies r. Since $r \in P^M$ was chosen arbitrarily, we conclude that $L \cup X_1$ is a model of P^M.

Since neither Case (i) nor Case (ii) is possible, we have a contradiction, and we conclude that M is a minimal model of P^M. Hence the second direction of the claim is established.

In order to complete the proof, it remains to bound the running time. The check whether M is a model of P^M can clearly be carried out in linear time. For each set $X_1 \subseteq M \cap X$ the algorithm runs in linear time. This follows directly from the fact that we can compute the least model of a Horn program in linear time [DG84]. As there are at most 2^k sets X_1 to consider, the total running time is $\mathcal{O}(2^k \|P\|)$. Thus, in particular, the decision is fixed-parameter tractable for parameter k. □

Figure 3.1 illustrates how we can exploit a strong $\mathcal{C}$-backdoor to find answer sets. For a given program P and a strong $\mathcal{C}$-backdoor X of P we have to consider $|2^X|$ truth assignments to the atoms in the backdoor X. For each truth assignment $\tau \in 2^X$ we reduce the program P to a program P_τ and compute the set $\mathrm{AS}(P_\tau)$. Finally, we obtain the set $\mathrm{AS}(P)$ by checking for each $M \in \mathrm{AS}(P_\tau)$ whether it gives rise to an answer set of P.

Example 3.4. Consider the set $\mathrm{AS}(P, X) = \{\{e, f\}, \{c, f\}, \{b, e, f\}, \{b, c, f\}\}$ of answer set candidates from Example 3.3 and check for each candidate $L = \{e, f\}$, $M = \{c, f\}$, $N = \{b, e, f\}$, and $O = \{b, c, f\}$ whether it is an answer set of P. Therefore we solve the problem Strong **Horn**-Backdoor Asp Check by means of Lemma 3.10.

First we test whether the sets L, M, N and O are models of P. We easily observe that N and O are models of P. But L and M are not models of P since they do not satisfy the rule $c \leftarrow e, f, \neg b$ and $b \leftarrow c$ respectively, and we can drop them as candidates. Then we positively answer the question whether N and O are models of its GL reducts P^N and P^O, respectively.

Next we consider the minimality and apply the algorithm of the proof of Lemma 3.10 for each subset of the backdoor $X = \{b, c\}$. We have the GL reduct $P^N = \{d \leftarrow a, e;\ e \vee c \leftarrow f;\ f \leftarrow d, c;\ c \leftarrow d;\ b \leftarrow c;\ f\}$. For $X_1 = \emptyset$ we obtain $P^N_{X_1 \subseteq X} = \{d \leftarrow a, e;\ e \leftarrow f;\ f \leftarrow d, c;\ \leftarrow d;\ \leftarrow c;\ f\}$. The set $L = \{e, f\}$ is the unique minimal model of $P^N_{X_1 \subseteq X}$. Since $L \subseteq N \setminus X$, $L \cup X_1 \subsetneq N$, and $L \cup X_1$ is a model of P^N, the algorithm returns *False*. We conclude that N is not a minimal model of P^N and thus N is not an answer set of P.

We obtain the GL reduct $P^O = \{d \leftarrow a, e;\ e \vee c \leftarrow f;\ f \leftarrow d, c;\ c \leftarrow d;\ b \leftarrow c;\ f\}$. For $X_1 = \emptyset$ we have $P^O_{X_1 \subseteq X} = \{d \leftarrow a, e;\ e \leftarrow f;\ f \leftarrow d, e;\ \leftarrow d;\ \leftarrow c;\ f\}$. The set $L = \{e, f\}$ is the unique minimal model of $P^O_{X_1 \subseteq X}$. Since $L \cup X_1 \subsetneq O$, the algorithm returns *True*. For $X_2 = \{b\}$ we get $P^O_{X_2 \subseteq X} = \{d \leftarrow a, e;\ e \leftarrow f;\ f \leftarrow d, e;\ \leftarrow d;\ f\}$ and the unique minimal model $L = \{e, f\}$. Since $L \subseteq O \setminus X$, the algorithm returns *True*. For $X_3 = \{c\}$ we obtain $P^O_{X_3 \subseteq X} = \{d \leftarrow a, e;\ f \leftarrow d;\ \leftarrow;\ f\}$ and no minimal model. Thus, the algorithm returns *True*. For $X_4 = \{b, c\}$ we have $P^O_{X_4 \subseteq X} = \{d \leftarrow a, e;\ f \leftarrow d;\ f\}$ and the unique minimal model $L = \{f\}$. Since $L \cup X_1 \subsetneq O$, the algorithm returns *True*. Since only $\{b, c, f\} \in \mathrm{AS}(P, X)$ is an answer set of P, we obtain $\mathrm{AS}(P) = \{\{b, c, f\}\}$. ⊣

In view of Lemmas 3.9 and 3.10, the computation of $\mathrm{AS}(P)$ is fixed-parameter tractable for parameter k if we know a strong $\mathcal{C}$-backdoor X of size at most k for P, and each program in $\mathcal{C}$ is normal and its stable sets can be computed in polynomial time. This consideration leads to the following definition and result.

Definition 3.11. *A class $\mathcal{C}$ of programs is* enumerable *if for each $P \in \mathcal{C}$ we can compute $\mathrm{AS}(P)$ in polynomial time. If $\mathrm{AS}(P)$ can be computed in linear time, then the class $\mathcal{C}$ is* linear-time enumerable.

We would like to note that this is a stronger property than being enumerable with polynomial-time delay; the latter is usually used in the context of enumeration problems and also mentioned in Section 8.5 for a certain parameter.

Theorem 3.12. *Let $\mathcal{C}$ be an enumerable class of normal programs. The problems in AspFull are all fixed-parameter tractable when parameterized by the size of a strong $\mathcal{C}$-backdoor, assuming that the backdoor is given as input.*

Proof. Let X be the given backdoor, $k = |X|$ and n the input size of P. Since $P_\tau \in \mathcal{C}$ and $\mathcal{C}$ is enumerable, we can compute $\mathrm{AS}(P_\tau)$ in polynomial time for each $\tau \in 2^X$, say in time $\mathcal{O}(n^c)$ for some constant c. Observe that therefore $|\mathrm{AS}(P_\tau)| \leq \mathcal{O}(n^c)$ for each $\tau \in 2^X$. Thus, we obtain $\mathrm{AS}(P, X)$ in time $\mathcal{O}(2^k n^c)$, and $|\mathrm{AS}(P, X)| \leq \mathcal{O}(2^k n^c)$. By Lemma 3.9, $\mathrm{AS}(P) \subseteq \mathrm{AS}(P, X)$. By means of Lemma 3.10 we can decide whether $M \in AS(P)$ in time $\mathcal{O}(2^k n)$ for each $M \in \mathrm{AS}(P, X)$. Thus, we determine from $\mathrm{AS}(P, X)$ the set of all answer sets of P in time $\mathcal{O}(2^k \cdot n^c \cdot 2^k \cdot n + 2^k \cdot n^c) = \mathcal{O}(2^{2k} n^{c+1})$. Once we know $\mathrm{AS}(P)$, then we can also solve all problems in $\mathcal{AspFull}$ within polynomial time. □

Theorem 3.12 identifies conditions under which a small backdoor indeed reduces the search space for the main Asp reasoning problems, that is, to be exponential only in the backdoor size and not in the size of the entire instance. Hence under these conditions a small backdoor can be considered as a "clever reasoning shortcut" through the search space.

Remark. If we know that each program in $\mathcal{C}$ has at most one answer set, and P has a strong $\mathcal{C}$-backdoor of size k, then we can conclude that P has at most 2^k answer sets. Thus, we obtain an upper bound on the number of answer sets of P by computing a small strong $\mathcal{C}$-backdoor of P.

3.4 Backdoor Detection

Theorem 3.12 draws our attention to enumerable classes of normal programs. Given such a class $\mathcal{C}$, is the detection of $\mathcal{C}$-backdoors fixed-parameter tractable? If the answer is affirmative, we can drop in Theorem 3.12 the assumption that the backdoor is given as an input for this class.

Each class $\mathcal{C}$ of programs gives rise to the following two parameterized decision problems:

Strong $\mathcal{C}$-Backdoor Detection

Given:	A program P and an integer k.
Parameter:	The integer k.
Task:	Decide whether P has a strong $\mathcal{C}$-backdoor X of size at most k.

Deletion $\mathcal{C}$-Backdoor Detection

Given:	A program P and an integer k.
Parameter:	The integer k.
Task:	Decide whether P has a deletion $\mathcal{C}$-backdoor X of size at most k.

By a standard construction, known as self-reduction or self-transformation [Sch81; DF99; DF13], one can use a decision algorithm for Strong (Deletion) $\mathcal{C}$-Backdoor Detection to actually find the backdoor. We only require the target class to be hereditary.

Lemma 3.13. *Let $\mathcal{C}$ be a hereditary class of programs. If* Strong (Deletion) $\mathcal{C}$-Backdoor Detection *is fixed-parameter tractable, then also finding a strong (deletion) $\mathcal{C}$-backdoor of a given program P of size at most k is fixed-parameter tractable (for parameter k).*

Proof. We proceed by induction on k. If $k = 0$ the statement is clearly true. Let $k > 0$. Given (P, k) we check for all $a \in \mathrm{at}(P)$ whether P_τ and $P_{\tau'}$ have a strong $\mathcal{C}$-backdoor of size at most $k-1$ where $\tau(a) = 1$ and $\tau'(a) = 0$. If the answer is *No* for all a, then P has no strong $\mathcal{C}$-backdoor of size k. If the answer is *Yes* for a, then by induction hypothesis we can compute a strong $\mathcal{C}$-backdoor X of size at most $k-1$ of P_τ and $P_{\tau'}$, and $X \cup \{a\}$ is a strong $\mathcal{C}$-backdoor of P. Similarly, we proceed for deletion $\mathcal{C}$-backdoor. Given (P, k) we check for all $a \in \mathrm{at}(P)$ whether $P - \{a\}$ has a deletion $\mathcal{C}$-backdoor of size at most $k-1$. If the answer is *No* for all a, then P has no deletion $\mathcal{C}$-backdoor of size k. If the answer is *Yes* for a, then by induction hypothesis we can compute a deletion $\mathcal{C}$-backdoor X of size at most $k-1$ of $P - \{a\}$, and $X \cup \{a\}$ is a deletion $\mathcal{C}$-backdoor of P. □

Remark. One could consider also target classes where empty rules are detected (an analogy to "empty clause detection" [DGS07; DGS14] in the Sat setting) which would yield smaller backdoors. However, backdoor detection is already W[1]-hard for almost all base classes, including Horn, in the Sat setting when empty clause detection is added [Sze08]. These W[1]-hardness results carry over to the Asp setting if empty rule detection is added.

We will investigate in detail on the computational complexity of the backdoor detection problem for various target classes in the following Chapter 4.

Background and Related Work

Backdoors were originally introduced by Williams, Gomes, and Selman [WGS03a; WGS03b] as a tool to analyze the behavior of DPLL-based Sat solvers. Since then, backdoors have been frequently used in the literature for theoretical investigations [RKH04; SS09b; KKS08]. The study of the parameterized complexity of backdoor detection was initiated by Nishimura, Ragde, and Szeider [NRS04] who considered satisfiability backdoors for the target classes Horn and 2CNF. Since then, the study has been extended to various other target classes, including clustering formulas [NRS07], renamable Horn formulas [RO09], QHorn formulas [Gas+13], Nested formulas [GS12c], acyclic formulas [GS12a], and formulas of bounded incidence treewidth [GS13]; we also refer to a recent survey [GS12b]. Moreover, it has been empirically determined that backdoors can be small, see e.g., [LB11; Gar12; WGS03a; DGS09].

Several results extend the concept of backdoors to other problems, e.g., backdoors for constraint satisfaction problems [WGS03a], quantified Boolean formulas [SS09a], abstract argumentation [DOS12], and abductive reasoning [PRS13]. Samer and Szeider [SS08] have introduced *backdoor trees* for propositional satisfiability which provide a more refined

concept of backdoor evaluation and take the interaction of variables that form a backdoor into account.

Dilkina, Gomes, and Sabharwal [DGS07; DGS14] have considered strong backdoors with *"empty clause detection"* (empty clauses trivially yield satisfiability). Empty clause detection is present in many modern Sat solvers and often leads to much smaller backdoors in practice. However, Dilkina, Gomes, and Sabharwal have also established that backdoor detection for the target classes Horn and 2CNF is already harder than NP when empty clause detection is added. Moreover, Szeider [Sze08] has shown that (strong) backdoor detection is W[1]-hard for almost all target class when empty clause detection is added and thus unlikely to be fixed-parameter tractable.

Contribution and Discussion

In this chapter we have introduced backdoors to the domain of answer set programming. Similar to truth reducts of formulas we have defined truth assignment reducts of disjunctive answer set programs which in particular generalize a concept of Gottlob, Scarcello, and Sideri [GSS02] (see Section 8.4). Moreover, we have introduced strong and deletion backdoors for disjunctive answer set programs.

When we exploit backdoors to solve propositional satisfiability or constraint satisfaction we have a two step approach which consists of (i) backdoor detection and (ii) backdoor evaluation. Similarly, we have a two step approach to solve problems in $\mathcal{A}sp\mathcal{F}ull$ (consistency, brave and skeptical reasoning, answer set counting and enumeration of all answer sets) by means of backdoors; again (i) backdoor detection and (ii) evaluating the backdoor for the respective reasoning task.

In this chapter we have specified the problems strong and deletion backdoor detection which are quite similar to Sat backdoors. We are interested in backdoors into a target class where the considered problem is tractable and backdoors can be found easily. Since in many cases we cannot find a strong backdoor efficiently, one relaxes the notion of strong backdoors to deletion backdoors. Therefore, a target class needs a certain property which ensures that deletion backdoors are also strong backdoors. In propositional satisfiability this property is clause induced and similar we have the property hereditary in answer set programming.

While Sat target classes are usually polynomial-time tractable, target classes for answer set programming backdoors have to be enumerable for fixed-parameter-tractability of backdoor evaluation. However, it turns out that backdoor evaluation is much more sophisticated for answer set programming. This seems to be not too surprising from the perspective of computational complexity as the propositional satisfiability problem is NP-complete whereas the problems in $\mathcal{A}sp\mathcal{R}eason$ for disjunctive programs are located on the second level of the Polynomial Hierarchy. In particular, for propositional satisfiability or constraint satisfaction we require the solution (if it exists) to be a model of the formula whereas answer sets have in addition to be minimal with respect to the GF reduct. The

minimality check is co-NP-complete in general. However, we establish in Lemma 3.10 the new result that the minimality check is fixed parameter tractable when parameterized by the size of a strong backdoor into a subclass of normal programs. The main and new result of this chapter is Theorem 3.12 which states that the problems in $\mathcal{AspFull}$ are fixed-parameter tractable when parameterized by the size of a given strong backdoor.

CHAPTER 4

Tractability Backdoors

In this chapter we study the complexity of backdoor detection when parameterized by the size of a backdoor into some fixed target class. Before we can use a backdoor to solve reasoning problems of answer set programming, we have to find it first, which is mostly computationally hard. We usually consider small backdoors whereas the instance itself might be large, since the backdoor approach is only reasonable on instances that allow small backdoors. We restrict the considered target classes to enumerable target classes, where we can compute for each program in the class all answer sets in polynomial time. When we can find such a backdoor significantly faster than by brute force search, in other words backdoor detection is fixed-parameter tractable, the target class is suitable for the backdoor approach (according to Theorem 3.12).

In Section 4.1 we consider the target class **Horn**. We establish that deletion **Horn**-backdoors and strong **Horn**-backdoors coincide. We provide an approach to detect such backdoors by means of a reduction to the vertex cover problem of a certain graph representation of the given program. In Section 4.2 we consider acyclicity-based target classes among them stratified programs and beyond which we define by the absence of certain types of cycles in graph representations of the given program, more precisely no cycles (**no-C**), no directed cycles (**no-DC**), no bad cycles (**no-BC**), no even cycles (**no-EC**), and their combinations. We show that detecting strong backdoors into the acyclicity-based target classes is unlikely to be fixed-parameter tractable. Figure 4.4 (see p. 45) illustrates the established complexity results. We show by means of reductions to certain feedback vertex set problems that detecting deletion backdoors into most of the acyclicity-based target classes is fixed-parameter tractable. For the remaining target classes (based on directed even acyclicity) we provide hardness results for deletion backdoor detection. Figure 4.5 (see p. 46) illustrates the established complexity results. Finally, we provide background and related work, a summary of our contribution, and a discussion of the results. This chapter is based on published work [FS15b].

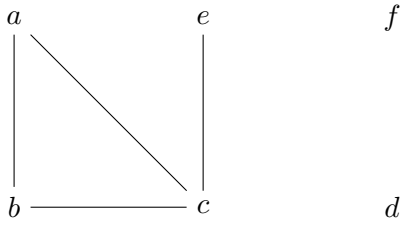


Figure 4.1: Negation dependency graph N_p of the program P from Example 2.1.

4.1 Target Class Horn

In this section we consider the important case **Horn** as the target class for backdoors. As a consequence of Lemma 2.1, **Horn** is linear-time enumerable. The following lemma shows that strong and deletion **Horn**-backdoors coincide.

Lemma 4.1. *A set X is a strong* **Horn***-backdoor of a program P if and only it is a deletion* **Horn***-backdoor of P.*

Proof. Since **Horn** is hereditary, Lemma 3.7 establishes the if-direction. For the only-if direction, we assume for the sake of a contradiction that X is a strong **Horn**-backdoor of P but not a deletion **Horn**-backdoor of P. Hence there is a rule $r' \in P - X$ that is neither tautological nor Horn. Let $r \in P$ be a rule from which r' was obtained in forming $P - X$. We define $\tau \in 2^X$ by setting all atoms in $X \cap (H(r) \cup B^-(r))$ to 0, all atoms in $X \cap B^+(r)$ to 1, and all remaining atoms in $X \setminus \mathrm{at}(r)$ arbitrarily to 0 or 1. Since r is not tautological, this definition of τ is sound. It follows that $r' \in P_\tau$, contradicting the assumption that X is a strong **Horn**-backdoor of P. □

Remark. Recall that we assume that programs do not contain any tautological rules (see Section 3.2). Consider the program $a \vee b \leftarrow a, \neg c$; the set $X = \{a\}$ is a strong **Horn**-backdoor, however X not a deletion **Horn**-backdoor. Hence, it is important that tautologies are excluded for Lemma 4.1.

Definition 4.2. *Let P be a program. The* negation dependency graph N_p *is the graph defined on the set of atoms of the given program P, where two distinct atoms x, y are joined by an edge xy if there is a rule $r \in P$ with $x \in H(r)$ and $y \in H(r) \cup B^-(r)$.*

Example 4.1. Figure 4.1 visualizes the negation dependency graph N_p of program P from Example 2.1. ⊣

The following lemma states how we can use recent results on the vertex cover problem to find deletion backdoors for the target class **Horn**. A *vertex cover* of a graph $G = (V, E)$ is a set $S \subseteq V$ such that for every edge $uv \in E$ we have $\{u, v\} \cap S \neq \emptyset$.

Lemma 4.3. *Let P be a program. A set $X \subseteq \mathrm{at}(P)$ is a deletion* **Horn***-backdoor of P if and only if X is a vertex cover of the negation dependency graph N_p.*

Proof. Let $X \subseteq \mathrm{at}(P)$ be a deletion **Horn**-backdoor of P. Consider an edge uv of N_p. By construction of N_p there is a corresponding rule $r \in P$ with (i) $u, v \in H(r)$ and $u \neq v$ or (ii) $u \in H(r)$ and $v \in B^-(r)$. Since X is a deletion **Horn**-backdoor, $|H(r) - X| \leq 1$ and $B^-(r) - X = \emptyset$. Thus, if Case (i) applies, $\{u, v\} \cap X \neq \emptyset$. If Case (ii) applies, again $\{u, v\} \cap X \neq \emptyset$. We conclude that X is a vertex cover of N_p.

Conversely, assume that X is a vertex cover of N_p. Consider a rule $r \in P - X$ for proof by contradiction assume that r is not Horn (in particular r is not a constraint). If $|H(r)| \geq 2$ then there are two variables $u, v \in H(r)$ and an edge uv of N_p such that $\{u, v\} \cap X = \emptyset$, contradicting the assumption that X is a vertex cover. Similarly, if $B^-(r) \neq \emptyset$ then we take a variable $u \in B^-(r)$ and a variable $v \in H(r)$; such v exists since r is not a constraint. Thus, N_p contains the edge uv with $\{u, v\} \cap X = \emptyset$, contradicting the assumption that X is a vertex cover. Hence the claim holds. □

Example 4.2. For instance, the negation dependency graph N_p of program P from Example 2.1 consists of the triangle $\{a, b, c\}$ and a path (c, e). Then $\{b, c\}$ is a vertex cover of N_p. We observe easily that there exists no vertex cover of size 1. Thus, $\{b, c\}$ is a smallest strong **Horn**-backdoor of P. ⊣

Remark. Note that the unparameterized version (see Section 2.4) of Strong **Horn**-Backdoor Detection is NP-hard. Since the reduction presented in the proof of Lemma 4.3 can be used straightforward as a reduction of the unparameterized version of the vertex cover problem to Strong **Horn**-Backdoor Detection.

Theorem 4.4. Strong **Horn**-Backdoor Detection *is fixed-parameter tractable. In fact, given a program with n atoms we can find a strong* **Horn***-backdoor of size at most k in time $\mathcal{O}(1.2738^k + kn)$ or decide that no such backdoor exists.*

Proof. Let P be a given program. Let N_p be the negation dependency graph of P. According to Lemma 4.3, a set $X \subseteq \mathrm{at}(P)$ is a vertex cover of N_p if and only if X is a deletion **Horn**-backdoor of P. Then a vertex cover of size at most k, if it exists, can be found in time $\mathcal{O}(1.2738^k + kn)$ by Chen, Kanj, and Xia [CKX10]. By Lemma 4.1 this vertex cover is also a strong **Horn**-backdoor of P. □

Now we can use Theorem 4.4 to strengthen the fixed-parameter tractability result of Theorem 3.12 by dropping the assumption that the backdoor is given.

Corollary 4.5. *All the problems in $\mathcal{AspFull}$ are fixed-parameter tractable when parameterized by the size of a smallest strong* **Horn***-backdoor of the given program.*

4.2 Target Classes Based on Acyclicity

There are two causes for a program to have a large number of answer sets: (i) disjunctions in the heads of rules, and (ii) certain cyclic dependencies between rules. Disallowing both yields enumerable classes.

In order to define acyclicity we associate with each disjunctive program P its *dependency digraph* D_p and its *(undirected) dependency graph* U_p. The dependency digraph was defined by Apt, Blair, and Walker [ABW88] which slight differs from our notion as no additional edges on head atoms are introduced. The following definition is closely related to the notion suggested by Gottlob, Scarcello, and Sideri [GSS02].

Definition 4.6. *Let P be a program. The* dependency digraph *is the digraph D_p which has as vertices the atoms of P and a directed edge (x, y) between any two distinct atoms x, y for which there is a rule $r \in P$ with $x \in H(r)$ and $y \in B^+(r) \cup B^-(r)$ or $x, y \in H(r)$. We call the edge (x, y)* negative *if there is a rule $r \in P$ with $x \in H(r)$ and $y \in B^-(r)$ or $x, y \in H(r)$.*

Definition 4.7. *Let P be a program. The* (undirected) dependency graph *is the graph U_p obtained from the dependency digraph D_p*

1. *by replacing each negative edge $e = (x, y)$ with two edges xv_e, $v_e y$ where v_e is a new* negative vertex*, and*
2. *by replacing each remaining directed edge (u, v) with an edge uv.*

Example 4.3. Figure 4.2 visualizes the dependency digraph D_p and the dependency graph U_p of program P from Example 2.1. ⊣

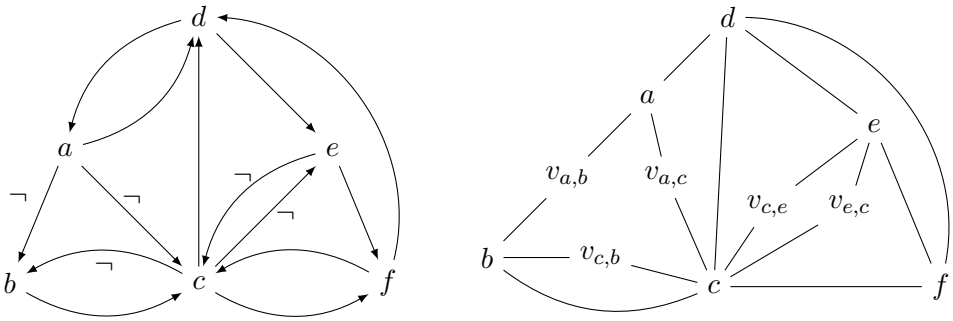


Figure 4.2: Dependency digraph D_p (left) and dependency graph U_p (right) of program P from Example 2.1.

Definition 4.8. *Let P be a program.*

1. *A* directed cycle of P *is a directed cycle in the dependency digraph D_p.*

2. *A directed cycle is* bad *if it contains a negative edge, otherwise it is* good.
3. *A directed cycle is* even *if it contains an even number of negative edges, otherwise it is* odd.
4. *A* cycle of P *is a cycle in the dependency graph* U_p.
5. *A cycle is* bad *if it contains a negative vertex, otherwise it is* good.
6. *A cycle is* even *if it contains an even number of negative vertices, otherwise it is* odd.

Definition 4.9. *The following classes of programs are defined in terms of the absence of certain kinds of cycles:*

- **no-C** *contains all programs that have no cycles,*
- **no-BC** *contains all programs that have no bad cycles,*
- **no-DC** *contains all programs that have no directed cycles,*
- **no-DC2** *contains all programs that have no directed cycles of length at least 3 and no directed bad cycles,*
- **no-DBC** *contains all programs that have no directed bad cycles,*
- **no-EC** *contains all programs that have no even cycles,*
- **no-BEC** *contains all programs that have no bad even cycles,*
- **no-DEC** *contains all programs that have no directed even cycles, and*
- **no-DBEC** *contains all programs that have no directed bad even cycles.*

We let $\mathcal{A}cyc$ *denote the family of all the nine classes defined above. We also write* $\mathcal{D}$-$\mathcal{A}cyc$ *to denote the subfamily* {**no-DC**, **no-DC2**, **no-DBC**, **no-DEC**, **no-DBEC**} $\subseteq \mathcal{A}cyc$.

Example 4.4. Consider the dependency graphs of program P from Example 2.1 as depicted in Figure 4.2. For instance the sequence (d, e, f) is a cycle, (d, a) is a directed cycle (of length 2), (d, e, f) and (c, e, f) are directed cycles (of length 3), $(a, v_{(a,c)}, c, d)$ is a bad cycle, (b, c) is a directed bad cycle. The sequence (d, e, f) is an even cycle and a directed even cycle, (c, e) is a directed bad even cycle.

The set $X = \{c\}$ is a strong **no-DBEC**-backdoor since the truth assignment reducts $P_{\neg c} = P_{c=0} = \{d \leftarrow a, e;\ a \leftarrow d, \neg b;\ e \leftarrow f;\ f\}$ and $P_c = P_{c=1} = \{d \leftarrow a, e;\ f \leftarrow d;\ b;\ f\}$ are in the target class **no-DBEC**. X is also a strong **no-BEC**-backdoor, since $P_{\neg c} \in$ **no-BEC** and $P_c \in$ **no-BEC**. The answer sets of P_τ are $\mathrm{AS}(P_{\neg c}) = \{\{e, f\}\}$ and $\mathrm{AS}(P_c) = \{\{b, f\}\}$. Thus, $\mathrm{AS}(P, X) = \{\{e, f\}, \{b, c, f\}\}$, and since only $\{b, c, f\}$ is an answer set of P, we obtain $\mathrm{AS}(P) = \{\{b, c, f\}\}$. ⊣

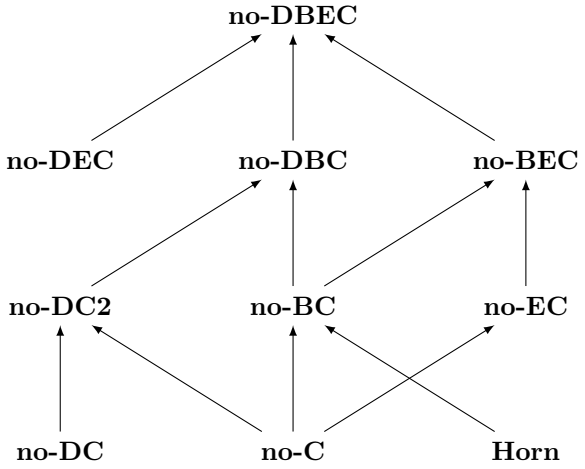


Figure 4.3: Relationship between classes of programs with respect to their generality. A directed path from a class $\mathcal{C}$ to a class $\mathcal{C}'$ indicates that $\mathcal{C} \subseteq \mathcal{C}'$. If there is no path between two classes $\mathcal{C}$ and $\mathcal{C}'$, then neither $\mathcal{C} \subseteq \mathcal{C}'$ nor $\mathcal{C}' \subseteq \mathcal{C}$ and we say $\mathcal{C}$ and $\mathcal{C}'$ are *incomparable*.

The dependency and dependency digraphs contain bad even cycles through head atoms for non-singleton heads. This has the following consequence.

Observation 4.10. $\mathcal{C} \subseteq$ **Normal** *holds for all* $\mathcal{C} \in \mathcal{A}cyc$.

If we have two programs $P \subseteq P'$, then clearly the dependency (di)graph of P is a sub(di)graph of the dependency (di)graph of P'. This has the following consequence.

Observation 4.11. *All* $\mathcal{C} \in \mathcal{A}cyc$ *are hereditary.*

The following is a direct consequence of the definitions of the various classes in $\mathcal{A}cyc$.

Observation 4.12. *If* $\mathcal{C}, \mathcal{C}' \in \mathcal{A}cyc \cup \{$**Horn**$\}$ *such that the digraph in Figure 4.3 contains a directed path from the class* $\mathcal{C}$ *to the class* $\mathcal{C}'$*, then* $\mathcal{C} \subseteq \mathcal{C}'$*. If no inclusion between two classes is indicated, then the classes are in fact incomparable.*

Proof. We first consider the acyclicity-based target classes. By definition we have **no-DC** $\subsetneq$ **no-DBC** and **no-C** $\subsetneq$ **no-BC** $\subsetneq$ **no-DBC**; it is easy to see that the inclusions are proper. However, contrary to what one expects, **no-C** $\not\subseteq$ **no-DC**, which can be seen by considering the program $P_1 = \{x \leftarrow y,\ y \leftarrow x\}$. But the class **no-DC2** which requires that a program has no directed cycles but may have directed good cycles of length 2 (as in P_1) generalizes both classes **no-C** and **no-DC**. By definition we have **no-DBC** $\subsetneq$ **no-DBEC**, **no-DEC** $\subsetneq$ **no-DBEC**, **no-EC** $\subsetneq$ **no-BEC**, **no-C** $\subsetneq$ **no-EC**, and **no-DC** $\subsetneq$ **no-DEC**.

Next we consider the target class **Horn**. Let $\mathcal{C} \in \{\textbf{no-C}, \textbf{no-DC}, \textbf{no-DC2}, \textbf{no-EC}\}$. We easily observe that **Horn** $\not\subseteq \mathcal{C}$ by considering the program $P_2 = \{a \leftarrow b;\ b \leftarrow c;\ c \leftarrow a\}$ which is obviously Horn but does not belong to $\mathcal{C}$. Conversely, we observe that $\mathcal{C} \not\subseteq$ **Horn** by considering the program $P_3 = \{a \leftarrow \neg b\}$ which belongs to $\mathcal{C}$ but is obviously not Horn. Thus, $\mathcal{C}$ and **Horn** are incomparable. We observe that **Horn** $\subsetneq$ **no-BC** by again considering the program P_3 which belongs to **no-BC**, but is obviously not Horn, and by considering the fact that all rules r in a Horn program P satisfy $|H(r)| \leq 1$ and $B^-(r) = \emptyset$ which yields that the dependency graph U_p contains no bad vertices and hence gives us that U_p contains no bad cycles.

□

The class **no-DBC** coincides with the well-known class of *stratified* programs [ABW88; Gel89; CH85]. A normal program P is *stratified* if there is a mapping $str : \mathrm{at}(P) \to \mathbb{N}$, called *stratification*, such that for each rule r in P the following holds: (i) if $x \in H(r)$ and $y \in B^+(r)$, then $str(x) \leq str(y)$ and (ii) if $x \in H(r)$ and $y \in B^-(r)$, then $str(x) < str(y)$.

Lemma 4.13 (Apt, Blair, and Walker, 1988)**. Strat = no-DBC**.

The class **no-DBEC**, the largest class in $\mathcal{A}cyc$, has already been studied by Zhao and Lin [Zha02; LZ04a], who showed that every program in **no-DBEC** has at most one answer set, and this answer set can be found in polynomial time. For **no-DBC** the unique answer set can even be found in linear time [NR94].

In our context this has the following important consequence.

Proposition 4.14. *All classes in $\mathcal{A}cyc$ are enumerable, the classes $\mathcal{C} \in \mathcal{A}cyc$ with $\mathcal{C} \subseteq$ **no-DBC** are even linear-time enumerable.*

In view of Observation 4.10 and Proposition 4.14, all classes in $\mathcal{A}cyc$ satisfy the requirement of Theorem 3.12 and are therefore in principle suitable target classes of a backdoor approach. Therefore, we will study the parameterized complexity of Strong $\mathcal{C}$-Backdoor Detection and Deletion $\mathcal{C}$-Backdoor Detection for $\mathcal{C} \in Acyc$. As we shall see in the next two subsections, the results for Strong $\mathcal{C}$-Backdoor Detection are throughout negative, however for Deletion $\mathcal{C}$-Backdoor Detection there are several (fixed-parameter) tractable cases.

4.2.1 Strong Backdoor Detection

Proposition 4.15. *Assume that the input program may contain tautological rules. Then, for every target class $\mathcal{C} \in \mathcal{A}cyc$, the problem* Strong $\mathcal{C}$-Backdoor Detection *is* W[2]*-hard, and hence unlikely to be fixed-parameter tractable.*

Proof. We give an fpt-reduction from the W[2]-complete problem Hitting Set (see Section 2.4) to Strong $\mathcal{C}$-Backdoor Detection. Let (S, k) be an instance of this

problem with $\mathsf{S} = \{S_1, \ldots, S_m\}$. We construct a program P as follows. As atoms we take the elements of $U = \bigcup_{i=1}^{m} S_i$ and new atoms a_i^j and b_i^j for $1 \leq i \leq m$, $1 \leq j \leq k+1$. For each $1 \leq i \leq m$ and $1 \leq j \leq k+1$ we take two rules r_i^j, s_i^j where $H(r_i^j) = \{a_i^j\}$, $B^-(r_i^j) = S_i \cup \{b_i^j\}$, $B^+(r_i^j) = S_i$ (which is a tautological rule); $H(s_i^j) = \{b_i^j\}$, $B^-(s_i^j) = \{a_i^j\}$, $B^+(s_i^j) = \emptyset$.

We show that S has a hitting set of size at most k if and only if P has a strong $\mathcal{C}$-backdoor of size at most k.

($\Rightarrow$). Let H an hitting set of S of size at most k. We choose an arbitrary truth assignment $\tau \in 2^H$ and show that $P_\tau \in C$. Since H is a hitting set, each rule r_i^j will be removed when forming P_τ. Hence the only rules left in P_τ are the rules s_i^j, and so $P_\tau \in \textbf{no-DC} \cap \textbf{no-C} \subseteq \mathcal{C}$. Thus, H is a strong $\mathcal{C}$-backdoor of P.

($\Leftarrow$). Let X be a strong $\mathcal{C}$-backdoor of P of size at most k. We show that $H = X \cap U$ is a hitting set of S. Choose $1 \leq i \leq m$ and consider S_i. We first consider the case $\textbf{no-DC} \subseteq \mathcal{C}$. For each $1 \leq j \leq k+1$ the program P contains a bad even directed cycle (a_i^j, b_i^j). In order to destroy these cycles, X must contain an atom from S_i, since otherwise, X would need to contain for each $1 \leq j \leq k+1$ at least one of the atoms from each cycle, but then $|X| \geq k+1$, contradicting the assumption on the size of X. Hence H is a hitting set of S. Now we consider the case $\textbf{no-C} \subseteq \mathcal{C}$. For each $1 \leq j \leq k+1$ the program P contains a bad even cycle $(a_i^j, v_{a_i^j,b_i^j}, b_i^j, v_{b_i^j,a_i^j})$. In order to destroy these cycles, X must contain an atom from S_i, since otherwise, X would need to contain an atom from each cycle, again a contradiction. Hence H is a hitting set of S. Consequently, the W[2]-hardness of Strong $\mathcal{C}$-Backdoor Detection follows. □

For the target classes in $\mathcal{D}$-*Acyc* we can avoid the use of tautological rules in the reduction and so strengthen Proposition 4.15 as follows (it would be interesting to know if this is also possible for the remaining classes mentioned in Proposition 4.15).

Theorem 4.16. *For every target class $\mathcal{C} \in \mathcal{D}$-Acyc, the problem* Strong $\mathcal{C}$-Backdoor Detection *is* W[2]*-hard, and hence unlikely to be fixed-parameter tractable.*

Proof. In order to show that Strong $\mathcal{C}$-Backdoor Detection is W[2]-hard for $\mathcal{C} \in \mathcal{D}$-*Acyc* when we forbid tautological rules in the input, we modify the reduction used in the proof of Proposition 4.15 from Hitting Set by redefining the rules r_i^j, s_i^j. We put $H(r_i^j) = \{a_i^j\}$, $B^-(r_i^j) = S_i \cup \{b_i^j\}$, $B^+(r_i^j) = \emptyset$; $H(s_i^j) = \{b_i^j\}$, $B^-(s_i^j) = \{a_i^j\}$, $B^+(s_i^j) = U$. By the very same argument as in the proof of Proposition 4.15 we can show that S has a hitting set of size at most k if and only if P has a strong $\mathcal{C}$-backdoor of size at most k. We would like to mention that this reduction does not work for the undirected cases as it yields undirected cycles $(b_i^j, u, b_{i'}^{j'}, u')$ for any $u, u' \in U$. □

For the class **no-DBEC** we can again strengthen the result and show that detecting a strong **no-DBEC**-backdoor is already co-NP-hard for backdoor size 0; hence the problem is co-para-NP-hard (see Section 2.4).

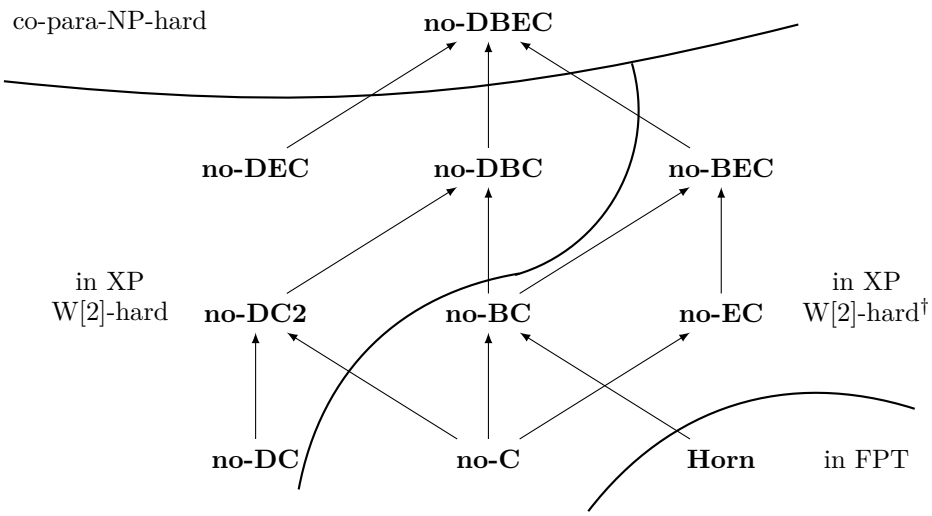


Figure 4.4: Known complexity of the problem Strong $\mathcal{C}$-Backdoor Detection. †: When we allow tautologies in the input program, see Theorem 4.16.

Theorem 4.17. *The problem* Strong **no-DBEC**-Backdoor Detection *is* co-para-NP*-hard, and hence not fixed-parameter tractable unless* P = co-NP*.*

Proof. Recall that a path does not visit the same vertex twice. We reduce from the following problem, which is NP-complete [FHW80; LP84],

Directed Path via a Node
Given: A digraph G and $s, m, t \in V$ distinct vertices.
Task: Decide whether G contains a directed path from s to t via m.

Let $G = (V, E)$ be a digraph and $s, m, t \in V$ distinct vertices. We define a program P as follows: For each edge $e = (v, w) \in E$ where $w \neq m$ we take a rule r_e: $w \leftarrow v$. For each edge $e = (v, m)$ we take a rule r_e: $m \leftarrow \neg v$. Finally we add the rule $r_{s,t}$: $s \leftarrow \neg t$. We observe that the dependency digraph of P is exactly the digraph we obtain from G by adding the "reverse" edge (t, s) (if not already present), and by marking (t, s) and all incoming edges of m as negative.

We show that G has a path from s to t via m if and only if $P \notin$ **no-DBEC**. Assume G has such a path. Then this path must contain exactly one incoming edge of m, and hence it contains exactly one negative edge. The path, together with the negative edge (t, s), forms a directed bad even cycle of P, hence $P \notin$ **no-DBEC**. Conversely, assume $P \notin$ **no-DBEC**. Hence the dependency digraph of P contains a directed bad even cycle, i.e., a cycle that contains at least two negative edges. As it can contain at most one incoming edge of m, the cycle contains exactly one incoming edge of m and the reverse edge (t, s). Consequently, the cycle induces in G a directed path from s to t via m. □

Figure 4.4 illustrates the known complexity results of the problem Strong $\mathcal{C}$-Backdoor Detection. An arrow from $\mathcal{C}$ to $\mathcal{C}'$ indicates that $\mathcal{C}'$ is a proper subset of $\mathcal{C}$ and

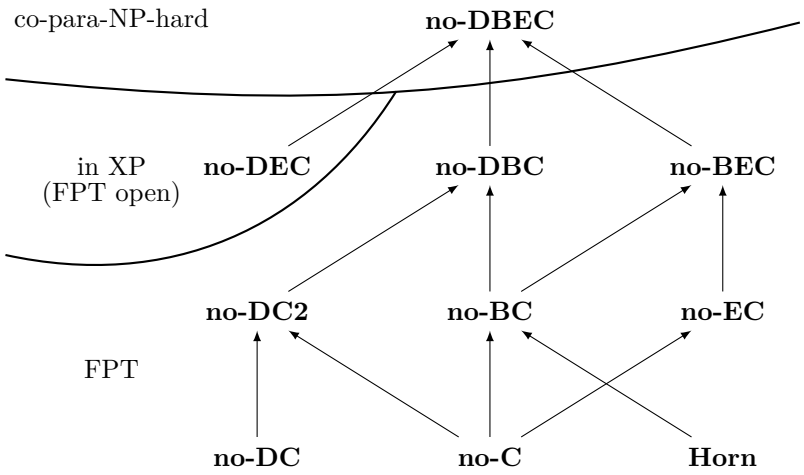


Figure 4.5: Relationship between classes of programs and known complexity of the problem Deletion $\mathcal{C}$-Backdoor Detection. An arrow from $\mathcal{C}$ to $\mathcal{C}'$ indicates that deletion $\mathcal{C}$-backdoors are smaller than deletion $\mathcal{C}'$-backdoors. The FPT-results are established in Theorems 4.4 and 4.18. The XP-result is established in Theorem 4.20. The co-para-NP-hardness result is established in Theorem 4.21.

hence the size of a smallest strong $\mathcal{C}'$-backdoor is at most the size of a smallest strong $\mathcal{C}$-backdoor.

Remark. Note that the unparameterized version of Strong $\mathcal{C}$-Backdoor Detection for the acyclicity-based target classes $\mathcal{C}$ is NP-hard since W[2]-hardness implies NP-hardness and co-para-NP-hardness also implies co-NP-hardness. Moreover, the reduction presented in the proof of Proposition 4.15 and Theorems 4.16, 4.17 can be used straightforward as a reduction of the unparameterized version of Hitting Set, Directed Path via a Node respectively, to Strong $\mathcal{C}$-Backdoor Detection.

4.2.2 Deletion Backdoor Detection

The W[2]-hardness results of Theorems 4.16 and 4.17 suggest to relax the problem and to look for *deletion backdoors* instead of strong backdoors. In view of Lemma 3.7 and Observation 4.11, every deletion backdoor into one of the considered acyclicity-based target classes is also a strong backdoor into that target class, hence the backdoor approach of Theorem 3.12 works.

Fortunately, the results of this section show that the relaxation indeed gives us fixed-parameter tractability of backdoor detection for most considered classes. Figure 4.5 illustrates these results that we obtain by making use of very recent progress in fixed-parameter algorithmics on various variants of *feedback vertex set* and *cycle transversal* problems.

Consider a graph $G = (V, E)$ and a set $W \subseteq V$. A cycle in G is a W-cycle if it contains at least one vertex from W. A set $T \subseteq V$ is a *W-cycle transversal* of G if every W-cycle of G is also a T-cycle. A set $T \subseteq V$ is an *even-length W-cycle transversal* of G if every W-cycle of G of even length is also a T-cycle. A V-cycle transversal is also called a *feedback vertex set*.

We give analog definitions for a digraph $G = (V, E)$ and $W \subseteq V$. A directed cycle in G is a directed W-cycle if it contains at least on vertex from W. A set $T \subseteq V$ is a *directed W-cycle transversal* of G if every directed W-cycle of G is also a directed T-cycle. A set $T \subseteq V$ is an *directed even-length W-cycle transversal* of G if every directed W-cycle of G of even length is also a directed T-cycle. A directed V-cycle transversal is also called a *directed feedback vertex set*.

Theorem 4.18. *The problem* Deletion $\mathcal{C}$-Backdoor Detection *is fixed-parameter tractable for all* $\mathcal{C} \in \mathcal{Acyc} \setminus \{\textbf{no-DEC}, \textbf{no-DBEC}\}$.

Proof. Let P be a program and $k \geq 0$. Let U_p be the dependency graph and D_p the dependency digraph of P, respectively. Next we consider the various target classes $\mathcal{C}$ mentioned in the statement of the theorem, one by one, and show how we can decide whether P has a deletion $\mathcal{C}$-backdoor of size at most k.

First we consider "undirected" target classes. Downey and Fellows [DF99; DF13] have shown that finding a feedback vertex set of size at most k is fixed-parameter tractable. We apply their algorithm to the dependency graph U_p. If the algorithm produces a feedback vertex set S of size at most k, then we can form a deletion **no-C**-backdoor of P of size at most k by replacing each negative vertex in S by one of its two neighbors, which always gives rise to an atom of P. If U_p has no feedback vertex set of size at most k, then P has no deletion **no-C**-backdoor of size at most k. Hence Deletion **no-C**-Backdoor Detection is fixed-parameter tractable. Similarly, Deletion **no-BC**-Backdoor Detection is fixed-parameter tractable by finding a W-feedback vertex set of U_p, taking as W the set of bad vertices of U_p. Cygan et al. [Cyg+13] and Kawarabayashi and Kobayashi [KK12] showed that finding a W-feedback vertex set is fixed-parameter tractable, hence so is Deletion **no-BC**-Backdoor Detection.

In order to extend this approach to Deletion **no-EC**-Backdoor Detection, we would like to use fixed-parameter tractability of finding an even W-cycle transversal, which was established by Misra et al. [Mis+12] for $W = V$, and by Kakimura, Kawarabayashi, and Kobayashi [KKK12] for general W. In order to do this, we use the following trick of Aracena, Gajardo, and Montalva [MAG08] that turns cycles containing an even number of bad vertices into cycles of even length. From D_p we obtain a graph U'_p by replacing each negative edge $e = (x, y)$ with three edges xu_e, $u_e v_e$, and $v_e y$ where u_e and v_e are new negative vertices, and by replacing each remaining directed edge (u, v) with two edges uw_e and $w_e v$ where w_e is a new (non-negative) vertex. We observe that U'_p can be seen as being obtained from D_p by subdividing edges. Hence there is a natural 1-to-1 correspondence between cycles in U_p and cycles in U'_p. Moreover, a cycle of U_p

containing an even number of negative vertices corresponds to a cycle of U'_p of even length, and a bad cycle of U_p corresponds to a bad cycle of U'_p. Thus, when we have an even cycle transversal S of U'_p, we obtain a deletion **no-EC**-backdoor by replacing each negative vertex $v \in S$ by its non-negative neighbor. Hence Deletion **no-EC**-Backdoor Detection is fixed-parameter tractable. For Deletion **no-BEC**-Backdoor Detection we proceed similarly, using an even W-cycle transversal of U'_p, letting W be the set of negative vertices of U'_p.

We now proceed with the remaining "directed" target classes **no-DC**, **no-DC2**, and **no-DBC**.

Let $G = (V, E)$ be a digraph. Evidently, the directed feedback vertex sets of D_p are exactly the deletion **no-DC**-backdoors of P. Hence, by using the fixed-parameter algorithm of Chen et al. [Che+08] for finding directed feedback vertex sets we obtain fixed-parameter tractability of Deletion **no-DC**-Backdoor Detection.

To make this work for Deletion **no-DC2**-Backdoor Detection we consider instead of D_p the digraph D'_p obtained from D_p by replacing each negative edge $e = (u, v)$ by two (non-negative) edges (u, w_e), (w_e, v), where w_e is a new vertex. The directed cycles of D_p and D'_p are in a 1-to-1 correspondence. However, directed cycles of length 2 in D'_p correspond to good cycles of length 2 in D_p. Bonsma and Lokshtanov [BL11] showed that finding a directed feedback vertex set that only needs to cut cycles of length at least 3 is fixed-parameter tractable. Applying this algorithm to D'_p (and replacing each vertex w_e in a solution with one of its neighbors) yields fixed-parameter tractability of Deletion **no-DC2**-Backdoor Detection.

The approach for Deletion **no-DC**-Backdoor Detection extends to Deletion **no-DBC**-Backdoor Detection by considering directed W-feedback vertex sets of the digraph D'_p obtained from D_p using a simple construction already mentioned by Cygan et al. [Cyg+13] where we replace each negative edge $e = (u, v)$ by two (non-negative) edges (u, w_e), (w_e, v) and $W = \{ w_e : e \text{ is a negative edge} \}$. The directed W-cycles of D'_p and the directed bad cycles of D_p are obviously in a 1-to-1 correspondence. Thus, when we have a directed W-feedback vertex set S of D'_p, we obtain a deletion **no-DBC**-backdoor by replacing each vertex $v \in S \cap W$ by its neighbor. The fixed-parameter tractability of finding a directed W-feedback vertex set was shown by Chitnis et al. [Chi+12]. □

According to Observation 4.11, the classes mentioned in Theorem 4.18 are hereditary. Hence using Theorem 4.18 we can drop the assumption in Theorem 3.12 that the backdoor is given. We obtain directly the following results:

Theorem 4.19. *For all $\mathcal{C} \in \mathcal{A}cyc \setminus \{$**no-DEC**, **no-DBEC**$\}$ all problems in $\mathcal{A}sp\mathcal{F}ull$ are fixed-parameter tractable when parameterized by the size of a smallest deletion $\mathcal{C}$-backdoor.*

Let us now turn to the two classes **no-DEC**, **no-DBEC** excluded in Theorem 4.18. We cannot establish that Deletion **no-DEC**-Backdoor Detection is fixed-parameter tractable, as the underlying even cycle transversal problem seems to be currently out of

reach to be solved. However, in Theorem 4.20 below, we can at least show that for every constant k, we can decide in polynomial time whether a strong **no-DEC**-backdoor of size at most k exists; thus, the problem is in XP. For Deletion **no-DBEC**-Backdoor Detection the situation is different: here we can rule out fixed-parameter tractability under the complexity theoretical assumption P $\neq$ co-NP (Theorem 4.21).

Theorem 4.20. *The problem* Deletion **no-DEC**-Backdoor Detection *is in* XP.

Proof. Let P be a program, n the input size of P, and k be a constant. We are interested in a deletion **no-DEC**-backdoor of P of size at most k. We loop over all possible sets $X \subseteq \mathrm{at}(P)$ of size at most k. Since k is a constant, there is a polynomial number $\mathcal{O}(n^k)$ of such sets X. To decide whether X is a deletion **no-DEC**-backdoor of P, we need to check whether $P - X \in$ **no-DEC**. For the membership check $P - X \in$ **no-DEC** we have to decide whether D_{P-X} contains a bad even cycle. We use a directed variant of the trick in the proof of Theorem 4.18 (in fact, the directed version is slightly simpler). Let D_{P-X} be the dependency digraph of $P - X$. From D_{P-X} we obtain a new digraph D'_{P-X} by subdividing every non-negative edge, i.e., we replace each non-negative edge $e = (x, y)$ by two (non-negative) edges (x, u_e), (u_e, y) where u_e is a new vertex. Obviously, directed even cycles in D_{P-X} are in 1-to-1 correspondence with directed cycles of even length in D'_{P-X}. Whether a digraph contains a directed cycle of even length can be checked in polynomial time by means of the following results: Vazirani and Yannakakis [VY88] have shown that finding a cycle of even length in a digraph is equivalent to finding a so-called Pfaffian orientation of a graph. Robertson, Seymour, and Thomas [RST99] have shown that a Pfaffian orientation can be found in polynomial time, hence the test works in polynomial time. □

Theorem 4.21. *The problem* Deletion **no-DBEC**-Backdoor Detection *is* co-para-NP*-hard, and hence not fixed-parameter tractable unless* P = co-NP

Proof. The theorem follows from the reduction in the proof of Theorem 4.17. □

Remark. We note that the unparameterized version of Deletion $\mathcal{C}$-Backdoor Detection for the acyclicity-based target classes $\mathcal{C}$ is NP-hard and co-NP-hard respectively, since the reduction presented in the proof of Theorems 4.18 and 4.20 can be used straightforward as a reduction of the unparameterized version of the variants of the minimal feedback vertex set problem, Directed Path via a Node respectively, to Deletion $\mathcal{C}$-Backdoor Detection.

Background and Related Work

The parameterized complexity of backdoor detection has been widely investigated in the context of propositional satisfiability. Nishimura, Ragde, and Szeider [NRS04] have first considered the target classes Horn and 2CNF and since then various classes have been investigated. Gaspers and Szeider [GS12b] have provided an overview on the Schaefer

target classes [Sch78] (including Horn), the base class renamable Horn, acyclicity-based and subsolver-based classes. Subsolver-based classes are closely related to variants of the well-known DPLL (Davis-Putnam-Logeman-Loveland) algorithm which forms the basis for DPLL-based Sat solvers. A subsolver is a variant of the DPLL-algorithm where certain solving techniques have been removed. Very recently Oikarinen and Järvisalo [OJ14] have introduced similar subsolver-based classes that are closely related to modern Asp algorithms. However, certain Asp techniques do not have a direct counterpart in Sat solving.

Ben-Eliyahu [Ben96] has proposed algorithms to determine answer sets of normal programs. The algorithms run faster on instances that have a small number of non-Horn rules or a small number of atoms in negative bodies which can be seen as exploiting Horn fragments. Gottlob, Scarcello, and Sideri [GSS02] have proposed fixed-parameter tractable algorithms to solve the main Asp reasoning problems when parameterized by the feedback width of a certain graph representation of a given normal program. Lin and Zhao [LZ04a] have established fixed-parameter tractable algorithms to determine answer sets of normal programs when parameterized in the size of directed bad even cycles.

Contribution and Discussion

In this chapter we have investigated the problem backdoor detection for backdoors into various enumerable target classes, in detail **Horn**, **no-C** (no cycles), **no-DC** (no directed cycles), **no-BC** (no bad cycles), **no-EC** (no even cycles), and their combinations. We would like to mention that some of the combinations (**no-DC2**, **no-EC**, **no-BEC**, and **no-DEC**) yield new-found target classes. We have established fixed-parameter tractability of backdoor detection for backdoors into the target class **Horn** by means of a reduction to the vertex cover problem. The technique is quite similar to backdoor detection for backdoors into the target class **Horn** in the propositional setting [NRS04]. The backdoor approach for **Horn** backdoors generalizes algorithms proposed by Ben-Eliyahu [Ben96] to determine answer sets of normal programs; later we provide a comprehensive comparison in Section 8.3. We have also provided strong evidence that strong backdoor detection for backdoors into the various acyclicity-based target classes is unlikely to be fixed-parameter tractable. However, we have established fixed-parameter tractability of deletion backdoor detection for backdoors into the considered classes $\mathcal{C}$ where $\mathcal{C}$ is a subclass of **no-DBC** or **no-BEC**. The results require new algorithms and deploy very recent algorithmic results from parameterized complexity theory on various versions of feedback vertex set problems to the domain of answer set programming. Our acyclicity-based target classes have no direct counterpart in the propositional setting, even so very recently discovered classes relate in a wider sense to acyclicity [GS12a; GS13]. The fixed-parameter tractable algorithms introduced by Gottlob, Scarcello, and Sideri [GSS02] to solve the problems in *AspReason* are based on deleting atoms from a dependency graph which is in fact a parameterization by deletion backdoors into the target class **no-C**. Our results are more general than the results by Gottlob, Scarcello, and Sideri [GSS02]; while the results by Gottlob, Scarcello, and Sideri [GSS02] are restricted to normal programs our

results apply to the full set of disjunctive programs. The results in earlier work [GSS02] deal with the target class **no-C** including improvements based on strongly connected components in contrast we establish fixed-parameter tractability even for the class of stratified programs (**no-DBC**) (see Section 8.4 for a detailed comparison). Moreover, we provide a comprehensive view on the parameterized complexity of strong and deletion backdoor detection for backdoors into combinations of the classes based on no cycles, no directed cycles, no bad cycles, and no even cycles. The notion of even cycle introduced by Lin and Zhao [LZ04a] relates to our notion of deletion backdoors into **no-DEC** and **no-DBEC**.

CHAPTER 5

Lifting Parameters

In this chapter we introduce a general method to lift Asp parameters that are defined for normal programs to disjunctive programs. Thereby we extend several algorithms that have been suggested for normal programs to disjunctive programs. The lifting method also gives us an alternative approach to obtain some results of Section 4.2.

In Section 5.1 we provide some general notation to speak about parameters of answer set programming. In Section 5.2 we introduce our novel lifting technique and prove that the size of a smallest deletion backdoor is not effected by lifting, due to the fact that lifting is based on deletion **Normal**-backdoors. In Section 5.3 we show that our lifting method extends fixed-parameter tractability of problems in *AspReason* when parameterized by an Asp parameter under the condition that both problems (i) deciding whether the paramter is of size at most k (parameter detection) and (ii) computing all answer sets (problem enumeration) are fixed-parameter tractable for normal programs when parameterized by the size of the parameter. This chapter is based on published work [FS15b].

5.1 ASP Parameters

The following definition allows us to speak about parameters for programs in a more abstract way.

Definition 5.1. *An* Asp parameter *is a function p that assigns to every program P some non-negative integer $p(P)$ such that $p(P') \leq p(P)$ holds whenever P' is obtained from P by deleting rules or deleting atoms from rules. If p is only defined for normal programs, we call it a* normal Asp parameter*. For an* Asp *parameter p we write $p^{\downarrow}$ to denote the* Asp *parameter obtained by restricting p to normal programs.*

We impose the condition $p(P') \leq p(P)$ for technical reasons. This is not a limitation, as most reasonable parameters and all parameters considered in this paper satisfy this condition.

There are natural Asp parameters associated with backdoors:

Definition 5.2. *For a class $\mathcal{C}$ of programs and a program P let $\mathrm{sb}_{\mathcal{C}}(P)$ denote the size of a smallest strong $\mathcal{C}$-backdoor and $\mathrm{db}_{\mathcal{C}}(P)$ denote the size of a smallest deletion $\mathcal{C}$-backdoor of P.*

5.2 Lifting

We will "lift" normal Asp parameters to general disjunctive programs as follows.

Definition 5.3. *For a normal Asp parameter p we define the Asp parameter $p^{\uparrow}$ by setting, for each disjunctive program P, $p^{\uparrow}(P)$ as the minimum of $|X| + p(P - X)$ over all inclusion-minimal deletion* **Normal***-backdoors X of P.*

The next lemma shows that this definition is compatible with deletion $\mathcal{C}$-backdoors if $\mathcal{C} \subseteq$ **Normal**. In other words, if $\mathcal{C}$ is a class of normal programs, then we can divide the task of finding a deletion $\mathcal{C}$-backdoor for a program P into two parts: (i) to find a deletion **Normal**-backdoor X, and (ii) to find a deletion $\mathcal{C}$-backdoor of $P - X$.

Lemma 5.4 (Self-Lifting)**.** *Let $\mathcal{C}$ be a class of normal programs. Then $\mathrm{db}_{\mathcal{C}}(P) = (\mathrm{db}_{\mathcal{C}}^{\downarrow})^{\uparrow}(P)$ for every program P.*

Proof. Let $\mathcal{C}$ be a class of normal programs, and P a program. Let X be a deletion $\mathcal{C}$-backdoor of P of size $\mathrm{db}_{\mathcal{C}}(P)$. Thus, $P - X \in \mathcal{C} \subseteq$ **Normal**. Hence X is a deletion **Normal**-backdoor of P. We select an inclusion-minimal subset X' of X that is still a deletion **Normal**-backdoor of P (say, by starting with $X' = X$, and then looping over all the elements x of X, and if $X' \setminus \{x\}$ is still a deletion $\mathcal{C}$-backdoor, then setting $X' := X' \setminus \{x\}$). What we end up with is an inclusion-minimal deletion **Normal**-backdoor X' of P of size at most $\mathrm{db}_{\mathcal{C}}(P)$. Let $P' = P - X'$ and $X'' = X \setminus X'$. P' is a normal program. Since $P' - X'' = P - X$, it follows that $P' - X'' \in \mathcal{C}$. Hence X'' is a deletion $\mathcal{C}$-backdoor of P. Thus, by the definition of $db_{\mathcal{C}}^{\uparrow}$, we have that $db_{\mathcal{C}}^{\uparrow}(P) \leq |X'| + |X''| = \mathrm{db}_{\mathcal{C}}(P)$.

Conversely, let $\mathrm{db}_{\mathcal{C}}^{\uparrow}(P) = k$. Hence there is a deletion **Normal**-backdoor X' of P such that $|X'| + \mathrm{db}_{\mathcal{C}}(P - X') = k$. Let $P' = P - X'$. Since $\mathrm{db}_{\mathcal{C}}(P') \leq k - |X'|$, it follows that P' has a deletion $\mathcal{C}$-backdoor X'' of size $k - |X'|$. We put $X = X' \cup X''$ and observe that $P - X = P' - X'' \in \mathcal{C}$. Hence X is a deletion $\mathcal{C}$-backdoor of P. Since $\mathrm{db}_{\mathcal{C}}(P) \leq |X| \leq |X'| + |X''| \leq \mathrm{db}_{\mathcal{C}}^{\uparrow}(P) \leq k$, the lemma follows. □

Example 5.1. Consider program P from Example 2.1 and let $\#\mathrm{neg}(P)$ denote the number of distinct atoms that occur in negative rule bodies of a normal program (we will discuss this parameter in more detail in Section 8.3).

We determine $\#\text{neg}^{\uparrow}(P) = 2$ by the following observations: The set $X_1 = \{c\}$ is a deletion **Normal**-backdoor of P since $P - X_1 = \{\, d \leftarrow a, e;\ a \leftarrow d, \neg b;\ e \leftarrow f;\ f \leftarrow d;\ \leftarrow f, e, \neg b;\ \leftarrow d;\ b;\ f \,\}$ belongs to the class **Normal**. The set $X_2 = \{e\}$ is a deletion **Normal**-backdoor of P since $P - X_2 = \{\, d \leftarrow a;\ a \leftarrow d, \neg b, \neg c;\ c \leftarrow f;\ f \leftarrow d, c;\ c \leftarrow f, \neg b;\ c \leftarrow d;\ b \leftarrow c;\ f \,\}$ belongs to the class **Normal**. Observe that X_1 and X_2 are the only inclusion-minimal deletion **Normal**-backdoors of the program P. We obtain $\#\text{neg}^{\uparrow}(P, X_1) = 2$ since $\#\text{neg}(P - X_1) = 1$. We have $\#\text{neg}^{\uparrow}(P, X_2) = 3$ since $\#\text{neg}(P - X_2) = 2$. Thus, $\#\text{neg}^{\uparrow}(P) = 2$. ⊣

For every Asp parameter p we consider the following problem.

Bound$[p]$

Given:	A program P and an integer k.
Parameter:	The integer k.
Task:	Decide whether $p(P) \leq k$ holds.

For a problem $L \in \mathcal{AspFull}$ and an Asp parameter p we write $L[p]$ to denote the problem L parameterized by p. That is, the instance of the problem is augmented with an integer k, the parameter, and for the input program P it holds that $p(P) \leq k$. Moreover, we write $L[p]_{\text{N}}$ to denote the restriction of $L[p]$ where instances are restricted to normal programs P. Similarly, Bound$[p]_{\text{N}}$ is the restriction of Bound$[p]$ to normal programs. For all the problems $L[p]_{\text{N}}$, p only needs to be a normal Asp parameter.

5.3 Lifting Theorem

Next we state the main result of this chapter.

Theorem 5.5 (Lifting)**.** *Let p be a normal* Asp *parameter such that* Bound$[p]_{\text{N}}$ *and* Enum$[p]_{\text{N}}$ *are fixed-parameter tractable. Then for all $L \in \mathcal{AspFull}$ the problem $L[p^{\uparrow}]$ is fixed-parameter tractable.*

We need some definitions and auxiliary results to establish the theorem.

Definition 5.6. *Let P be a disjunctive program. The* head dependency graph *H_p of the program P is the graph which has as vertices the atoms of P and an edge between any two distinct atoms if they occur together in the head of a rule of P.*

Lemma 5.7. *Let P be a disjunctive program. A set $X \subseteq \text{at}(P)$ is a deletion* **Normal***-backdoor of P if and only if X is a vertex cover of the head dependency graph H_p.*

Proof. Let X be a deletion **Normal**-backdoor of P. Consider an edge uv of H_p, then there is a rule $r \in P$ with $u, v \in H(r)$ and $u \neq v$. Since X is a deletion **Normal**-backdoor of P, we have $\{u, v\} \cap X \neq \emptyset$. We conclude that X is a vertex cover of H_p.

Conversely, assume that X is a vertex cover of H_p. We show that X is a deletion **Normal**-backdoor of P. Assume to the contrary, that $P - X$ contains a rule r whose head contains two variables u, v. Consequently, there is an edge uv in H_p such that $\{u, v\} \cap X = \emptyset$, contradicting the assumption that X is a vertex cover. □

Lemma 5.8. *Let $G = (V, E)$ be a graph, $n = |E|$, and let k be a non-negative integer. G has at most 2^k inclusion-minimal vertex covers of size at most k, and we can list all such vertex covers in time $\mathcal{O}(2^k n)$.*

Proof. We build a binary search tree T of depth at most k where each node t of T is labeled with a set S_t. We build the tree recursively, starting with the root r with label $S_r = \emptyset$. If S_t is a vertex cover of G we stop the current branch, and t becomes a "success" leaf of T. If t is of distance k from the root and S_t is not a vertex cover of G, then we also stop the current branch, and t becomes a "failure" leaf of T. It remains to consider the case where S_t is not a vertex cover and t is of distance smaller than k from the root. We pick an edge $uv \in E$ such that $u, v \notin S_t$ (such edge exists, otherwise S_t would be a vertex cover) and add two children t', t'' to t with labels $S_{t'} = S_t \cup \{u\}$ and $S_{t''} = S_t \cup \{v\}$. It is easy to see that for every inclusion-minimal vertex cover S of G of size at most k there is a success leaf t with $S_t = S$. Since T has $\mathcal{O}(2^k)$ nodes, the lemma follows. □

From Lemmas 5.7 and 5.8 we immediately obtain the next result.

Proposition 5.9. *Every disjunctive program of input size n has at most 2^k inclusion-minimal deletion* **Normal***-backdoors of size at most k, and all these backdoors can be enumerated in time $\mathcal{O}(2^k n)$.*

Proof of Theorem 5.5. Let p be a normal Asp parameter such that $\textsc{Enum}[p]_{\mathrm{N}}$ and $\textsc{Bound}[p]_{\mathrm{N}}$ are fixed-parameter tractable. Let P be a given disjunctive program of input size n and k an integer such that $p^{\uparrow}(P) \leq k$. In the following, when we say some task is solvable in "*fpt-time*", we mean that it can be solved in time $\mathcal{O}(f(k)\, n^c)$ for some computable function f and a constant c.

By Proposition 5.9 we can enumerate all inclusion-minimal deletion **Normal**-backdoors of size at most k in time $\mathcal{O}(2^k n)$. We can check whether $p(P - X) \leq k - |X|$ for each such backdoor X in fpt-time since $\textsc{Bound}[p]_{\mathrm{N}}$ is fixed-parameter tractable by assumption. Since $p^{\uparrow}(P) \leq k$, there is at least one such set X where the check succeeds.

We pick such set X and compute $\mathrm{AS}(P, X)$ in fpt-time. That this is possible can be seen as follows. Obviously, for each truth assignment $\tau \in 2^X$ the program P_τ is normal since $P - X$ is normal, and clearly $p(P_\tau) \leq p(P - X) \leq k$ by Definition 5.1. We can compute $\mathrm{AS}(P_\tau)$ in fpt-time since $\textsc{Enum}[p]_{\mathrm{N}}$ is fixed-parameter tractable by assumption. Since there are at most 2^k such programs P_τ, we can indeed compute the set $\mathrm{AS}(P, X)$ in fpt-time.

By Lemma 3.9 we have $\mathrm{AS}(P) \subseteq \mathrm{AS}(P, X)$, hence it remains to check for each $M \in \mathrm{AS}(P, X)$ whether it gives rise to an answer set of P. Since X is a deletion **Normal**-backdoor of P, and since one easily observes that **Normal** is hereditary, it follows by Lemma 3.7 that X is a strong **Normal**-backdoor of P. Hence Lemma 3.10 applies, and we can decide whether $M \in \mathrm{AS}(P)$ in time $\mathcal{O}(2^k n)$. Hence we can determine the set $\mathrm{AS}(P)$ in fpt-time. Once we know the set $\mathrm{AS}(P)$, we obtain for every problem $L \in \mathcal{AspFull}$ that $L[p^{\uparrow}]$ is fixed-parameter tractable. □

Example 5.2. Consider program P from Example 2.1 with the the deletion **Normal**-backdoor $X_1 = \{c\}$ from Example 5.1. We want to enumerate all answer sets of P. We obtain with Ben-Eliyahu's algorithm [Ben96] the sets $\mathrm{AS}(P_{\bar{c}}) = \{\{e, f\}\}$ and $\mathrm{AS}(P_c) = \{\{b, f\}\}$, and so we get the set $\mathrm{AS}(P, X) = \{\{e, f\}, \{b, c, f\}\}$ of answer set candidates. By means of the algorithm that solves the problem Strong $\mathcal{C}$-Backdoor Asp Check (see Lemma 3.10) we observe that $\{b, c, f\}$ is the only answer set of P. ⊣

Remark. Note that Definitions 4.2 and 4.6 introduce additional edges or certain cycles, respectively, on head atoms for non-singleton heads in the constructed graphs. This construction has the same effect on the size of smallest deletion backdoors into Horn or acyclicity-based target classes as the lifting of corresponding parameters from normal to disjunctive programs.

Background and Related Work

Parameterized complexity theory does not provide a general tool to lift parameters that are defined for a complexity class to a larger complexity class. Methods used in Asp to extend semantics or transformation from one class of programs into another class might be considered as a lifting technique. Results by Gelfond and Lifschitz [GL91] lift the stable model semantics to disjunctive programs which was defined only for normal programs [GL88]. Fages [Fag94] has shown that Clark's completion [Cla78] allows a transformation of tight programs into Sat. Lin and Zhao [LZ04a] have lifted Clark's completion from tight to normal programs by means of loop formulas. Later the results by Lee and Lifschitz [LL03] lifted loop formulas from normal to disjunctive programs.

Contribution and Discussion

We have established a lifting theorem that allows us to lift all parameters that are based on deleting rules or atoms from rules from normal programs to disjunctive programs. To our knowledge this is the first such framework for answer set programming. A similar technique might be useful for other problems from domains in artificial intelligence where the problem complexity lies beyond NP however when the input is restricted such that complexity is NP-complete the problem fixed-parameter tractable for a certain parameter.

CHAPTER 6

Polynomial-Time Preprocessing

In this chapter we consider kernelizations for backdoor detection and backdoor evaluation in the context of Asp. We establish that for some target classes, backdoor detection admits a polynomial kernel. We further provide strong theoretical evidence that for all target classes considered, backdoor evaluation does not admit a polynomial kernel.

If we want to solve a hard problem, then in many settings, it is beneficial to first apply a polynomial preprocessing to a given problem instance. In particular, polynomial-time preprocessing techniques have been developed in Asp solving (see e.g., [Fab+99; Geb+08; Geb+11a]). However, polynomial-time preprocessing for NP-hard problems has mainly been subject of empirical studies where provable performance guarantees are missing, mainly due to the fact, that if we can show that we can reduce in polynomial-time a problem instance by just one bit, then by iterating this reduction we can solve the instances in polynomial time. In contrast, the framework of parameterized complexity offers with the notion of *kernelization* a useful mathematical framework that admits the rigorous theoretical analysis of polynomial-time preprocessing for NP-hard problems.

In Section 6.1 we briefly provide some background and notation on kernelization. In Section 6.2 we investigate whether the deletion backdoor detection problems of Theorems 3.12, 4.4, and 4.18 also admit a polynomial kernel. Subsequently, we study in Section 6.3 polynomial preprocessing of the problems in *AspReason* when parameterized by the size of a strong backdoor into the considered target class and answer the question whether the problems admit a polynomial kernel. We conclude the chapter with background and related work, a summary of our contribution, and a discussion of the results. This chapter is based on published work [FS15b].

6.1 Kernelization

A kernelization is a polynomial-time reduction that replaces the input by a smaller input, called a "kernel", whose size is bounded by some computable function of the parameter only. A well known result of parameterized complexity theory is that a decidable problem is fixed-parameter tractable if and only if it admits a kernelization [DFS99]. The result leads us to the question of whether a problem also has a kernelization that reduces instances to a size which is polynomially bounded by the parameter, so-called *polynomial kernels*. Indeed, many NP-hard optimization problems admit polynomial kernels when parameterized by the size of the solution [Ros10].

We will later use the following problem:

Vertex Cover

Given:	A graph $G = (V, E)$ and an integer k.
Parameter:	The integer k.
Task:	Decide whether there is a vertex cover $S \subseteq V$ (see Section 4.1) of size at most k.

Next we give a more formal definition of kernelization. Let $L, L' \subseteq \Sigma^* \times \mathbb{N}$ be parameterized problems. A *bi-kernelization* is a polynomial-time many-to-one reduction from the problem L to problem L' where the size of the output is bounded by a computable function of the parameter. That is, a bi-kernelization is an algorithm that, given an instance $(I, k) \in \Sigma^* \times \mathbb{N}$ outputs for a constant c in time $\mathcal{O}((\|I\| + k)^c)$ a pair $(I', k') \in \Sigma^* \times \mathbb{N}$, such that (i) $(I, k) \in L$ if and only if $(I', k') \in L'$ and (ii) $\|I'\| + k' \leq g(k)$ where g is an arbitrary computable function, called the size of the kernel. If $L' = L$ then the reduction is called a *kernelization*, the reduced instance a *kernel*. If g is a polynomial then we say that L admits a *polynomial bi-kernel*, or *polynomial kernel* if in addition $L = L'$. For instance, the problem Vertex Cover has a kernel of at most $2k$ vertices and thus admits a polynomial kernel [CKX10]. L is called *compressible* if it admits a polynomial bi-kernel.

A kernelization of a decidable parameterized problem L immediately yields a fixed-parameter tractable algorithm for L since we can apply the kernelization and then use any brute force method and we still have a fixed-parameter tractable algorithm. On the other hand a fixed-parameter tractable algorithm gives a kernelization. Suppose that an algorithm A is fixed-parameter tractable, i.e., the algorithm A solves instances (I, k) in time $\mathcal{O}(f(k)\|I\|^c)$ for a computable function f and a constant c. Let (I, k) be an instance. If $\|I\| \leq f(k)$ then we output (I, k). If $\|I\| \geq f(k)$ then we apply the algorithm A which decides in time $\mathcal{O}(\|I\|^{c+1})$ whether $(I, k) \in L$. If $(I, k) \in L$ we output an arbitrary instance $(I', 1) \in L$; if $(I, k) \notin L$ we output an arbitrary instance $(I', 1) \notin L$ of the problem. In total the algorithm runs in time at most $\mathcal{O}(\|I\|^{c+1})$ which is in fact polynomial. The connection was observed by Downey, Fellows, and Stege [DFS99] and is stated in the following proposition:

The following proposition states the connection between fixed-parameter tractable problems and kernels, as observed by Downey, Fellows, and Stege [DFS99]:

Proposition 6.1 (Downey, Fellows, and Stege, 1999, Flum and Grohe, 2006)**.** *A parameterized problem is fixed-parameter tractable if and only if it is decidable and has a kernelization.*

Thus, our fixed-parameter tractability results of Theorems 3.12, 4.4, and 4.18 immediately provide that the mentioned problems admit a kernelization.

6.2 Backdoor Detection

In the following, we investigate whether the deletion backdoor detection problems for backdoors into the target class **Horn** (Section 4.1), the target class **Normal** (Section 5.3) or the acyclicity-based target classes (Section 4.2.2) admit polynomial kernels.

The first result of this section is quite positive.

Theorem 6.2. *For $\mathcal{C} \in \{\textbf{Horn}, \textbf{Normal}, \textbf{no-C}\}$ the problem* Deletion $\mathcal{C}$-Backdoor Detection *admits a polynomial kernel. For $\mathcal{C} \in \{\textbf{Horn}, \textbf{Normal}\}$ the kernel has a linear number of atoms, for $\mathcal{C} = \textbf{no-C}$ the kernel has a quadratic number of atoms.*

Proof. First consider the case $\mathcal{C} \in \{\textbf{Horn}, \textbf{Normal}\}$. Let (P, k) be an instance of Deletion $\mathcal{C}$-Backdoor Detection. We obtain in polynomial time the negation dependency graph N_p of P (see Definition 4.2) and consider (N_p, k) as an instance of Vertex Cover. Similar we obtain in polynomial time the head dependency graph H_p of P (see Definition 5.6) and consider (H_p, k), respectively, as an instance of Vertex Cover. We use the kernelization algorithm of Chen, Kanj, and Xia [CKX10] for Vertex Cover and reduce in polynomial time (N_p, k) to a Vertex Cover instance (G, k') where $G = (V, E)$ with at most $2k$ many vertices. It remains to translate G into a program P' where $N_{p'} = G$ by taking for every edge $xy \in E$ a rule $x \leftarrow \neg y$. Similar we use the kernelization algorithm of Chen, Kanj, and Xia [CKX10] for Vertex Cover and reduce in polynomial time (H_p, k) to a Vertex Cover instance (G, k') where $G = (V, E)$ with at most $2k$ many vertices. It remains to translate G into a program P' where $H_{p'} = G$ by taking for every edge $xy \in E$ a rule $x \vee y$. Now (P', k') is a polynomial kernel with a linear number of atoms. Now (P', k') is a polynomial kernel with a linear number of atoms.

Second consider the case $\mathcal{C} = \textbf{no-C}$. Let (P, k) be an instance of Deletion **no-C**-Backdoor Detection. We obtain in polynomial time the dependency graph U_p of P and consider (U_p, k) as an instance of Feedback Vertex Set (see Section 4.2.2). We use the kernelization algorithm of Thomassé [Tho09] for Feedback Vertex Set and reduce in polynomial time (U_p, k) to a Feedback Vertex Set instance (G', k') with at most $4k^2$ vertices. As above we translate $G = (V, E)$ into a program P' where

$U_{P'} = G$ by taking for every edge $xy \in E$ a rule $x \leftarrow \neg y$. Now (P', k') is a polynomial kernel with a quadratic number of atoms. □

Similar to the construction in the proof of Theorem 4.18 we can reduce for the remaining classes the backdoor detection problem to variants of feedback vertex set. However, for the other variants of feedback vertex set no polynomial kernels are known.

We would like to point out that the kernels obtained in the proof of Theorem 6.2 are equivalent to the input program with respect to the existence of a backdoor, hence the kernels can be used to find a backdoor. However the obtained kernels are not equivalent with respect to the decision of the problems in *AspReason*, in consequence the kernel cannot be used for backdoor evaluation. In the next subsection we consider kernels with respect to problems in *AspReason*.

6.3 Backdoor Evaluation

Next we consider the problems in *AspReason*. We will see that neither of them admits a polynomial kernel when parameterized by the size of a strong $\mathcal{C}$-backdoor into the considered target classes, subject to standard complexity theoretical assumptions.

Our superpolynomial lower bounds for kernel size are based on a result by Fortnow and Santhanam [FS11] regarding satisfiability parameterized by the number of variables.

Sat[Vars]

Given:	A CNF formula F.
Parameter:	The number k of variables of F.
Task:	Decide whether F is satisfiable.

Proposition 6.3 (Fortnow and Santhanam, 2011)**.** *If* Sat[Vars] *is compressible, then* NP ⊆ co-NP/poly.

Recall that the complexity class NP/poly consists of all problems that can be solved non-deterministically in non-uniform polynomial time (for a definition see Section 2.3). It is well-known from Yap's Theorem [Yap83] that if NP ⊆ co-NP/*poly* then the Polynomial Hierarchy collapses to its third level ($\Sigma_3^p = \Pi_3^p$), which is considered unlikely by standard complexity theoretical assumptions.

The following theorem extends a result for normal programs [GS14]. We need a different line of argument, as the technique used by Gaspers and Szeider [GS14] only applies to problems in NP or co-NP.

Theorem 6.4. *Let* $\mathcal{C} \in$ *Acyc* $\cup \{$**Horn**$\}$*. Then no problem in AspReason admits a polynomial kernel when parameterized by the size of a smallest strong* $\mathcal{C}$*-backdoor or deletion* $\mathcal{C}$*-backdoor, unless* NP ⊆ co-NP/poly.

Proof. We show that the existence of a polynomial kernel for any of the above problems implies that Sat[Vars] is compressible, and hence by Proposition 6.3 the collapse would follow.

First consider the problem Consistency. From a CNF formula F with k variables we use a reduction of Niemelä [Nie99] and construct a program P_1 as follows: Among the atoms of our program P_1 will be two atoms a_x and $a_{\bar{x}}$ for each variable $x \in var(F)$, an atom b_C for each clause $C \in F$. We add the rules $a_{\bar{x}} \leftarrow \neg a_x$ and $a_x \leftarrow \neg a_{\bar{x}}$ for each variable $x \in var(F)$. For each clause $C \in F$ we add for each $x \in C$ the rule $b_C \leftarrow a_x$ and for each $\neg x \in C$ the rule $b_C \leftarrow a_{\bar{x}}$. Additionally, for each clause $C \in F$ we add the rule $\leftarrow \neg b_C$. Now it is easy to see that the formula F is satisfiable if and only if the program P_1 has an answer set. We observe that $X = \{\, a_x : x \in var(F) \,\}$ ($X = \{\, a_x, a_{\bar{x}} : x \in var(F) \,\}$) is a smallest deletion (and smallest strong) $\mathcal{C}$-backdoor of P_1 for each $\mathcal{C} \in \mathcal{A}cyc$ ($\mathcal{C} = \mathbf{Horn}$). Hence (P_1, k), $(P_1, 2k)$ respectively, is an instance of Consistency, parameterized by the size of a smallest strong $\mathcal{C}$-backdoor or deletion $\mathcal{C}$-backdoor, and if this problem would admit a polynomial kernel, this would imply that Sat[Vars] is compressible.

For the problem Brave Reasoning we modify the reduction from above. We create a program P_2 that consists of all atoms and rules from P_1. Additionally, the program P_2 contains an atom t and a rule r with $H(r) = \{t\}$, $B^+(r) = \emptyset$, and $B^-(r) = \emptyset$. We observe that the formula F is satisfiable if and only if the atom t is contained in some answer set of P_2. Since X is still a backdoor of size k ($2k$), and a polynomial kernel for Brave Reasoning, again it would yield that Sat[Vars] is compressible.

Let UnSat[Vars] denote the problem defined exactly like Sat[Vars], just with yes and no answers swapped. A bi-kernelization for UnSat[Vars] is also a bi-kernelization for Sat[Vars] (with yes and no answers swapped). Hence Sat[Vars] is compressible if and only if UnSat[Vars] is compressible. An argument dual to the previous one for Brave Reasoning shows that a polynomial kernel for Skeptical Reasoning, parameterized by backdoor size, would yield that UnSat[Vars] is compressible, which, as argued above, would yield that Sat[Vars] is compressible. □

6.4 Background and Related Work

Kernelizations have been widely investigated in parameterized complexity. Standard notations are due to Downey, Fellows, and Stege [DFS99] and Flum and Grohe [FG06]. For a proof of the well-known connection between a fixed-parameter tractable problem and a parameterized problem we refer to other sources [DFS99; FG06]. Buss and Goldsmith [BG93] have discovered a quadratic kernel for the problem Vertex Cover. Chen, Kanj, and Jia [CKJ01] have observed that the kernel can be improved to at most $2k$ vertices and $\mathcal{O}(k^2)$ edges using Ilp methods to solve vertex cover [NL75] and later improved the results [CKX10]. Abu-Khzam et al. [Abu+07] have carried out an experimental evaluation of kernelizations for Vertex Cover. Lampis [Lam11] has established that Vertex Cover has a kernel of at most $2k - c \log k$ vertices for some fixed constant c.

Papadimitriou and Yannakakis [PY96] have observed that Vertex Cover does not have a kernel of size $\mathcal{O}(\log k)$ unless P = NP. Dell and Melkebeek [DM14] have improved the lower bounds by showing that there are no kernels with $\mathcal{O}(k^{2-\varepsilon})$ edges for every $\varepsilon > 0$ unless co-NP $\subseteq$ NP/poly.

Bodlaender, Thomassé, and Yeo [BTY11] have provided a way to transform kernel lower bounds from one parameterized problem to another. Fortnow and Santhanam [FS11] have established that Sat is not compressible unless NP $\subseteq$ co-NP/poly. Yap's Theorem [Yap83] establishes that the Polynomial Hierarchy collapses to its third level if NP $\subseteq$ co-NP/poly, which is considered unlikely by standard complexity theoretical assumptions. The result by Fortnow and Santhanam [FS11] provides the basis for our kernel lower bounds. Gaspers and Szeider [GS14] have considered kernelizations for various problems in artificial intelligence.

Several preprocessing techniques have been developed in Asp solving. Gebser et al. [Geb+08] have introduced an approach to simplify a program by identifying equivalences among its important parts and using them to build a compact representation of the program. Faber et al. [Fab+99] have suggested database optimization techniques for non-ground answer set programming. Morak and Woltran [MW12] have established a method based on kernelizations for preprocessing non-ground Asp. Bomanson, Gebser, and Janhunen [BGJ14] have determined an improved transformation of cardinality or weight rules into normal rules.

Even so the problems in *AspReason* do not not admit a polynomial kernel when parameterized by the size of a strong backdoor into the considered target classes under standard complexity theoretical assumptions, it might still be possible that the problems admit polynomially sized Turing kernels when parameterized by the size of a strong backdoor. In the area of optimization problems such Turing kernels, which consist of polynomial disjunction of polynomial kernels, have been discovered as an alternative approach. For example, Binkele-Raible et al. [Bin+12] have established that there is a polynomial-time reduction from the problem finding a rooted oriented spanning tree of at least k leaves to a disjunction of n independent kernels of size $\mathcal{O}(k^3)$ for a digraph of n vertices; although the problem does not admit a polynomial kernel unless the polynomial hierarchy collapses. Hence disjunctive kernelizations are stronger than the reduction to a single kernel.

6.5 Contribution and Discussion

In this chapter we analyzed preprocessing for backdoor detection and backdoor evaluation by means of kernelizations. We have obtained both positive and negative results. We have established that backdoor detection into the target classes **Horn**, **Normal**, and **no-C** admit a polynomial kernel; for Deletion **no-C**-Backdoor Detection the kernel has a quadratic number of atoms and for Deletion **Horn**-Backdoor Detection and Deletion **Normal**-Backdoor Detection the kernel has even a linear number of atoms. We obtain the results by standard methods from parameterized complexity theory, more

specifically reductions to Vertex Cover, which provide the linear kernels [CKX10], and feedback vertex set, which provides the quadratic kernel [Tho09]. We state that one would similarly obtain kernels for backdoor detection into the target classes, where backdoor detection is fixed-parameter tractable, by reductions to various feedback vertex set problems. However, it is currently an open question in parameterized complexity theory whether the various feedback vertex set problems admit polynomial kernels [Ros10]. We have established super-polynomial kernel lower bounds for backdoor evaluation by showing that a polynomial kernel for backdoor evaluation implies NP $\subseteq$ co-NP/poly, which then implies the collapse of the Polynomial Hierarchy to its third level according to Yap's theorem [Yap83], however the collapse is considered highly unlikely in standard complexity theory. In that way we obtain a strict theoretical bound for potential polynomial-time preprocessing methods for backdoor evaluation. Our kernel lower bounds are not limited to a particular preprocessing technique, they apply to any technique using backdoor evaluation.

We think an interesting future research topic, both from theoretical as well as from practical perspective, is to study whether backdoor evaluation admits a polynomially sized Turing kernel, i.e., a reduction for preprocessing that produces a disjunction of several independent polynomially sized kernels instead of a single kernel similar to results in the area of optimization [Bin+12]. A positive result on polynomial sized Turing kernels would motivate to consider disjunctive preprocessing also for practical solvers.

CHAPTER 7

Backdoor Trees

In this chapter we consider backdoor trees, which provide a more precise approach to the evaluation of strong backdoors where we take the interaction of the assignments in the evaluation into account. So far we considered only the size k of a backdoor as a parameter. Evaluating a given backdoor results in 2^k truth assignment reducts. However, a truth assignment to fewer than k atoms can already yield a truth assignment reduct that belongs to the considered target class. Therefore, we consider binary decision trees which make gradually assigning truth values to backdoor atoms in a program precise and lead us to the notion of backdoor trees, originally defined for SAT [SS08]. We investigate under which conditions (i) we need to consider significantly fewer than 2^k truth assignment reducts and (ii) we can significantly improve the minimality check from Lemma 3.10 if those truth assignments set only a small number of atoms to true.

In Section 7.1 we provide motivating examples and introduce basic notions and concepts to define backdoor trees for answer set programming. We state general observations of backdoor trees for answer set programming and connections between strong backdoors and backdoor trees. In Section 7.2 we extend the results on backdoor evaluation from Section 3.3 to backdoor trees. We establish that not all truth assignment reducts are necessary to find all answer sets of a given program instead the truth assignment reducts that we obtain from a backdoor tree are sufficient. We establish that the minimality check is fixed-parameter tractable when parameterized by a composed parameter which incorporates considerations of (i) and (ii) from above of a given backdoor tree and adapt our method to solve the problems in *AspReason* when parameterized by the composed parameter of a given backdoor tree. In Section 7.3 we state connections between backdoors and backdoor trees. We compare the parameter size of a backdoor and our composed parameter of a backdoor tree. In Section 7.4 we consider backdoor tree detection. We conclude the chapter with background and related work, a summary of our contribution, and a discussion. This chapter contains unpublished work.

7.1 Backdoor Trees of Answer Set Programs

In this section we introduce backdoor trees for answer set programming. Backdoor trees have been first introduced by Samer and Szeider [SS08] in the context of propositional satisfiability.

When we exploit a backdoor X of a program P to find answer sets according to Theorem 3.12, we determine for each $\tau \in 2^X$ the answer sets of the simplified programs P_τ and then check whether these answer sets give rise to an answer set of P. Consequently, we have to consider all the $2^{|X|}$ truth assignments. Since the exponential blowup of the running time of the algorithms in the proofs of Theorem 3.12 depends only on the size of the backdoor X, we were so far only interested in efficient algorithms to find smallest backdoors into certain target classes. However, there are answer set programs where we can find a backdoor X for which we do not need all $2^{|X|}$ truth assignments, as "shorter" truth assignments already yield a simplified program that belongs to the considered target class. More formally, there is a truth assignment τ' such that $\tau'^{-1} \subsetneq \tau^{-1}$ for some $\tau \in 2^X$ and the truth assignment reduct $P_{\tau'}$ already belongs to the target class $\mathcal{C}$. Hence, when we gradually assign truth values to atoms (similar to binary search) instead of taking a truth assignment $\tau \in 2^X$, some atoms in τ can be irrelevant for the question whether the simplified program belongs to $\mathcal{C}$.

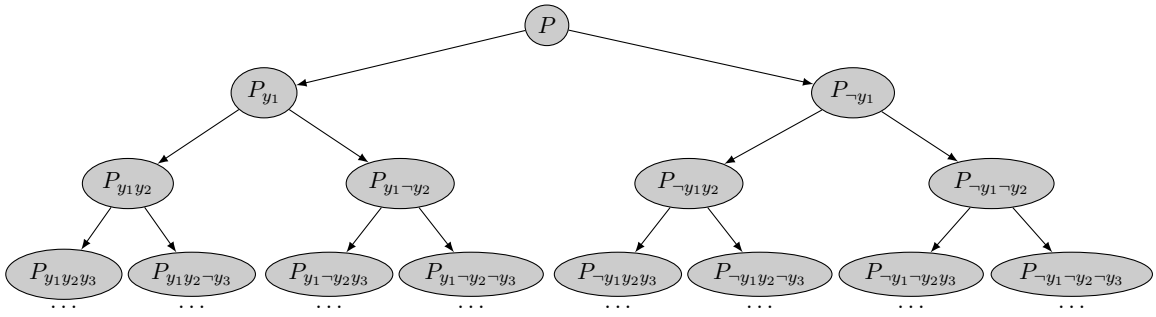
Interestingly, in some cases it is more effective to use a backdoor that is not a smallest backdoor into the target class $\mathcal{C}$. We show this in the following example.

Example 7.1. Let n be some integer. Consider the following program:

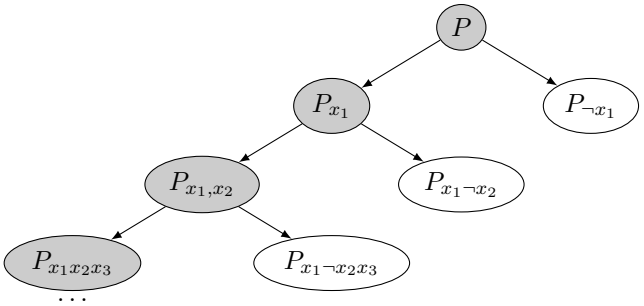
$$\begin{aligned} P := &\{\, y_0 \vee x_0 \leftarrow x_1, \ldots, x_{2n} \,\} \cup \\ &\{\, y_j \vee x_i \leftarrow x_0 \ldots, x_{i-1}, x_{i+1}, \ldots, x_{2n-1} : j = (i \bmod n), 1 \leq i < 2n \,\} \cup \\ &\{\, x_0 \leftarrow x_1, \ldots, x_{2n-1}, \neg y_j \,\} \\ &\{\, x_i \leftarrow x_0, x_{i-1}, x_{i+1}, x_{2n-1}, \neg y_j : j = (i \bmod n), 0 \leq i < 2n \,\}. \end{aligned}$$

We observe that $Y = \{y_0, \ldots, y_{n-1}\}$ is a smallest strong **Horn**-backdoor. Figure 7.1 (a) visualizes the truth assignments that we obtain when gradually constructing the truth assignment reducts $\tau \in 2^Y$. Obviously, we need all $2^{|Y|}$ truth assignment reducts since "removing" any atom from a truth assignment τ results in $P_\tau \notin$ **Horn**. The set $X = \{x_0, \ldots, x_{2n-1}\}$ is also a strong backdoor into **Horn**. The set X is larger than the set Y, but already for "shorter" truth assignments τ' than the truth assignment reducts $\tau \in 2^X$ we obtain that the truth assignment reduct belongs to **Horn**. For instance, the truth assignment $\tau' = \{\neg x_1\}$ yields the truth assignment reduct $P_{\neg x_1} = \{\, y_1 \leftarrow x_0, x_2, \ldots, x_{2n-1} \,\}$ which belongs to **Horn**. Hence, we obtain only $2n + 1$ truth assignment reducts, see Figure 7.1 (b). ⊣

Example 7.1 shows that gradually assigning backdoors (similar to binary search) can "earlier" yield truth assignment reducts that belong to the considered target class and that larger backdoors can yield an exponentially smaller number of such truth assignment



(a) Constructed from the strong **Horn**-backdoor Y



(b) Constructed from the strong **Horn**-backdoor X

Figure 7.1: Illustration of truth assignment reducts of the program P and the strong **Horn**-backdoors X and Y from Example 7.2. A gray colored note indicates that the respective program does not belong to **Horn**. A white colored note indicates that the respective program does belong to **Horn**.

reducts. In Section 3.3 we observed that the most important part for backdoor evaluation is to check whether a model is a minimal model (see proof of Lemma 3.10). The task is co-NP-complete in general, but fixed-parameter tractable when parameterized by the size of a smallest backdoor into a subclass of normal programs. For the minimality check we have to consider all backdoor atoms that have been set to true by any truth assignment. Hence the backdoor Y from Example 7.1 yields a significantly smaller number of truth assignment reducts, however for the minimality check we still have to consider all subsets of Y. Conversely, we construct subsequently in Example 7.2 a program where the number of truth assignments that we obtain from a smallest strong backdoor can be arbitrarily larger than the maximum number of atoms in a backdoor that have been set to true by any truth assignment on a much larger number of atoms.

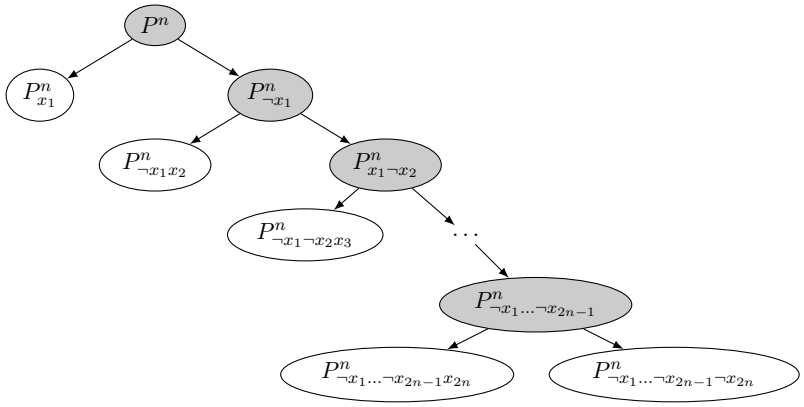


Figure 7.2: A **Horn**-backdoor tree $BT = (T, \chi)$ of program P^n from Example 7.2.

Example 7.2. Let n be some large integer. We define the following program:

$$\begin{aligned} P := {} & \{\, y_0 \vee \neg x_0 \leftarrow \neg x_1, \ldots, \neg x_{2n-1} \,\} \cup \\ & \{\, y_j \vee x_i \leftarrow \neg x_0 \ldots, \neg x_{i-1}, \neg x_{i+1}, \ldots, \neg x_{2n-1} : j = (i \bmod n), 1 \leq i < 2n \,\} \cup \\ & \{\, x_0 \leftarrow x_1, \ldots, x_{2n-1}, \neg y_j \,\} \\ & \{\, x_i \leftarrow \neg x_0, \neg x_{i-1}, \neg x_{i+1}, \neg x_{2n-1}, \neg y_j : j = (i \bmod n), 0 \leq i < 2n \,\}. \end{aligned}$$

We observe that $Y = \{y_0, \ldots, y_{n-1}\}$ is a smallest strong **Horn**-backdoor. Gradually constructing the truth assignment reducts as carried out above yields a complete tree with 2^n leaves and a maximum number n of atoms that might be set to true. Obviously, we need all $2^{|Y|}$ truth assignment reducts since "removing" any atom from a truth assignment τ results in $P_\tau \notin$ **Horn**. Further, we easily observe that $X = \{y_1, \ldots, y_n\}$ is a smallest strong **Horn**-backdoor. Figure 7.2 visualizes the truth assignments that we obtain when gradually constructing the truth assignment reducts $\tau \in 2^X$. There we have only $2n + 1$ truth assignment reducts where at most one atom is set to true. ⊣

Before we can make the observations from the previous examples precise, we provide some basic definitions. Let X be a set of atoms, $T = (N, E, r)$ a binary tree, and χ a labeling that maps any node $t \in N$ to a set $\chi(t) \subseteq \{\, a, \neg a : a \in X \,\}$. We denote by $X_1(t)$ the positive literals of the labeling $\chi(t)$, i.e., $X_1(t) := \chi(t) \cap X$. The *corresponding truth assignment* $\tau_{\chi(t)}$ of t is the truth assignment $\tau_{\chi(t)}$ where $\tau_{\chi(t)}(a) = 1$ if $a \in \chi(t)$ and $\tau_{\chi(t)}(a) = 0$ if $\neg a \in \chi(t)$. The pair $BT = (T, \chi)$ is a *binary decision tree* of P if the following conditions hold:

1. for the root r we have $\chi(r) = \emptyset$,
2. for any two nodes $t, t' \in N$, if t' is a child of t, then either $\chi(t') = \chi(t) \cup \{\neg a\}$ or $\chi(t') = \chi(t) \cup \{a\}$ for some atom $a \in X \setminus \tau^{-1}_{\chi(t)}$, and
3. for any three nodes $t, t_1, t_2 \in N$, if t_1 and t_2 are children of t, then $\chi(t_1) \neq \chi(t_2)$.

We denote by $\mathrm{at}(BT)$ the atoms *occurring* in truth assignments of BT, i.e.,

$$\mathrm{at}(BT) := \bigcup_{t \in N} \tau^{-1}_{\chi(t)}.$$

Next, we give a definition for backdoor trees of answer set programs.

Definition 7.1. *Let P be a program, $X = \mathrm{at}(P)$, and $BT = (T, \chi)$ be a binary decision tree and $T = (N, E)$. The pair $BT = (T, \chi)$ is a* $\mathcal{C}$-backdoor tree *of P if $P_\tau \in \mathcal{C}$ for every leaf $t \in N$ and $\tau = \tau_{\chi(t)}$. We denote by $\#\mathrm{leaves}(BT)$ the number of leaves of T, i.e.,*

$$\#\mathrm{leaves}(BT) := |\{\, t : t \text{ is a leaf of } T \,\}|.$$

We denote by $\mathrm{gs}(BT)$ the maximum number of atoms that have been set to true by a corresponding truth assignment of any leaf of T, more specifically,

$$\mathrm{gs}(BT) := \max\{\, |X_1(t)| : t \text{ is a leaf of } T \,\}.$$

For reasons explained below, we call $\mathrm{gs}(BT)$ the Gallo-Scutellà parameter *of BT.*

In other words, a backdoor tree of a program P is a binary decision tree where the nodes of the tree are labeled by truth assignments $\tau \in 2^X$ on subsets $X \subseteq \mathrm{at}(P)$, the corresponding partial assignment τ of an inner node yields a simplified program P_τ that does not belong to the considered target class, and the corresponding truth assignment τ of a leaf yields a simplified program P_τ that belongs to the considered target class.

Relationship to a Parameter by Gallo and Scutellà

The maximal number of positive variables in a propositional formula has been considered as parameter by Gallo and Scutellà [GS88] to measure in a certain sense the distance from being Horn. Recall that a formula is Horn if each clause has at most one positive literal. Gallo and Scutellà have also established an XP-algorithm (see Section 2.4) to determine the parameter of a propositional formula.

We consider the parameter in its original context and definition as nested classes of families of sets on a family to generalize Horn formulas. Let S be a family of sets $S_1, \ldots, S_m$, $\mathsf{S}_X = \mathsf{S} \setminus \{\, Y \in \mathsf{S} : X \subseteq Y \,\}$, and $\mathsf{S} - X := \{\, S \setminus X : S \in \mathsf{S} \,\}$ for some set X. Moreover, (i) $\mathsf{S} \in \Sigma_0$ if and only if $|S| \leq 1$ for each $S \in \mathsf{S}$ and (ii) $\mathsf{S} \in \Sigma_k$ if and only if there is some $v \in \bigcup_{1 \leq i \leq m} S_i$ such that $S_{\{v\}} \in \Sigma_{k-1}$ and $\mathsf{S} - \{v\} \in \Sigma_k$. Then, the class Γ_k consists of all propositional formulas F such that $F' \in \Sigma_k$ where F' is obtained from F by removing all negative literals (note that we consider F' as a set of clauses and a clause is a set of variables). Observe that Γ_0 consists of all Horn formulas.

A *backdoor tree of F into Horn formulas* is a binary decision tree where the formula F_τ is Horn for each leaf t and its corresponding truth assignment τ.

Proposition 7.2. *A propositional formula F belongs to Γ_k if and only if there is a backdoor tree $BT = (T, \chi)$ into Horn formulas of F such that* $\mathrm{gs}(BT) \leq k$.

Proof. Let F be a propositional formula and $BT = (T, \chi)$ be a backdoor tree BT into Horn formulas of F of Gallo-Scutellà parameter k. Observe that the labelings χ of the paths in T from the root to a leaf provide witnesses for Conditions (i) and (ii). Hence, $F \in \Gamma_k$. Conversely, we construct a $\mathcal{C}$-backdoor tree from the fact that $F \in \Gamma_k$. Take the binary decision tree (T, χ) where $T = (N, E, r)$ and $E = \emptyset$. Since $F \in \Gamma_k$, the formula $F' \in \Sigma_k$ where F' is obtained from F by removing all negative literals. By Condition (ii) there is a variable v such that $F'_{\{v\}} \in \Sigma_{k-1}$ and $F' - \{v\} \in \Sigma_k$. We add two fresh nodes n_1 and n_2 to N, edges (r, n_1) and (r, n_2) to E, and extend the mapping χ by labelings $\chi(n_1) := \chi(r) \cup \{v\}$ and $\chi(n_2) := \chi(r) \cup \{\neg v\}$. We proceed inductively with $F'_{\{v\}}$ and Σ_{k-1} for the leaf n_1 of T and $F' - \{v\}$ and Σ_k for the leaf n_2 of T. We easily observe that by construction BT is a $\mathcal{C}$-backdoor tree and since at most k variables are set to true $\mathrm{gs}(BT) \leq k$. □

7.2 Backdoor Tree Evaluation

In this section we establish for binary decision trees an analogue to Lemma 3.9. Again we consider the truth assignment reducts P_τ together with the atoms that are set to true and extend this notion to the corresponding truth assignments of the leaves for binary decision trees.

Definition 7.3. *Let P be a program, $X = \mathrm{at}(P)$, and $BT = (T, \chi)$ a binary decision tree.*

$$\mathrm{AS}(P, \tau) := \{\, M \cup \tau^{-1}(1) : M \in \mathrm{AS}(P_\tau) \,\} \quad \textit{and}$$
$$\mathrm{AS}(P, BT) := \{\, M : \ t \textit{ is a leaf of } T, \tau = \chi(t), M \in \mathrm{AS}(P, \tau) \,\}.$$

Similarly, to a subset of the atoms of a program and the resulting partial truth assignments (cf. Lemma 3.9), we obtain "answer set candidates" by evaluating the corresponding truth assignments of the leaves of a binary decision tree.

Lemma 7.4. *Let P be a program, $BT = (T, \chi)$ a binary decision tree of P, and $X := \mathrm{at}(BT)$. Then* $\mathrm{AS}(P) \subseteq \mathrm{AS}(P, BT) \subseteq \mathrm{AS}(P, X)$.

Proof. We first show that $\mathrm{AS}(P, BT) \subseteq \mathrm{AS}(P, X)$. By Definitions 3.8 and 7.3 we have $\mathrm{AS}(P, X) = \bigcup_{\tau \in 2^X} \mathrm{AS}(P, \tau)$. Let $U_\tau := \{\, \tau' : \tau' \in 2^X, \tau'(a) = \tau(a) \text{ for every } a \in \tau^{-1} \,\}$, in other words, U_τ contains all truth assignments $\tau' \in 2^X$ such that $\tau^{-1} \subseteq \tau'^{-1}$ for a possible truth assignment $\tau' \in 2^X$. It remains to observe that $\mathrm{AS}(P, \tau) \subseteq \bigcup_{\tau' \in U_\tau} \mathrm{AS}(P, \tau')$. We define $l := |\tau^{-1} \setminus \tau'^{-1}|$ for some truth assignments τ and τ' where $\tau^{-1} \subseteq \tau'^{-1}$ and proceed an induction proof on l. The case $l = 0$ is obvious. The case $l = 1$ follows simply from Lemma 3.9 since $\mathrm{AS}(P, \tau) = \mathrm{AS}(P, \tau_0) \cup \mathrm{AS}(P, \tau_1)$ where

$\tau_0(a) = 0$ and $\tau_1(a) = 1$ for the atom $a \in \tau^{-1} \setminus \tau'^{-1}$. Consider the case $l > 1$. Let τ'' be a truth assignment such that $\tau'^{-1} = \tau''^{-1} \setminus \{a\}$. Then by Lemma 3.9 we obtain $\mathrm{AS}(P, \tau'') \subseteq \mathrm{AS}(P, \tau'_0) \cup \mathrm{AS}(P, \tau'_1)$ where $\tau'_0(a) = \tau'_1(a) = \tau''(a)$ for each $a \in \tau''^{-1}$, $\tau'_0(b) = 0$ and $\tau'_1(b) = 1$ for some $b \notin \tau^{-1}$. Since, $\mathrm{AS}(P, \tau) \subseteq \mathrm{AS}(P, \tau'')$ by induction, we conclude $\mathrm{AS}(P, \tau) \subseteq \bigcup_{\tau' \in U_\tau} \mathrm{AS}(P, \tau')$.

The proof of $\mathrm{AS}(P) \subseteq \mathrm{AS}(P, BT)$ is quite similar to the proof of Lemma 3.9. Let $M \in \mathrm{AS}(P)$ be chosen arbitrarily. We consider the truth assignments $\tau = \chi(t)$ for each leaf t of T. Let $M' = M \setminus \tau^{-1}(1)$. Observe that $M' \in \mathrm{AS}(P_\tau)$ implies $M \in \mathrm{AS}(P, BT)$ since $M = M' \cup \tau^{-1}(1)$ by definition. Hence, to establish the lemma, it suffices to show that $M' \in \mathrm{AS}(P_\tau)$. We have to show that M' is a model of $P_\tau^{M'}$, and that no proper subset of M' is a model of $P_\tau^{M'}$ (which we already carried out in the proof of Lemma 3.9). Consequently, $\mathrm{AS}(P) \subseteq \mathrm{AS}(P, BT)$ and the Lemma is established. □

Theorem 7.5. *Let $\mathcal{C} \subseteq$* **Normal** *be an enumerable class. The problems in AspFull are all fixed-parameter tractable when parameterized by $\mathrm{gs}(BT) + \#\mathrm{leaves}(BT)$ of a $\mathcal{C}$-backdoor tree BT, assuming that the backdoor tree is given as input.*

Before proving this Theorem we need to make some observations. In view of Lemma 7.4 we have to consider the corresponding truth assignment reducts of the leaves in the backdoor tree. For each truth assignment $\tau \in \mathrm{ta}(T)$ we reduce the program P to a program P_τ and compute the set $\mathrm{AS}(P_\tau)$. Then, we obtain the set $\mathrm{AS}(P)$ by checking for each $M \in \mathrm{AS}(P_\tau)$ whether it gives rise to an answer set of P. The crucial part is again to consider minimality with respect to the Gelfond-Lifschitz reduct. A similar construction as in Lemma 3.9 can be used. For the leaf t and its corresponding truth assignment τ we can guarantee minimality with respect to the reduct $(P_\tau)^M$. Setting atoms to true by the truth assignment τ does apparently not guarantee minimality with respect to P^M (cf. Lemma 7.4). Hence, we have to check for each atom in $\tau^{-1}(1)$ whether there is a "justification" to set the atom to true. We consider the following parameterized problem.

$\mathcal{C}$-Backdoor Tree Asp Check(GS)

Given:	A program P, a $\mathcal{C}$-backdoor tree $BT = (T, \chi)$ of P, a leaf t of T, and a set $M \subseteq \mathrm{at}(P)$.
Parameter:	The Gallo-Scutellà parameter $k = \mathrm{gs}(BT)$.
Task:	Decide whether M is an answer set of P.

We establish the following result.

Proposition 7.6. *Let $\mathcal{C} \subseteq$* **Normal**. *The problem $\mathcal{C}$-Backdoor Tree Asp Check(GS) is fixed-parameter tractable. More specifically, given a program P of input size n, a $\mathcal{C}$-backdoor tree $BT = (T, \chi)$ of P of Gallo-Scutellà parameter $k = \mathrm{gs}(BT)$, a leaf t of T, and a set $M \subseteq \mathrm{AS}(P, \tau_{\chi(t)})$ of atoms, we can check in time $\mathcal{O}(2^k \cdot n)$ whether M is an answer set of P.*

Proof. We would like to use the same construction as in Lemma 3.10 and therefore need slightly stronger arguments.

Let $BT = (T, \chi)$ be a $\mathcal{C}$-backdoor tree of P. We first check whether M is a model of P^M. If M is not a model of P^M, then M cannot be an answer set of P. Hence, assume that M is a model of P^M.

We follow the construction in the proof of Lemma 3.10 and construct from P^M a program $P^M_{Y \subseteq X_1(t)}$ by (i) removing all rules r for which $H(r) \cap Y \neq \emptyset$, and (ii) replacing for all remaining rules r the head $H(r)$ with $H(r) \setminus X_1(t)$, and the positive body $B^+(r)$ with $B^+(r) \setminus Y$.

Claim: $P^M_{Y \subseteq X_1(t)}$ is Horn.

We first establish that $P_{Y \subseteq X_1(t)}$ is normal. Since $BT = (T, \chi)$ is a backdoor tree, the partial truth assignment reduct P_τ is normal for each leaf t of T and $\chi := \tau_{\chi(t)}$. Let r' be an arbitrarily chosen rule in P_τ. Then there is a corresponding rule $r \in P$ and a corresponding rule $r'' \in P_{Y \subseteq X_1(t)}$. Since we remove in both constructions exactly the same literals from the head of every rule, $H(r') = H(r'')$ holds. Consequently, $P_{Y \subseteq X_1(\tau)}$ is normal and $P^M_{Y \subseteq X_1(\tau)}$ is Horn.

We use the same procedure as in the proof of Lemma 2.1.

> If $P^M_{Y \subseteq X_1(t)}$ has no model, then stop and return *True*.
>
> Otherwise, compute the unique minimal model L of the Horn program $P^M_{Y \subseteq X_1(t)}$. If $L \subseteq M \setminus X$, $L \cup Y \subsetneq M$, and $L \cup Y$ is a model of P^M, then return *False*. Otherwise return *True*.

Again for each set $Y \subseteq M \cap X_1(t)$ the above procedure runs in linear time by Lemma 2.1. By Lemma 7.4 we consider only the atoms that have been set to true in M, i.e., $Y \subseteq X_1(t)$ and run the algorithm from above for each $X_1 \subseteq X \cap M$ and M is a minimal model of P^M if and only if the algorithm returns *True* for each $Y \subseteq X_1(t)$. Consequently, we have to consider at most $2^{|X_1(t)|}$ many subsets of $X_1(t)$. Since $k = \max\{\, |X| : X \in X_1(T) \,\}$, we obtain a running time of $\mathcal{O}(2^k \|P\|)$.

Claim: M is a minimal model of P^M if and only if the algorithm returns True for each $Y \subseteq M \cap X_1(t)$.

The claim follows directly from the proof of Lemma 3.10 and establishes the correctness of the above procedure.

Consequently the problem is fixed-parameter tractable when parameterized by k.

□

We are now in position to establish Theorem 7.5.

Proof of Theorem 7.5. The proof is similar to the proof of Theorem 3.12. Let $BT = (T, r, \chi)$ be the given $\mathcal{C}$-backdoor tree, $g = \text{gs}(BT)$, $l = \#\text{leaves}(BT)$, $T = (N, E, r)$, and n the input size of P. According to Lemma 7.4, $\text{AS}(P) \subseteq \text{AS}(P, BT)$. Since $P_\tau \in \mathcal{C}$ and $\mathcal{C}$ is enumerable, we can compute $\text{AS}(P_\tau)$ in polynomial time for each leaf $t \in N$ and $\tau = \tau_{\chi(t)}$, say in time $\mathcal{O}(n^c)$ for some constant c. Hence $|\text{AS}(P_\tau)| \leq \mathcal{O}(n^c)$ for each leaf $t \in N$ and $\tau = \tau_{\chi(t)}$. By Proposition 7.5 we can decide whether $M \in AS(P)$ in time $\mathcal{O}(2^g \cdot n^c)$ and $|\text{AS}(P, \tau)| \leq \mathcal{O}(2^g \cdot n^c)$ for each $M \in \text{AS}(P, \tau)$ where $\tau = \tau_{\chi(t)}$ and t is a leaf of T. Since there are at most l many leaves, we can compute $\text{AS}(P, T)$ and check whether for $M \in \text{AS}(P, T)$ also $M \in \text{AS}(P)$ holds in time $\mathcal{O}(l \cdot 2^g \cdot n^c)$ and $|\text{AS}(P, T)| \leq \mathcal{O}(l \cdot 2^g \cdot n^c)$. Then we can also solve all problems in *AspFull* from $\text{AS}(P)$ within polynomial time. Consequently, the problem is fixed-parameter tractable when parameterized by $g + l$.

□

There are two factors for hardness of Asp problems when parameterized by the Gallo-Scutellà parameter plus the size a backdoor tree (i) atoms that are set to true which yield potential candidates and are hence important for the minimality check in each leaf; and (ii) leaves in a backdoor tree which yield the truth assignment reducts we have to consider. Both factors of hardness are "used" in the proof of Theorem 7.5. Hence, in contrast to Sat backdoor trees we do not simply parameterize the reasoning problems in *AspReason* by $\#\text{leaves}(BT)$ of a given backdoor tree $BT = (T, \chi)$ of P to obtain a more refined view on backdoors. Instead we also consider $\text{gs}(BT)$ which is the maximum number of atoms that are set to true in a leaf of T. This is attributed to the minimality check where we have to consider the number of atoms that are set to true.

7.3 Relation to Backdoors

In this section we investigate on connections between backdoors and backdoor trees. We show that our composed parameter based on backdoor trees is more general than the size of a backdoor.

Lemma 7.7. *Let P be a program and $\mathcal{C}$ be an hereditary class of programs. If BT is a $\mathcal{C}$-backdoor tree of P, then $\text{at}(BT)$ is a strong $\mathcal{C}$-backdoor of P.*

Proof. Let $X = \text{at}(BT)$. For every leaf $t \in T$ we have $P_\tau \in \mathcal{C}$ where $\tau = \tau_{\chi(t)}$ according to Definition 7.1. Then, we observe that one obtains all truth assignments $2^{|X|}$ simply by extending the truth assignments $\tau = \tau_{\chi(t)}$ by truth assignments τ' on the atoms $\text{at}(BT) \setminus \tau^{-1}$. Since $\mathcal{C}$ is hereditary and already $P_\tau \in \mathcal{C}$, also $P_{\tau \cup \tau'} \in \mathcal{C}$. Hence, for all possible truth assignments $\tau \in 2^X$ we have $P_\tau \in \mathcal{C}$. Consequently, X is a strong $\mathcal{C}$-backdoor of P and the lemma holds. □

We make the following observations about binary decision trees.

Observation 7.8. *Let BT be a binary decision tree. Then,*

$$\mathrm{gs}(BT) \leq |\mathrm{at}(BT)| \leq \#\mathrm{leaves}(BT) - 1.$$

Proof. We observe the inequalities from the fact that we add in every level exactly one atom to the labeling. □

Observation 7.9. *Let BT be a binary decision tree, $n = |\mathrm{at}(BT)|$, $g = \mathrm{gs}(BT)$, and $l = \#\mathrm{leaves}(BT)$. Then, $l \leq (1+n)^g$.*

Proof. Let $BT = (T, \chi)$ a binary decision tree, $T = (N, E, r)$, $n = |\mathrm{at}(BT)|$, $g = \mathrm{gs}(BT)$, and $l = \#\mathrm{leaves}(BT)$. According to Definition 7.1, for every leaf t of T we have $|\chi(t)| \leq n$ and at most g atoms are set to true in the corresponding truth assignment $\tau_{\chi(t)}$. Hence, we have at most $\sum_{i=0}^{g} \binom{n}{i}$ possible combinations of truth assignments. Thus, $l \leq \sum_{i=0}^{g} \binom{n}{i}$. By binomial expansion we obtain $\sum_{i=0}^{g} \binom{n}{i} \leq \sum_{i=0}^{g} (n^i \cdot 1^{g-i}) \leq (1+n)^g$. □

We establish that every strong backdoor of size k yields a backdoor tree consisting of at least $k+1$ leaves and at most 2^k leaves.

Lemma 7.10. *Let P be a program, $\mathcal{C}$ an hereditary class of programs, X a strong $\mathcal{C}$-backdoor of smallest size of P, and $BT = (T, \chi)$ a $\mathcal{C}$-backdoor tree of smallest number of leaves of P. Then,*

$$|X| + 1 \leq \#\mathrm{leaves}(BT) \leq 2^{|X|}.$$

Proof. Let P be a program, X a strong $\mathcal{C}$-backdoor of smallest size of P, and $BT = (T, \chi)$ a $\mathcal{C}$-backdoor tree of smallest number of leaves of P. According to Lemma 7.7 the set $\mathrm{at}(BT)$ is also a strong $\mathcal{C}$-backdoor of P. By Observation 7.8 and the definition a backdoor tree we have $|\mathrm{at}(BT)| \leq \#\mathrm{leaves}(BT) - 1$. It remains to observe that we can construct from X a complete binary decision tree (T', χ) of P with $2^{|X|}$ leaves by labeling the root of T' by $\emptyset$, and for each level the nodes by an a or $\neg a$ (which did not occur in a lower level) for $a \in X$ in an arbitrary fixed order. We obtain for a leaf t that $\chi(t) = \tau$ if and only if $\tau \in 2^X$. Hence, $P_\tau \in \mathcal{C}$ for every $\tau = \chi(t)$ and leaf t. Then (T', χ) is a $\mathcal{C}$-backdoor tree of P where T has $2^{|X|}$ many leaves. Since the number of leaves of T is less or equal the number of leaves of T', we conclude $\#\mathrm{leaves}(BT) \leq 2^{|X|}$. Consequently, the lemma holds. □

Lemma 7.11. *Let P be a program, $\mathcal{C}$ a hereditary class of programs, X a strong $\mathcal{C}$-backdoor of smallest size of P, and $BT = (T, \chi)$ a $\mathcal{C}$-backdoor tree of smallest Gallo-Scutellà parameter $\mathrm{gs}(BT)$ of P. Then, $\mathrm{gs}(BT) \leq |X|$.*

Proof. Let P be a program, X a strong $\mathcal{C}$-backdoor of smallest size of P, and $BT = (T, \chi)$ a $\mathcal{C}$-backdoor tree of smallest Gallo-Scutellà parameter $\mathrm{gs}(BT)$ of P. Again we can construct from X a $\mathcal{C}$-backdoor tree (T', χ) of P which is a complete binary tree

(cf. proof of Lemma 7.10). Since $\chi(t) = \tau$ if and only if $\tau \in 2^X$ for every leaf t, $|\{ X_1(t) : t \text{ is a leaf of } T \}| = 2^{|X|}$. Hence, $\max\{ |X_1(t)| : t \text{ is a leaf of } T \} = |X|$ and we conclude $\text{gs}(BT) \leq |X|$. Consequently, the proposition holds. □

7.4 Backdoor Tree Detection

In this section we pay attention to the detection of backdoor trees. We first define the following decision problem:

$\mathcal{C}$-Backdoor-Tree Detection(GS,Leaves)

Given:	A program P, an integer $g \geq 0$, and an integer $l \geq 0$.
Parameter:	The integer $g + l$.
Task:	Decide whether P has a $\mathcal{C}$-backdoor tree BT of Gallo-Scutellà parameter $\text{gs}(BT) \leq g$ and $\#\text{leaves}(BT) \leq l$.

Similar to the other backdoor detection problems, by self-reduction or self-transformation [Sch81; DF99; DF13], we can use a decision algorithm for $\mathcal{C}$-Backdoor-Tree Detection(GS,Leaves) to actually find the backdoor. Again we only require the target class to be hereditary.

Lemma 7.12. *Let $\mathcal{C}$ be a hereditary class of programs. If $\mathcal{C}$-Backdoor-Tree Detection(GS,Leaves) is fixed-parameter tractable, then also finding a $\mathcal{C}$-backdoor tree of a given program P of Gallo-Scutellà parameter at most g and at most l leaves is fixed-parameter tractable (when parameterized by $g + l$).*

Proof. Given a program P of input size n and integers $g \geq 0$ and $l \geq 0$. We check whether P has a $\mathcal{C}$-backdoor tree of Gallo-Scutellà parameter at most g and at most l leaves by means of Algorithm 7.1. Assume that the decision problem $\mathcal{C}$-Backdoor-Tree Detection(GS,Leaves) is fixed-parameter tractable and runs in time $\mathcal{O}(f(g + l) \cdot n^c)$ for some constant c and some computable function f.

Steps 1 and 2 of Algorithm 7.1 are trivial. In Step 3 we use the decision problem $\mathcal{C}$-Backdoor-Tree Detection(GS,Leaves) to find an atom $a \in \text{at}(P)$ where (i) for the truth assignment τ_1 that assigns $\tau_1(a) = 1$ the program P_{τ_1} has a $\mathcal{C}$-backdoor tree BT_1 of $\text{gs}(BT_1) \leq g - 1$ and $\#\text{leaves}(BT_1) \leq l_1$ and (ii) for the truth assignment τ_0 that assigns $\tau_0(a) = 0$ the program P_{τ_0} has a $\mathcal{C}$-backdoor tree BT_0 of $\text{gs}(BT_0) \leq g$ and $\#\text{leaves}(BT_0) \leq l_0$, and $2 \leq l_1 + l_0 \leq l$ for some integers l_1 and l_0. Such an atom a and a $\mathcal{C}$-backdoor tree $BT = (T, \chi)$ of $\text{gs}(BT) \leq g$ and $\#\text{leaves}(BT) \leq l$ exist since $\mathcal{C}$-Backdoor-Tree Detection(GS,Leaves) returns *Yes* in Step 1. Then, by definition of a binary decision tree there are also trees BT_1 and BT_0 where $\#\text{leaves}(BT_1) + \#\text{leaves}(BT_0) \leq l$. There are only linear many combinations $l_1 + l_0 \leq l$ and we determine the integers l_1 and l_0 simply by means of binary search in Steps 3a–3g. We ensure that $\mathcal{C}$-Backdoor-Tree Detection(GS,Leaves) yields *No* for every value of i and *Yes* for

ALGORITHM 7.1: $\mathcal{C}$-BdTreeComp(P, g, l)

Input: A disjunctive program P, an integer g, an integer l, and a truth assignment τ.
Output: A $\mathcal{C}$-backdoor tree of Gallo-Scutellà parameter at most g and at most l leaves or None if no such $\mathcal{C}$-backdoor tree exists.

1. Return None
   if $\mathcal{C}$-Backdoor-Tree Detection(GS,Leaves) returns *No* for input P, g, and l.
2. Return the backdoor tree (T, χ) where $T = (\{r\}, \emptyset, r)$ and $\chi(r) = \emptyset$ if $P \in \mathcal{C}$.
3. For each atom $a \in \text{at}(P)$ carry out the following steps:
   a) Let $i := 0$, $j := l - 1$, and τ_1 be the truth assignment that assigns $\tau_1(a) = 1$
   b) Decide $\mathcal{C}$-Backdoor-Tree Detection(GS,Leaves) for P_{τ_1} and integers $g - 1$ and j
      i. Proceed with Step 3c if the answer is *Yes*.
      ii. Proceed with the next atom in Step 3 if the answer is *No*.
   c) Let $m := \lfloor \frac{i+j}{2} \rfloor$
   d) Let $l_1 := j$ and proceed with Step 3g if $i = m$ or $j = m$
   e) Decide $\mathcal{C}$-Backdoor-Tree Detection(GS,Leaves) for P_{τ_1} and integers $g - 1$ and m
      i. Let $j := m$ if the answer is *Yes*.
      ii. Let $i := m$ if the answer is *No*.
      iii. Proceed with Step 3c
   f) Let τ_0 be the truth assignment that assigns $\tau_0(a) = 0$.
   g) Decide $\mathcal{C}$-Backdoor-Tree Detection(GS,Leaves) for P_{τ_0}, and integers g and $l - l_1$.
      i. Proceed with Step 4 if the answer is *Yes*.
      ii. Proceed with the next atom in Step 3 if the answer is *No*.
4. Compute BT_1 using $\mathcal{C}$-BdTreeComp$(P_{\tau_1}, g - 1, m)$ and compute BT_0 using $\mathcal{C}$-BdTreeComp$(P_{\tau_0}, g, l - m)$
5. Return $BT = (T, \chi)$ from $BT_1 = (T_1, \chi_1)$ where $T_1 = (N_1, E_1, r_1)$ and $BT_0 = (T_0, \chi_0)$ where $T_0 = (N_0, E_0, r_0)$ as follows:
   - let r be a fresh node, i.e., $r \notin (N_1 \cup N_0)$,
   - $N := N_1 \cup N_0 \cup \{r\}$, $E := E_1 \cup E_0 \cup \{(r, r_1), (r, r_0)\}$, $T := (N, E, r)$, and
   - $\chi(t) := \begin{cases} \chi_1(t) \cup \{a\}, & \text{if } t \in N_1, \\ \chi_0(t) \cup \{\neg a\}, & \text{if } t \in N_0, \end{cases}$

every value of j; by setting i initially to 0 (Step 3a), checking whether $\mathcal{C}$-Backdoor-Tree Detection(GS,Leaves) yields *Yes* for the initial values (Step 3b), and assigning i and j accordingly in Step 3e. In Step 4 we compute the backdoor trees BT_1 and BT_0 for P_{τ_1} and P_{τ_0}. Finally, in Step 5 we merge BT_1 and BT_0 into a solution BT for the input program. Obviously, BT is a $\mathcal{C}$-backdoor tree of P of $\text{gs}(BT) \leq g$ and $\#\text{leaves}(BT) \leq l$.

By assumption Step 1 runs in time $\mathcal{O}(f(g+l)\cdot n^c)$ for some constant c, Step 2 runs in time $\mathcal{O}(n^d)$ for some constant d, since we can check in polynomial time whether $P \in \mathcal{C}$. Step 3 runs in time $\mathcal{O}(n \cdot (\lceil \log_2 l \rceil + 2) \cdot f(g+l) \cdot n^c)$. Hence, Steps 1–3 run in time $\mathcal{O}(f(g+l)\cdot n^c + n^d + f(g+l)\cdot(\lceil \log_2 l \rceil + 2)\cdot n^{c+1}) = \mathcal{O}(n^e \cdot (f(g+l) + f(g+l)\lceil \log_2 l \rceil))$ for some constant e. Since a binary search tree with l leaves has exactly $l-1$ inner nodes, we have to run the recursion in Step 4 for at most $l-2$ times. Thus, the algorithm runs in time $\mathcal{O}(n^e \cdot l \cdot (f(g+l) + f(g+l)\lceil \log_2 l \rceil)) = \mathcal{O}(n^e \cdot f'(g+l))$ for some computable function f'. □

In the following, we consider backdoor tree detection when parameterized by the Gallo-Scutellà parameter and the number of leaves of a backdoor tree. Therefore, we consider notions coined by Samer and Szeider [SS08] in the setting of propositional satisfiability and apply it to answer set programming.

Theorem 7.13. *The problem* **Horn**-Backdoor-Tree Detection(GS,Leaves) *is fixed-parameter tractable.*

Before we can establish the result we introduce the notion of a loss-free kernelization for answer set programming and establish how we can use loss-free kernelizations to solve the backdoor tree detection problem.

Definition 7.14. *Let $\mathcal{C}$ be a class of programs. A* loss-free kernelization *of the problem* Strong $\mathcal{C}$-Backdoor Detection *is a polynomial-time algorithm that given an instance (I, k), either correctly decides that I does not have a strong $\mathcal{C}$-backdoor of size at most k, or computes a set $K \subseteq \mathrm{at}(P)$ such that the following conditions hold:*

1. *$X \subseteq K$ for every minimal strong $\mathcal{C}$-backdoor X of size at most k and*
2. *there is a computable function f such that $|K| \leq f(k)$.*

A loss-free kernelization is a many-to-one reduction that requires in contrast to a kernelization from Chapter 6 strictly stronger conditions. We establish the following proposition about target classes $\mathcal{C}$ that admit loss-free kernelizations.

Proposition 7.15. *Let $\mathcal{C}$ be a class of programs. If the problem* Strong $\mathcal{C}$-Backdoor Detection *has a loss-free kernelization, then the problem* $\mathcal{C}$-Backdoor-Tree Detection(GS,Leaves) *is fixed-parameter tractable.*

Proof. Let $\mathcal{C}$ be a class of programs, P a program, and $g, l > 0$ integers. We consider g as bound for the Gallo-Scutellà parameter and l for the maximum number of leaves. If $l \leq 2$ we can check in polynomial time by definition of a loss-free kernelization whether $P \in \mathcal{C}$. Assume $l \geq 2$. We compute by means of the loss-free kernelization in polynomial-time a set $K \subseteq \mathrm{at}(P)$ (if it exists). If P has a $\mathcal{C}$-backdoor tree $BT = (T, \chi)$ of Gallo-Scutellà

parameter g and at most l leaves, then $\mathrm{at}(BT)$ is a strong $\mathcal{C}$-backdoor, $g \leq \mathrm{at}(BT)$, and $\#\mathrm{leaves}(BT) \leq l - 1$. Since the problem Strong $\mathcal{C}$-Backdoor Detection has a loss-free kernelization, the size of K is bounded by l and the number of binary decision trees T where $\mathrm{at}(BT) \subseteq K$ is bounded by some computable function of l. Finally, we check for at most $f(l) \cdot |K|$ times by testing at most $f(l)$ times whether $P_\tau \in \mathcal{C}$ and $|\tau^{-1}| \leq g$ for the leaves in T and hence determine whether BT is a $\mathcal{C}$-backdoor tree of P of Gallo-Scutellà parameter g and of at most l leaves. Hence the proposition follows. □

The following is a direct consequence of results presented by Samer and Szeider [SS08].

Lemma 7.16. *The problem* Strong **Horn**-Backdoor Detection *has a loss-free kernelization with loss-free kernels of size* $k^2 + k$.

Proof. Let P be a program and k an integer. According to Lemmas 4.1 and 4.3 the set $X \subseteq \mathrm{at}(P)$ is a strong **Horn**-backdoor of P if and only if X is a vertex cover of the negation dependency graph N_p. Thus, we consider kernelizations of Vertex Cover for the input graph N_p. In fact the well-known algorithm by Buss and Goldsmith [BG93] provides a kernelization with kernels of size $k^2 + k$. Buss' kernelization works as follows: Consider instance (N_p, k). Let $X \subseteq V$ be the set of vertices with more than k neighbors. If $X > k$ then output an arbitrary instance $(I', 1) \notin$ Vertex Cover (as $(I, k) \notin$ Vertex Cover). Otherwise, consider the instance (N'_p, k') where N'_p is obtained from N_p by removing the vertices in X and all isolated vertices, and $k' = k - |X|$. Since each vertex in V has at most k neighbors, a vertex cover of N'_p of size k' can cover at most $k \cdot k'$ edges. Hence, there can be at most $k'(k+1) \leq k^2 + k$ vertices. Then if N'_p contains more than $k^2 + k$ vertices return $(I', 1) \notin$ Vertex Cover (as $(I, k) \notin$ Vertex Cover). Otherwise, return (N'_p, k') and the set X. It remains to observe that the algorithm runs in time $\mathcal{O}(k + \|N_p\|)$, and yields a loss-free kernel of size at most $k^2 + k$ as $X' \subseteq N'_P \cup X$ for every minimal strong $\mathcal{C}$-backdoor X' of size at most k. Consequently, the algorithm is a loss-free kernelization and the lemma sustains. □

We are now in position to establish Theorem 7.13.

Proof of Theorem 7.13. It follows directly from Proposition 7.15 and Lemma 7.16. □

Then we can drop the assumption in Theorem 7.5 that the backdoor is given.

Corollary 7.17. *Let* $\mathcal{C} \subseteq$ **Normal** *be an enumerable class. The problems in* $\mathcal{A}sp\mathcal{F}ull$ *are all fixed-parameter tractable when parameterized by* $\mathrm{gs}(BT) + \#\mathrm{leaves}(BT)$ *of a smallest* $\mathrm{gs}(BT) + \#\mathrm{leaves}(BT)$ $\mathcal{C}$*-backdoor tree* BT.

Proof. Let P be a program and k an integer. Since there are only linear many combinations for $k = g + l$, we can use Lemma 7.12 to find a $\mathcal{C}$-backdoor tree BT of smallest $\mathrm{gs}(BT) + \#\mathrm{leaves}(BT)$ where $\mathrm{gs}(BT) \leq g$ and $\#\mathrm{leaves}(BT) \leq l$ or to decide that no such backdoor tree exists. The remainder follows from Theorem 7.5. □

Background and Related Work

Backdoor trees have been introduced by Samer and Szeider [SS08] in the context of propositional satisfiability. Backdoor trees provide a more refined concept of backdoor evaluation and take the interaction of variables that form a backdoor into account. The propositional satisfiability problem can be solved by means of a backdoor trees and is fixed parameter tractable when parameterized by the number of leaves in a backdoor tree. The problems of detecting backdoor trees into 2CNF and Horn formulas are fixed parameter tractable. Gallo and Scutellà [GS88] have considered generalized classes $(\Gamma_1, \Gamma_2, \ldots)$ of propositional Horn formulas by investigating the maximal number of positive literals that have be set to true to obtain a propositional Horn formula. The notion provides a parameter that measures in a certain sense the distance of a propositional formula from being Horn. The problem to decide whether a propositional formula belongs to a class Γ_k (parameter detection) has been shown to be in XP, however it is an open research question whether it is also in FPT.

Contribution and Discussion

We have introduced backdoor trees to answer set programming. The general concepts are similar to the propositional setting. We also take the number of leaves of a backdoor tree into account. However, the minimality check, which is necessary to verify minimality of potential answer set candidates with respect to the Gelfond-Lifschitz reduct, yields again (cf. Section 3.3) an additional hardness factor. Therefore, we parameterize the problem of backdoor tree evaluation by the composed parameter number of leaves of a backdoor tree and maximum number of atoms that are set to true by a corresponding truth assignment in a leaf. The former parameter is crucial to bound the number of potential truth assignment reducts and hence to bound the number of answer set candidates. The latter parameter is crucial to bound the number of atoms in any truth assignment, which we additionally have to consider for the minimality check. Our parameterization raises the question of whether we can drop one parameter from the composed parameter. On the one hand, one could parameterize the evaluation problem just by the number of leaves of the backdoor tree, which yields fixed-parameter tractability, but then the evaluation algorithm does not necessarily yield any speedup in the algorithm since we still have to consider the minimality check where a bound on the number of leaves does not pay off when using our minimality check approach. In other words, the evaluation problem is fixed-parameter tractable when parameterized by the number of leaves of backdoor tree. We obtain a parameter which might be significantly smaller, but the running time of the evaluation algorithm can be significantly worse (exponentially). On the other hand, one could parameterize the evaluation problem just by the Gallo-Scutellà parameter (the maximal number of atoms that we have to set to true in any leaf) of the backdoor tree. Since the Gallo-Scutellà parameter of a backdoor tree can be arbitrarily small compared to the number of leaves of a backdoor tree (and hence the size of a smallest backdoor), we obtain an arbitrarily smaller parameter. However, since our upper bound from Section 7.3

for the number of truth assignment reducts is $(1+n)^g$, where n is the number of atoms of the given program and g the Gallo-Scutellà parameter of the backdoor tree, the number of truth assignment reducts remains non-uniformly bounded. Hence, it remains open whether we obtain fixed-parameter tractability. Moreover, the problem backdoor tree detection when parameterized by the Gallo-Scutellà parameter is only known to be in XP and the question of whether it can be carried out in fixed-parameter tractable time is currently an open research question. Finally, we would like to note that it is unlikely that the problem backdoor tree detection is fixed-parameter tractable for the remaining target classes since we already established W[2]-hardness and co-para-NP-hardness, respectively, for strong backdoor detection in Chapter 4.

CHAPTER 8

Theoretical Comparison of Asp Parameters

In this chapter we compare the size of backdoors into various target class with each other, with recently studied Asp parameters, and with several structural restrictions considered in the literature. Parameterized complexity allows to compare those parameters in terms of their *generality* among each other. Figure 8.1 illustrates the results in terms of a lattice. In the previous chapters we defined and investigated the computational complexity of strong or deletion backdoor detection for backdoors into various target classes. We already mentioned that various related parameters occurred in the literature, even so most parameters have not been considered within the framework of parameterized complexity.

In Section 8.1 we provide some general notation to compare Asp parameters. Since we need various examples to illustrate the definitions of various parameters and to separate parameters from each other, we also define auxiliary programs in the first section. In Section 8.2 we consider Asp parameters that are based on the size of deletion or strong backdoors and establish basic properties. Some parameters that measure in a certain sense the distance of a program from being Horn have been proposed in the literature. We consider those parameters in Section 8.3 and compare them with the size of deletion backdoors into **Horn**. In Section 8.4 we compare Asp parameters that measure in a way the distance of a program from being stratified with deletion and strong backdoors and the **Horn**-related parameters. In Section 8.5 we investigate how a parameter that measures in a sense the tree-likeness (*treewidth*) of a certain graph representation (incidence graph) of a program relates to the other parameters. Subsequently, we consider the treewidth of a more general graph representation (dependency graph) as a parameter in Section 8.6. Following this, we review in Section 8.7 two composed parameters where the length of the longest cycle in a graph representation (positive dependency graph) is considered together with (i) the treewidth of a graph that represents interactions between head

atoms and related body atoms or (ii) the distance of the graph from being acyclic. In Section 8.8 we consider a parameter that measures the distance of a program from being acyclic with respect to bad even cycles. In Section 8.9 we compare the number of loop formulas, which measures in a certain sense the distance of a program from being tight, with the other parameters. Note that loop formulas are a key practical concept of some disjunctive Asp solvers, but the reasoning problems are not fixed-parameter tractable when parameterized in this parameter. In Section 8.10 we finally consider in a way the distance to head-cycle-free programs, which generalize disjunctive tight programs, as an Asp parameter and study its relationship to other Asp parameters.

The domination lattice in Figure 8.1 (see p. 88) provides an overview on the relationship between Asp parameters when restricted to normal programs. This chapter is based on published work [FS15b].

8.1 Parameter Comparison

Let p and q be Asp parameters. We say that p *dominates* q (in symbols $p \preceq q$) if there is a function f such that $p(P) \leq f(q(P))$ holds for all programs P. The parameters p and q are *similar* (in symbols $p \sim q$) if $p \preceq q$ and $q \preceq p$. The parameter p *strictly dominates* q (in symbols $p \prec q$) if $p \preceq q$ but not $q \preceq p$, and p and q are *incomparable* (in symbols $p \bowtie q$) if neither $p \preceq q$ nor $q \preceq p$.

Observation 8.1. *Let p and q be* Asp *parameters and $L \in \mathcal{AspFull}$. If p dominates q and $L[p] \in$ FPT, then also $L[q] \in$ FPT.*

Observation 8.2. *Let p and q be normal* Asp *parameters and $\circ \in \{\preceq, \prec, \bowtie\}$. Then $p \circ q$ if and only if $p^{\uparrow} \circ q^{\uparrow}$.*

Proof. Let p and q be normal Asp parameters. We will show that $p^{\uparrow} \preceq q^{\uparrow}$ iff $p \preceq q$. Since $\prec$ and $\bowtie$ can be defined in terms of $\preceq$, this shows the claimed observation.

Assume $p^{\uparrow} \preceq q^{\uparrow}$. Hence there is a function f such that for all programs P we have $p^{\uparrow}(P) \leq f(q^{\uparrow}(P))$. In particular, if P is normal we have by Definition 5.3 that $p(P) = p^{\uparrow}(P)$ and $q(P) = q^{\uparrow}(P)$, hence $p(P) \leq q(P)$, and so $p \preceq q$.

Now assume $p \preceq q$. Hence there is a function f such that for all normal programs P we have $p(P) \leq f(q(P))$. Let f', f'' be the monotonic functions on non-negative integers defined by $f'(n) = \max_{0 \leq i \leq n} f(n)$ and $f''(n) = f'(n) + n$. Let P be a program and X a minimal deletion **Normal**-backdoor of P. We have $|X| + p(P - X) \leq |X| + f(q(P - X)) \leq |X| + q(P - X) + f'(q(P - X)) \leq |X| + q(P - X) + f'(|X| + q(P - X)) \leq f''(|X| + q(P - X))$. Hence $p^{\uparrow} \preceq q^{\uparrow}$ follows by Definition 5.3. □

Let $\mathcal{C}$ be a class of programs. In the following, we omit $\cdot^{\downarrow}$ (see Definition 5.1) for the parameters $\mathrm{db}_{\mathcal{C}}$ and $\mathrm{sb}_{\mathcal{C}}$ whenever it is clear from the context that we compare $\mathrm{db}_{\mathcal{C}}$ or $\mathrm{sb}_{\mathcal{C}}$ with a normal Asp parameter.

In the following, we define various auxiliary programs which we will use as examples, to separate the parameters from each other and establish incomparability or strictness results.

Example 8.1. Let m and n be some large integers. We define the following programs:

$$
\begin{aligned}
P_1^n &:= \{\, a \leftarrow \neg b_1, \ldots, \neg b_n \,\},\\
P_2^n &:= \{\, a_i \leftarrow \neg b : 1 \leq i \leq n \,\},\\
P_{31}^n &:= \{\, b_i \leftarrow \neg a;\ a \leftarrow \neg b_i : 1 \leq i \leq n \,\},\\
P_{32}^n &:= \{\, b_i \leftarrow a;\ a \leftarrow b_i : 1 \leq i \leq n \,\},\\
P_{33}^n &:= P_{31}^n \cup \{\, a \leftarrow d_1;\ d_i \leftarrow d_{i+1} : 1 \leq i < n \,\} \cup \{\, c_i \leftarrow b_i;\ d_i \leftarrow c_i;\ d_i \leftarrow b_i : 1 \leq i \leq n \,\},\\
P_{34}^n &:= P_{33}^n \cup \{\, d_i \leftarrow \neg b_i : 1 \leq i \leq n \,\},\\
P_{35}^n &:= P_{33}^n \setminus \{\, a \leftarrow \neg b_i;\ b_i \leftarrow \neg a : 1 \leq i \leq n \,\} \cup \{\, a_0 \leftarrow \neg a \,\} \cup \{\, b_i \leftarrow a_0 : 1 \leq i \leq n \,\},\\
P_4^n &:= \{\, c_i \leftarrow \neg a_i;\ c_i \leftarrow b_i;\ b_i \leftarrow \neg a_i;\ a_i \leftarrow e_i;\ e_i \leftarrow d_i;\ d_i \leftarrow a_i : 1 \leq i \leq n \,\},\\
P_{51}^n &:= \{\, b_i \leftarrow \neg a_i;\ a_i \leftarrow \neg b_i : 1 \leq i \leq n \,\},\\
P_{52}^n &:= \{\, b_i \leftarrow a_i;\ a_i \leftarrow \neg b_i : 1 \leq i \leq n \,\},\\
P_{53}^n &:= \{\, b_i \leftarrow a_i;\ a_i \leftarrow b_i : 1 \leq i \leq n \,\},\\
P_{54}^n &:= \{\, b_i \leftarrow a_i;\ c_i \leftarrow b_i;\ a_i \leftarrow c_i : 1 \leq i \leq n \,\},\\
P_6^n &:= \{\, a \leftarrow b_1, \ldots, b_n, c_i : 1 \leq i \leq n \,\},\\
P_7^n &:= \{\, a_j \leftarrow a_i : 1 \leq i < j \leq n \,\},\\
P_8^{m,n} &:= \{\, b \leftarrow a_1, \ldots, a_m \,\} \cup \{\, c_i \leftarrow c_{i+1} : 1 \leq i \leq n \,\} \cup \{\, c_{n+1} \leftarrow c_1 \,\},\\
P_9^n &:= \{\, a_2 \leftarrow \neg a_1;\ a_3 \leftarrow \neg a_2 \,\} \cup \{\, b_i \leftarrow a_3;\ a_1 \leftarrow b_i : 1 \leq i \leq n \,\}, \text{ and}\\
P_{11}^n &:= \{\, a_i \vee b \leftarrow c;\ c \leftarrow b;\ b \leftarrow a_i : 1 \leq i \leq n \,\}.
\end{aligned}
$$

⊣

8.2 Asp Parameters Based on Backdoor Size

Backdoor-based Asp parameters can be related to each other in terms of their underlying target classes. We just need a very weak assumption stated in the following which holds for all target classes considered in the paper. Therefore, we need the following definition: A class $\mathcal{C}$ of programs is *closed under the union of disjoint copies* if for every $P \in \mathcal{C}$ and disjoint copies $P_1, \ldots, P_i$ of P also $P \cup P_1 \cup \ldots \cup P_i \in \mathcal{C}$. We say a program P' is a *disjoint copy* of P if P' is isomorphic to P and $\mathrm{at}(P) \cap \mathrm{at}(P') = \emptyset$.

Proposition 8.3. *Let $\mathcal{C}, \mathcal{C}'$ be classes of programs that are closed under the union of disjoint copies. If $\mathcal{C} \subseteq \mathcal{C}'$, then $\mathrm{db}_{\mathcal{C}'} \preceq \mathrm{db}_{\mathcal{C}}$ and $\mathrm{sb}_{\mathcal{C}'} \preceq \mathrm{sb}_{\mathcal{C}}$, even $\mathrm{db}_{\mathcal{C}'}(P) \leq \mathrm{db}_{\mathcal{C}}(P)$ and $\mathrm{sb}_{\mathcal{C}'}(P) \leq \mathrm{sb}_{\mathcal{C}}(P)$ for every program P. If $\mathcal{C}' \setminus \mathcal{C}$ contains a program with at least one atom, then $\mathcal{C} \subseteq \mathcal{C}'$ implies $\mathrm{db}_{\mathcal{C}'} \prec \mathrm{db}_{\mathcal{C}}$ and $\mathrm{sb}_{\mathcal{C}'} \prec \mathrm{sb}_{\mathcal{C}}$.*

Proof. The first statement is obvious. For the second statement, let $P \in \mathcal{C}' \setminus \mathcal{C}$ with $|at(P)| \geq 1$. We construct the program P^n consisting of n disjoint copies of P and observe that $P^n \in \mathcal{C}'$ but $\mathrm{db}_{\mathcal{C}}(P^n), \mathrm{sb}_{\mathcal{C}}(P^n) \geq n$. □

Hence the relationships between target classes as stated in Observation 4.12 carry over to the corresponding backdoor-based Asp parameters that is, if $\mathcal{C} \subseteq \mathcal{C}'$ then a smallest strong (deletion) $\mathcal{C}'$-backdoor is at most the size of a smallest strong (deletion) $\mathcal{C}$-backdoor.

According to Lemma 3.7 every deletion $\mathcal{C}$-backdoor is a strong $\mathcal{C}$-backdoor if $\mathcal{C}$ is hereditary, hence it also holds for smallest backdoors and we immediately get from the definitions:

Observation 8.4. *If $\mathcal{C}$ is hereditary, then $\mathrm{sb}_{\mathcal{C}}$ dominates $\mathrm{db}_{\mathcal{C}}$.*

According to Lemma 4.1, every strong **Horn**-backdoor of a program is a deletion **Horn**-backdoor and vice versa. Hence we obtain the following statement:

Observation 8.5. $\mathrm{sb}_{\mathbf{Horn}} \sim \mathrm{db}_{\mathbf{Horn}}$.

According to Lemma 9.9, every strong **Normal**-backdoor of a program is a deletion **Normal**-backdoor and vice versa. Hence we obtain the following statement:

Observation 8.6. $\mathrm{sb}_{\mathbf{Normal}} \sim \mathrm{db}_{\mathbf{Normal}}$.

Observation 8.7. *We make the following observations about programs from Example 8.1.*

1. *Consider program P^n_{31} and P^n_{32} and let $P \in \{P^n_{31}, P^n_{32}\}$. Since $P - \{a\}$ is Horn and contains no cycle and no directed cycle, we obtain $\mathrm{db}_{\mathbf{Horn}}(P) \leq 1$, $\mathrm{db}_{\mathbf{no\text{-}C}}(P) \leq 1$, and $\mathrm{db}_{\mathbf{no\text{-}DC}}(P) \leq 1$. According to Observation 8.3, we have $\mathrm{db}_{\mathcal{C}}(P^n_{31}) \leq 1$ and $\mathrm{db}_{\mathcal{C}}(P^n_{32}) \leq 1$ where $\mathcal{C} \in \{\mathbf{Horn}\} \cup \mathcal{Acyc}$.*

2. *Consider program P^n_{33}. Since $P^n_{33} - \{a\}$ is Horn and contains no directed cycle and no bad cycle, we obtain $\mathrm{db}_{\mathbf{Horn}}(P^n_{33}) = 0$, $\mathrm{db}_{\mathbf{no\text{-}DC}}(P^n_{33}) \leq 1$, and $\mathrm{db}_{\mathbf{no\text{-}BC}}(P^n_{33}) \leq 1$. According to Observation 8.3, we have $\mathrm{db}_{\mathcal{C}}(P^n_{33}) \leq 1$ where $\mathcal{C} \in \{\mathbf{Horn}, \mathbf{no\text{-}BC}, \mathbf{no\text{-}BEC}\} \cup \mathcal{D}\text{-}\mathcal{Acyc}$.*

3. *Consider program P^n_{34}. Since $P^n_{34} - \{a\}$ contains no even cycle, $\mathrm{db}_{\mathbf{no\text{-}EC}}(P^n_{34}) \leq 1$.*

4. *Consider program P^n_4. The negation dependency graph of P^n_4 contains $2n$ disjoint paths a_ib_i and a_ic_i. Thus smallest deletion **Horn**-backdoor are of size at least n. P^n_4 contains n disjoint bad cycles, n directed cycles of length at least 3, and n directed even cycles. Hence smallest deletion $\mathcal{C}$-backdoors are of size at least n and thus $\mathrm{db}_{\mathcal{C}}(P^n_4) \geq n$ where $\mathcal{C} \in \{$**Horn**, **no-C**, **no-BC**, **no-DC**, **no-DC2**, **no-EC**, **no-BEC**, **no-DEC**$\}$.*

5. *Consider program* P^n_{51}. *The negation dependency graph of* P^n_{51} *contains* n *disjoint paths and thus* $\mathrm{db}_{\mathbf{Horn}}(P^n_{51}) = n$. P^n_{51} *contains* n *disjoint directed bad even cycles and thus* $\mathrm{db}_{\mathbf{no\text{-}DBEC}}(P^n_{51}) = n$. *According to Observation 8.3, we obtain* $\mathrm{db}_{\mathcal{C}}(P^n_{51}) \geq n$ *where* $\mathcal{C} \in \{\mathbf{Horn}\} \cup \mathcal{Acyc}$.

6. *Consider program* P^n_{52}. *Since* P^n_{52} *contains* n *disjoint directed bad cycles,* $\mathrm{db}_{\mathbf{no\text{-}DBC}}(P^n_{52}) = n$.

7. *Consider program* P^n_{54}. *Since* P^n_{54} *contains* n *disjoint even cycles,* n *disjoint directed cycles of length at least* 3, *and* n *disjoint directed even cycles, we obtain by Observation 8.3* $\mathrm{db}_{\mathcal{C}}(P^n_{54}) \geq n$ *where* $\mathcal{C} \in \{\mathbf{no\text{-}C}, \mathbf{no\text{-}DC}, \mathbf{no\text{-}DC2}, \mathbf{no\text{-}EC}, \mathbf{no\text{-}DEC}\}$.

8. *Consider program* P^n_6. *Since* P^n_6 *is Horn and contains no cycle and no directed cycle,* $\mathrm{db}_{\mathbf{Horn}}(P^n_6) = \mathrm{db}_{\mathbf{no\text{-}C}}(P^n_6) = \mathrm{db}_{\mathbf{no\text{-}DC}}(P^n_6) = 0$. *According to Observation 8.3, we have* $\mathrm{db}_{\mathcal{C}}(P^n_6) = 0$ *where* $\mathcal{C} \in \{\mathbf{Horn}\} \cup \mathcal{Acyc}$.

9. *Consider program* P^n_7. *Since* P^n_7 *is Horn and contains no bad cycle and no directed cycle,* $\mathrm{db}_{\mathbf{Horn}}(P^n_7) = \mathrm{db}_{\mathbf{no\text{-}BC}}(P^n_7) = \mathrm{db}_{\mathbf{no\text{-}DC}}(P^n_7) = 0$. *According to Observation 8.3, we have* $\mathrm{db}_{\mathcal{C}}(P^n_7) = 0$ *where* $\mathcal{C} \in \{\mathbf{Horn}, \mathbf{no\text{-}BC}, \mathbf{no\text{-}BEC}\} \cup \mathcal{D}\text{-}\mathcal{Acyc}$.

10. *Consider program* $P^{m,n}_8$. *Since* $P^{m,n}_8$ *is Horn and* $P^{m,n}_8 - \{c_1\}$ *contains no cycle and no directed cycle, we obtain* $\mathrm{db}_{\mathbf{Horn}}(P^{m,n}_8) = 0$, $\mathrm{db}_{\mathbf{no\text{-}C}}(P^{m,n}_8) \leq 1$, $\mathrm{db}_{\mathbf{no\text{-}DC}}(P^{m,n}_8) \leq 1$. *According to Observation 8.3, we have* $\mathrm{db}_{\mathcal{C}}(P^{m,n}_8) \leq 1$ *where* $\mathcal{C} \in \{\mathbf{Horn}\} \cup \mathcal{Acyc}$.

11. *Consider program* P^n_9. *Since* $P^n_9 - \{a_2\}$ *is Horn and* $P^n_9 - \{a_1\}$ *contains no cycle and no directed cycle, we have* $\mathrm{db}_{\mathbf{Horn}}(P^n_9) \leq 1$, $\mathrm{db}_{\mathbf{no\text{-}C}}(P^n_9) \leq 1$, *and* $\mathrm{db}_{\mathbf{no\text{-}DC}}(P^n_9) \leq 1$. *According to Observation 8.3, we have* $\mathrm{db}_{\mathcal{C}}(P^n_9) \leq 1$ *where* $\mathcal{C} \in \{\mathbf{Horn}\} \cup \mathcal{Acyc}$.

12. *Consider program* P^n_{11} *and let* $X = \{b\}$. *Since* $P^n_{11} - X$ *is normal,* X *is a deletion* **Normal***-backdoor of* P^n_{11}. *Since* $P^n_{11} - X$ *is Horn,* X *is a deletion* **Horn***-backdoor of* P^n_{11} *and* $\mathrm{db}_{\mathbf{Horn}}(P^n_{11} - X) = 1$. *Since* $P^n_{11} - X$ *contains no cycle, no even cycle, and no directed cycle, we have* $\mathrm{db}_{\mathcal{C}}(P^n_{11} - X) = 1$ *where* $\mathcal{C} \in \mathcal{Acyc}$. *Consequently,* $\mathrm{db}_{\mathcal{C}}(P^n_{11} - X) = 1$ *where* $\mathcal{C} \in \{\mathbf{Horn}, \mathbf{Normal}\} \cup \mathcal{Acyc}$.

8.3 Asp Parameters Based on the Distance from Horn

Our backdoor-based Asp parameter $\mathrm{db}_{\mathbf{Horn}}$ can be considered as a parameter that measures the distance of a program from being a Horn program. In the literature some normal Asp parameters have been proposed, that also can be considered as distance measures from Horn. In this section we compare them with $\mathrm{db}_{\mathbf{Horn}}$. Since the Asp parameters considered in the literature are normal, we compare the parameters for normal programs only. However, in view of Observation 8.2 the results also hold for the lifted parameters to disjunctive programs.

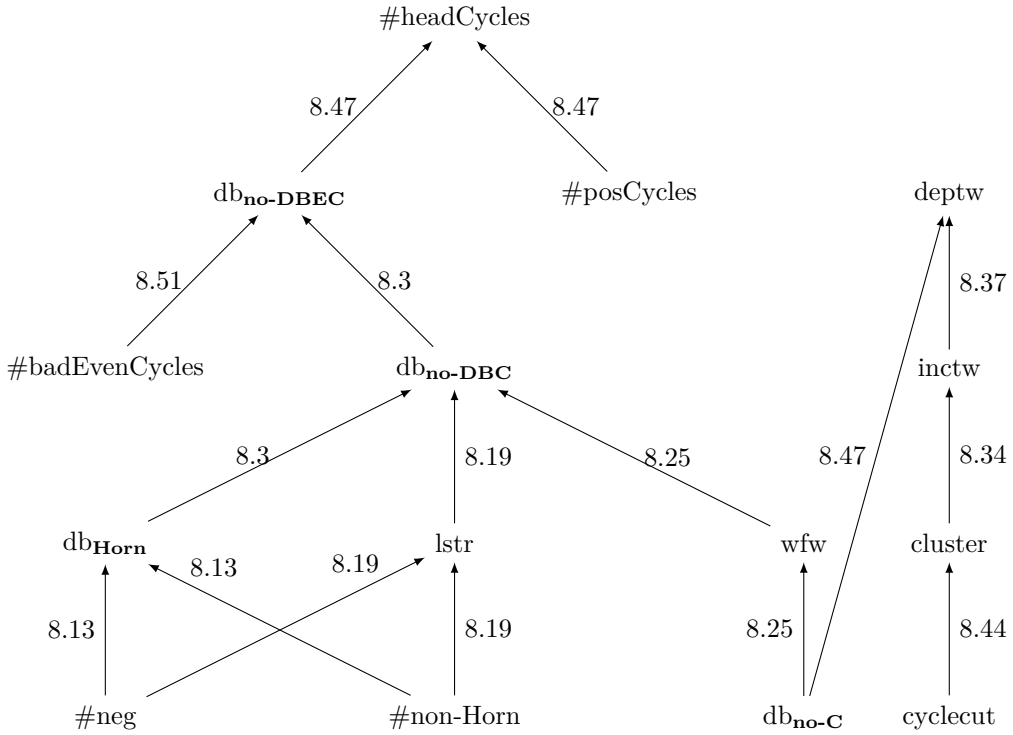


Figure 8.1: Domination Lattice (relationship between Asp parameters when restricted to normal programs). An arrow from p to p' indicates that p' strictly dominates p. Recall that deptw does not yield tractability (Proposition 8.37). When we do not restrict the parameters to normal programs and apply lifting (Observation 8.61) the parameter #headCycles is not strictly more general. A label i of an edge indicates that Proposition i establishes the result.

Definition 8.8 (Ben-Eliyahu [Ben96])**.** *Let P be a normal program. Then*

$$\#\text{neg}(P) := |\{\, a \in \text{at}(P) : a \in B^-(r) \text{ for some rule } r \in P \,\}|,$$
$$\#\text{non-Horn}(P) := |\{\, r \in P : r \text{ is not Horn} \,\}|.$$

Proposition 8.9 (Ben-Eliyahu [Ben96])**.** *For each $L \in \mathcal{AspFull}$, $L[\#\text{neg}]_{\text{N}} \in$ FPT and $L[\#\text{non-Horn}]_{\text{N}} \in$ FPT.*

Since Bound$[p]_{\text{N}}$ for $p \in \{\#\text{neg}, \#\text{non-Horn}\}$ is clearly solvable in polynomial time and thus fixed-parameter tractable, we can use the Lifting Theorem (Theorem 5.5) to obtain the following result.

Corollary 8.10. *For each $L \in \mathcal{AspFull}$, $L[\#\text{neg}^{\uparrow}] \in \text{FPT}$ and $L[\#\text{non-Horn}^{\uparrow}] \in \text{FPT}$.*

Observation 8.11. *We make the following observations about programs from Example 8.1.*

1. *Consider program P_1^n which contains n atoms that occur in $B^-(r)$ for some rule $r \in P$ and exactly one non-Horn rule. Thus, $\#\text{neg}(P_1^n) = n$ and $\#\text{non-Horn}(P_1^n) = 1$.*
2. *Consider program P_2^n which contains only the atom b that occurs in $B^-(r)$ for some rule $r \in P_2^n$ and n non-Horn rules. Thus, $\#\text{neg}(P_2^n) = 1$ and $\#\text{non-Horn}(P_2^n) = n$.*
3. *Consider program P_{31}^n which contains for $1 \leq i \leq n$ the atoms a and b_i that occur in $B^-(r)$ for some rule $r \in P_{31}^n$ and the non-Horn rules $b_i \leftarrow \neg a$ and $a \leftarrow \neg b_i$. Hence, $\#\text{neg}(P_{31}^n) = n + 1$ and $\#\text{non-Horn}(P_{31}^n) = 2n$.*
4. *Consider program P_{32}^n which is Horn. Thus, $\#\text{neg}(P_{32}^n) = \#\text{non-Horn}(P_{32}^n) = 0$.*
5. *Consider program P_{35}^n which contains only the atom a that occurs in $B^-(r)$ for some rule $r \in P_{35}^n$ and exactly one non-Horn rule. So $\#\text{neg}(P_{35}^n) = \#\text{non-Horn}(P_{35}^n) = 1$.*
6. *Consider program P_4^n which contains for $1 \leq i \leq n$ the atoms a_i that occur in $B^-(r)$ for some rule $r \in P_4^n$ and the non-Horn rules $b_i \leftarrow \neg a_i$ and $c_i \leftarrow \neg a_i$. Thus, $\#\text{neg}(P_4^n) = n$ and $\#\text{non-Horn}(P_4^n) = 2n$.*
7. *Consider program P_{51}^n which contains for $1 \leq i \leq n$ the atoms a_i and b_i that occur in $B^-(r)$ for some rule $r \in P$ and the non-Horn rules $b_i \leftarrow \neg a_i$ and $a_i \leftarrow \neg b_i$. Hence, $\#\text{neg}(P_{51}^n) = \#\text{non-Horn}(P_{51}^n) = 2n$.*
8. *Consider the program P_{52}^n which contains the atoms b_i that occur in $B^-(r)$ for some rule $r \in P_{52}^n$ and the non-Horn rules $a_i \leftarrow \neg b_i$. Hence, $\#\text{neg}(P_{52}^n) = \#\text{non-Horn}(P_{52}^n) = n$.*
9. *Consider programs P_{54}^n, P_6^n, P_7^n, and $P_8^{m,n}$ which are Horn. Thus, $\#\text{neg}(P_{54}^n) = \#\text{non-Horn}(P_{54}^n) = \#\text{neg}(P_6^n) = \#\text{non-Horn}(P_6^n) = \#\text{neg}(P_7^n) = \#\text{non-Horn}(P_7^n) = \#\text{neg}(P_8^{m,n}) = \#\text{non-Horn}(P_8^{m,n}) = 0$.*
10. *Consider the program P_9^n which contains only the atoms a_1 and a_2 that occur in $B^-(r)$ for some rule $r \in P_9^n$ and only the non-Horn rules $a_2 \leftarrow \neg a_1$ and $a_3 \leftarrow \neg a_2$. Hence, $\#\text{neg}(P_9^n) = \#\text{non-Horn}(P_9^n) = 2$.*
11. *Consider the program P_{11}^n. The set $X = \{b\}$ is a deletion* **Normal**-*backdoor of P_{11}^n. Since $P_{11}^n - X$ is Horn, we have $\#\text{neg}(P_{11}^n - X) = \#\text{non-Horn}(P_{11}^n - X) = 0$. Thus, $\#\text{neg}^{\uparrow}(P_{11}^n) = |X| + \#\text{neg}(P_{11}^n - X) = 1$ and $\#\text{non-Horn}^{\uparrow}(P_{11}^n) = |X| + \#\text{non-Horn}(P_{11}^n - X) = 1$.*

Proposition 8.12. #neg *and* #non-Horn *are incomparable.*

Proof. The proposition directly follows from considering P_1^n and P_2^n where $\#\text{neg}(P_1^n) = n$ and $\#\text{non-Horn}(P_1^n) = 1$; and $\#\text{neg}(P_2^n) = 1$ and $\#\text{non-Horn}(P_2^n) = n$ by Observation 8.11. □

However, it is easy to see that $\text{db}_{\mathbf{Horn}}$ dominates both parameters.

Proposition 8.13. $\text{db}_{\mathbf{Horn}}$ *strictly dominates* #neg *and* #non-Horn. $\text{db}_{\mathcal{C}}$ *and* #neg*; and* $\text{db}_{\mathcal{C}}$ *and* #non-Horn *are incomparable where* $\mathcal{C} \in \{$**no-C**, **no-DC**, **no-DC2**, **no-EC**, **no-DEC**$\}$.

Proof. For a normal program P define the sets $B^-(P) = \{\, a \in \text{at}(P) : a \in B^-(r) \text{ for some rule } r \in P \,\}$ and $H(P) = \{\, a \in H(r) : r \in P, r \text{ is not Horn} \,\}$. We observe that $B^-(P)$ and $H(P)$ are deletion **Horn**-backdoors of P, hence $\text{db}_{\mathbf{Horn}}(P) \leq \#\text{neg}(P)$ and $\text{db}_{\mathbf{Horn}}(P) \leq \#\text{non-Horn}(P)$. To show that $\text{db}_{\mathbf{Horn}}$ strictly dominates the two parameters, consider P_{31}^n where $\text{db}_{\mathbf{Horn}}(P_{31}^n) \leq 1$, but $\#\text{neg}(P_{31}^n) = n + 1$ and $\#\text{non-Horn}(P_{31}^n) = 2n$ by Observations 8.7 and 8.11.

The second statement follows from considering the programs P_{31}^n and P_{54}^n where $\text{db}_{\mathcal{C}}(P_{31}^n) \leq 1$ and $p(P_{31}^n) \geq n + 1$; and $\text{db}_{\mathcal{C}}(P_{54}^n) \geq n$ and $p(P_{54}^n) = 0$ for $\mathcal{C} \in \{$**no-C**, **no-DC**, **no-DC2**, **no-EC**, **no-DEC**$\}$ and $p \in \{\#\text{neg}, \#\text{non-Horn}\}$ by Observations 8.7 and 8.11. Hence $\text{db}_{\mathcal{C}} \bowtie \#\text{neg}$ and $\text{db}_{\mathcal{C}} \bowtie \#\text{non-Horn}$ for $\mathcal{C} \in \{$**no-C**, **no-DC**, **no-DC2**, **no-EC**, **no-DEC**$\}$. □

8.4 Asp Parameters Based on the Distance from Being Stratified

Ben-Eliyahu [Ben96] and Gottlob, Scarcello, and Sideri [GSS02] have considered Asp parameters that measure in a certain sense how far away a program is from being stratified. In this section we will investigate how these parameters fit into our landscape of Asp parameters. Similar to the last section the parameters have been considered for normal programs only, hence we compare the parameters for normal programs only. Again, in view of Observation 8.2 the results also hold for the lifted parameters to disjunctive programs.

Recall from Section 2.5 that $\text{SCC}(G)$ denotes the partition of the vertex set of a digraph into strongly connected components.

Definition 8.14 (Ben-Eliyahu [Ben96])**.** *Let P be a normal program, D_p its dependency digraph, and $A \subseteq \text{at}(P)$. $P_{/A}$ denotes the program obtained from P by (i) deleting all rules r in the program P where $H(r) \cap A = \emptyset$ and (ii) removing from the bodies of the remaining rules all literals $\neg a$ with $a \notin A$ (this corresponds to the well-known concept of a reduct). Then*

$$\text{lstr}(P) := \sum_{C \in \text{SCC}(D_p)} \min\{\#\text{neg}(P_{/C}), \#\text{non-Horn}(P_{/C})\}.$$

$\mathrm{lstr}(P)$ *is called the* level of stratifiability *of* P.

Proposition 8.15 (Ben-Eliyahu [Ben96])**.** *For each* $L \in \mathcal{AspFull}$, $L[\mathrm{lstr}]_{\mathrm{N}} \in \mathrm{FPT}$.

Since $\textsc{Bound}[\mathrm{lstr}]_{\mathrm{N}}$ is clearly solvable in polynomial time and thus fixed-parameter tractable, we can use the Lifting Theorem (Theorem 5.5) to obtain the following result.

Corollary 8.16. *For each* $L \in \mathcal{AspFull}$, $L[\mathrm{lstr}^{\uparrow}] \in \mathrm{FPT}$.

Observation 8.17. *We make the following observations about programs from Example 8.1.*

1. *Consider program* P_{31}^n *and let* $P = P_{31}^n$. *The partition* $\mathrm{SCC}(D_p)$ *contains only the set* $C = \mathrm{at}(P)$ *and thus* $P_{/C} = P$. *By Observation 8.11* $\#\mathrm{neg}(P) = n + 1$ *and* $\#\text{non-Horn}(P) = 2n$ *and hence* $\mathrm{lstr}(P_{31}^n) = n + 1$.

2. *Consider program* P_{32}^n *and let* $P = P_{32}^n$. *The partition* $\mathrm{SCC}(D_p)$ *contains only the set* $C = \mathrm{at}(P)$ *and* $P_{/C} = P$. *Since* $\#\mathrm{neg}(P) = 0$ *by Observation 8.11, we have* $\mathrm{lstr}(P_{32}^n) = 0$.

3. *Consider program* P_{35}^n *and let* $P = P_{35}^n$. *The partition* $\mathrm{SCC}(D_p)$ *contains only the set* $C = \mathrm{at}(P)$. *Thus,* $P = P_{/C}$. *Since* $\#\mathrm{neg}(P_{35}^n) = 1$ *by Observation 8.11, we conclude* $\mathrm{lstr}(P_{35}^n) \leq 1$.

4. *Consider program* P_4^n *and let* $P = P_4^n$. *We have* $\mathrm{SCC}(D_P)$ *contains exactly the sets* $A_i = \{a_i, e_i, d_i\}$, $B_i = \{b_i\}$, *and* $C_i = \{c_i\}$ *where* $1 \leq i \leq n$. *Hence* $P_{/A_i} = \{\, a_i \leftarrow e_i;\ e_i \leftarrow d_i;\ d_i \leftarrow a_i \,\}$ *and* $P_{/B_i} = \{\, b_i \,\}$ *and* $P_{/C_i} = \{\, c_i;\ c_i \leftarrow b_i \,\}$. *Since* $\#\mathrm{neg}(P_{/C}) = 0$ *for every* $C \in \mathrm{SCC}(D_P)$, *we have* $\mathrm{lstr}(P_4^n) = 0$.

5. *Consider program* P_{51}^n *and* P_{52}^n *and let* $P \in \{P_{51}^n, P_{52}^n\}$. *The partition* $\mathrm{SCC}(D_P)$ *contains exactly the sets* $C_i = \{a_i, b_i\}$ *where* $1 \leq i \leq n$ *and hence* $P_{/C_i} = \{\, b_i \leftarrow \neg a_i;\ a_i \leftarrow \neg b_i \,\}$ *and* $P_{/C_i} = \{b_i \leftarrow a_i;\ a_i \leftarrow \neg b_i : 1 \leq i \leq n\}$, *respectively. Since* $\#\mathrm{neg}(P_{/C_i}) = \#\text{non-Horn}(P_{/C_i}) = 2$, *respectively* $\#\mathrm{neg}(P_{/C_i}) = \#\text{non-Horn}(P_{/C_i}) = 1$, *and there are* n *components we obtain* $\mathrm{lstr}(P_{51}^n) = 2n$ *and* $\mathrm{lstr}(P_{52}^n) = n$.

6. *Consider program* P_6^n *and let* $P = P_6^n$. *The partition* $\mathrm{SCC}(D_p)$ *contains exactly the sets* $A = \{a\}$, $B_i = \{b_i\}$, *and* $C_i = \{c_i\}$ *where* $1 \leq i \leq n$. *Hence* $P_{/A} = \{\, a \leftarrow b_1, \ldots, b_n, c_i : 1 \leq i \leq n \,\}$ *and* $P_{/B_i} = P_{/C_i} = \emptyset$ *where* $1 \leq i \leq n$. *Since* $\#\mathrm{neg}(P_{/C}) = 0$ *for every* $C \in \mathrm{SCC}(D_P)$, *we have* $\mathrm{lstr}(P_6^n) = 0$.

7. *Consider program* P_7^n *and let* $P = P_7^n$. *The partition* $\mathrm{SCC}(D_p)$ *contains exactly the sets* $C_i = \{a_i\}$ *where* $1 \leq i \leq n$. *Thus,* $P_{/C_i} = \{\, a_i \leftarrow a_j : 1 \leq j < i \,\}$. *Hence* $\#\mathrm{neg}(P_{/C_i}) = 0$ *for every* $C \in \mathrm{SCC}(D_P)$. *We obtain* $\mathrm{lstr}(P_7^n) = 0$.

8. *Consider program* $P_8^{m,n}$ *and let* $P = P_8^{m,n}$. *The partition* $\mathrm{SCC}(D_P)$ *contains exactly the sets* $A_i = \{a_i\}$ *where* $1 \leq i \leq m$, $B = \{b\}$, *and* $C = \{\, c_i : 1 \leq i \leq n \,\}$. *Hence* $P_{/A_i} = \emptyset$ *where* $1 \leq i \leq m$, $P_{/B} = \{b \leftarrow a_1, \ldots, a_m\}$, *and* $P_{/C} = \{\, c_i \leftarrow c_{i+1} : 1 \leq i \leq n \,\} \cup \{\, c_{n+1} \leftarrow c_1 \,\}$. *Since* $\#\mathrm{neg}(P_{/A_i}) = 0$ *where* $1 \leq i \leq m$, $\#\mathrm{neg}(P_{/B}) = 0$, *and* $\#\mathrm{neg}(P_{/C}) = 0$, *we obtain* $\mathrm{lstr}(P_8^{m,n}) = 0$.

9. *Consider program* P_9^n *and let* $P = P_9^n$. *The partition* $\mathrm{SCC}(D_P)$ *contains only the set* $C = \mathrm{at}(P)$. *Hence* $P_{/C} = P$. *Since* $\#\mathrm{neg}(P) = \#\text{non-Horn}(P) = 2$, *we have* $\mathrm{lstr}(P_9^n) = 2$.

10. *Consider program* P_{11}^n. *The set* $X = \{b\}$ *is a deletion* **Normal**-*backdoor of* P_{11}^n *by Observation 8.7. We have* $P = P_{11}^n - X = \{\, a_i \leftarrow c;\ c;\ \leftarrow a_i : 1 \leq i \leq n \,\}$. *The partition* $\mathrm{SCC}(D_P)$ *contains the sets* $A_i = \{a_i\}$, *where* $1 \leq i \leq n$, *and* $C = \{c\}$. *Hence* $P_{/A_i} = \{\, a_i \leftarrow c \,\}$, *where* $1 \leq i \leq n$, *and* $P_{/C} = \{\, c \,\}$. *Since* $\#\mathrm{neg}(P_{/C}) = 0$ *for every* $C \in \mathrm{SCC}(D_P)$, *we obtain* $\mathrm{lstr}(P) = 0$. *Consequently,* $\mathrm{lstr}^{\uparrow}(P_{11}^n) = |X| + \mathrm{lstr}(P_{11}^n - X) = 1$.

Observation 8.18. lstr *strictly dominates* #neg *and* #non-Horn.

Proof. Let P be a normal program. We first show that $\sum_{C \in \mathrm{SCC}(D_p)} \#\mathrm{neg}(P_{/C}) \leq \#\mathrm{neg}(P)$. Define the set $B^-(P) = \{\, a \in \mathrm{at}(P) : a \in B^-(r) \text{ for some rule } r \in P \,\}$. By definition $B^-(P_{/A}) \subseteq B^-(P)$ for some $A \subseteq \mathrm{at}(P)$; thus, $\bigcup_{C \in \mathrm{SCC}(D_p)} B^-(P_{/C}) \subseteq B^-(P)$. Let $C, C' \in \mathrm{SCC}(D_p)$ and $C \neq C'$. By definition of a strongly connected component we have $C \cap C' = \emptyset$ and by definition we have that $B^-(P_{/C}) \subseteq C$ and $B^-(P_{/C'}) \subseteq C'$. Hence $B^-(P_{/C}) \cap B^-(P_{/C'}) = \emptyset$. Consequently $\sum_{C \in \mathrm{SCC}(D_p)} \#\mathrm{neg}(P_{/C}) \leq \#\mathrm{neg}(P)$. A similar argument shows that $\sum_{C \in \mathrm{SCC}(D_p)} \#\text{non-Horn}(P_{/C}) \leq \#\text{non-Horn}\ (P)$. Since $\mathrm{lstr}(P) = \sum_{C \in \mathrm{SCC}(D_p)} \min\{\#\mathrm{neg}(P_{/C}), \#\text{non-Horn}(P_{/C})\}$, we have $\mathrm{lstr}(P) \leq \#\mathrm{neg}(P)$ and $\mathrm{lstr}(P) \leq \#\text{non-Horn}(P)$. To show that lstr strictly dominates the two parameters, consider program P_4^n where $\mathrm{lstr}(P_4^n) = 0$, but $\#\mathrm{neg}(P_4^n) \geq n$ and $\#\text{non-Horn}\ (P_4^n) \geq 2n$ by Observations 8.11 and 8.17. Hence the observation is true. □

Proposition 8.19. $\mathrm{db}_{\textbf{no-DBC}}$ *strictly dominates* lstr. *Moreover,* $\mathrm{db}_{\mathcal{C}}$ *and* lstr *are incomparable for the remaining target classes namely* $\mathcal{C} \in \mathcal{A}cyc \setminus \{\textbf{no-DBC}, \textbf{no-DBEC}\} \cup \{\textbf{Horn}\}$.

Proof. We first show that $\mathrm{db}_{\textbf{no-DBC}}$ dominates lstr. For a normal program P define the sets $B^-(P) = \{\, a \in \mathrm{at}(P) : a \in B^-(r) \text{ for some rule } r \in P \,\}$ and $H(P) = \{\, a \in H(r) : r \in P, r \text{ is not Horn} \,\}$. Let $C \in \mathrm{SCC}(D_p)$, we define

$$X_C = \begin{cases} B^-(P_{/C}), & \text{if } |B^-(P_{/C})| \leq |H(P_{/C})|; \\ H(P_{/C}), & \text{otherwise.} \end{cases}$$

and $X = \{\, X_C : C \in \mathrm{SCC}(D_p) \,\}$. We show that X is a deletion **no-DBC**-backdoor of P. By definition for every directed bad cycle $c = (x_1, \ldots, x_l)$ of D_p the atom $x_i \in C'$ where $1 \leq i \leq l$ and $C' \in \mathrm{SCC}(D_p)$ (all vertices of c belong to the same strongly connected

component). Moreover, by definition we have for every negative edge $(x_i, x_j) \in D_p$ of the dependency digraph D_p a corresponding rule $r \in P$ such that $x_j \in H(r)$ and $x_i \in B^-(r)$. Since X_C consists of either $B^-(P_{/C})$ or $H(P_{/C})$, at least one of the atoms x_i, x_j belongs to X_C. Thus, for every directed bad cycle c of the program P at least one atom of the cycle belongs to X. Hence $P - X \in$ **no-DBC** and X is a deletion **no-DBC**-backdoor of P. We obtain $\mathrm{db}_{\textbf{no-DBC}}(P) \leq \mathrm{lstr}(P)$. To show that $\mathrm{db}_{\textbf{no-DBC}}$ strictly dominates lstr, consider program P^n_{31} where $\mathrm{db}_{\textbf{no-DBC}}(P^n_{31}) \leq 1$ and $\mathrm{lstr}(P^n_{31}) = n+1$ by Observations 8.7 and 8.17. Hence $\mathrm{db}_{\textbf{no-DBC}} \prec \mathrm{lstr}$.

Then we show that the parameters $\mathrm{db}_{\mathcal{C}}$ and lstr are incomparable. Consider the programs P^n_3 and P^n_4 where $\mathrm{db}_{\mathcal{C}}(P^n_{31}) \leq 1$ and $\mathrm{lstr}(P^n_{31}) = n + 1$; and $\mathrm{lstr}(P^n_4) = 0$ and $\mathrm{db}_{\mathcal{C}}(P^n_4) \geq n$ for $\mathcal{C} \in \{$**Horn**, **no-C**, **no-BC**, **no-DC**, **no-DC2**, **no-EC**, **no-BEC**, **no-DEC**$\}$ by Observations 8.7 and 8.17.We conclude $\mathrm{db}_{\mathcal{C}} \bowtie \mathrm{lstr}$. □

Definition 8.20 (Gottlob, Scarcello, and Sideri [GSS02])**.** *Let P be a normal program, D_p its dependency digraph, U_p its dependency graph, and $A \subseteq \mathrm{at}(P)$. $\hat{P}_{/A}$ denotes the program obtained from $P_{/A}$ by removing from the bodies of every rule all literals a with $a \notin A$. $\mathrm{at}^+(P)$ denotes the maximal set $W \subseteq \mathrm{at}(P)$ such that there is no bad W-cycle in the dependency graph U_p, in other words the set of all atoms that do not lie on a bad cycle of P. Then*

$$\mathrm{fw}(P) := \min\{\, |S| : S \text{ is a feedback vertex set of } U_p \,\} \text{ and}$$
$$\mathrm{wfw}(P) := \mathrm{fw}(\{\, r \in \hat{P}_{/C} - \mathrm{at}^+(\hat{P}_{/C}) : C \in \mathrm{SCC}(D_p), \hat{P}_{/C} \notin \textbf{no-DBC} \,\}).$$

$\mathrm{fw}(P)$ *is called the* feedback-width *of P, and* $\mathrm{wfw}(P)$ *is called the* weak-feedback-width *of P.*

Observation 8.21. *Let P be a normal program and D_p its dependency digraph. Then* $\mathrm{fw(P)} = \mathrm{db}_{\textbf{no-C}}(P)$ *and hence*

$$\mathrm{wfw(P)} = \mathrm{db}_{\textbf{no-C}}(\{\, r \in \hat{P}_{/C} - \mathrm{at}^+(\hat{P}_{/C}) : C \in \mathrm{SCC}(D_p), \hat{P}_{/C} \notin \textbf{no-DBC} \,\}).$$

Proposition 8.22 (Gottlob, Scarcello, and Sideri [GSS02])**.** *For each $L \in \mathcal{AspFull}$, $L[\mathrm{fw}]_{\mathrm{N}} \in$ FPT and $L[\mathrm{wfw}]_{\mathrm{N}} \in$ FPT.*

Since BOUND$[\mathrm{fw}]_{\mathrm{N}}$ and BOUND$[\mathrm{wfw}]_{\mathrm{N}}$ is fixed-parameter tractable, we can use the Lifting Theorem (Theorem 5.5) to obtain the following result.

Corollary 8.23. *For each $L \in \mathcal{AspFull}$, $L[\mathrm{fw}^{\uparrow}] \in$ FPT and $L[\mathrm{wfw}^{\uparrow}] \in$ FPT.*

Observation 8.24. *We make the following observations about programs from Example 8.1.*

1. *Consider the program P^n_{31} and let $P = P^n_{31}$. The partition $\mathrm{SCC}(D_P)$ contains only the set $C = \mathrm{at}(P)$. For every atom $a \in C$ the program P contains a bad $\{a\}$-cycle and thus $\mathrm{at}^+(\hat{P}_{/C}) = \emptyset$. Consequently, $\hat{P}_{/C} - \mathrm{at}^+(\hat{P}_{/C}) = \hat{P}_{/C} = P$. As*

$P \notin$ **no-DBC**, $\{\, r \in \hat{P}_{/C} - \mathrm{at}^+(\hat{P}_{/C}), C \in \mathrm{SCC}(D_p), \hat{P}_{/C} \notin$ **no-DBC** $\,\} = P$. *We have* $\mathrm{db}_{\textbf{no-C}}(P) = 1$ *by Observation 8.7 and according to Observation 8.21, we obtain* $\mathrm{wfw}(P^n_{31}) = 1$.

2. *Consider program* P^n_{32} *and let* $P = P^n_{32}$. *The partition* $\mathrm{SCC}(D_P)$ *contains only the set* $C = \mathrm{at}(P)$, $\hat{P}_{/C} = P$. *For every atom* $a \in C$ *we have* $\hat{P}_{/C} \in$ **no-DBC** *and thus* $\{\, r \in \hat{P}_{/C} - \mathrm{at}^+(\hat{P}_{/C}) : C \in \mathrm{SCC}(D_p), \hat{P}_{/C} \notin$ **no-DBC** $\,\} = \emptyset$. *Consequently,* $\mathrm{wfw}(P^n_{32}) = 0$.

3. *Consider the programs* P^n_{33}, P^n_{34}, *and* P^n_{35} *and let* $P \in \{P^n_{33}, P^n_{34}, P^n_{35}\}$. *We first observe that the dependency digraph of* P *contains only one strongly connected component. Hence the partition* $\mathrm{SCC}(D_P)$ *contains only the set* $C = \mathrm{at}(P)$. *For every atom* $a \in C$ *program* P *contains a bad* $\{a\}$*-cycle and thus* $\mathrm{at}^+(\hat{P}_{/C}) = \emptyset$. *Consequently,* $\hat{P}_{/C} - \mathrm{at}^+(\hat{P}_{/C}) = \hat{P}_{/C} = P$. *Since* $P \notin$ **no-DBC**, *we obtain* $\{\, r \in \hat{P}_{/C} - \mathrm{at}^+(\hat{P}_{/C}), C \in \mathrm{SCC}(D_p), \hat{P}_{/C} \notin$ **no-DBC** $\,\} = P$. *We have* $\mathrm{db}_{\textbf{no-C}}(P) = n$ *since* P *contains* n *disjoint* $\{b_i\}$*-cycles. According to Observation 8.21, we conclude* $\mathrm{wfw}(P^n_{33}) = \mathrm{wfw}(P^n_{34}) = \mathrm{wfw}(P^n_{35}) = n$.

4. *Consider program* P^n_4 *and let* $P = P^n_4$. *The partition* $\mathrm{SCC}(D_P)$ *contains exactly the sets* $A_i = \{a_i, d_i, e_i\}$, $B_i = \{b_i\}$, *and* $C_i = \{c_i\}$, *for* $1 \leq i \leq n$. *Hence* $\hat{P}_{/A_i} = \{\, a_i \leftarrow e_i;\ e_i \leftarrow d_i;\ d_i \leftarrow a_i \,\}$, $\hat{P}_{/B_i} = \{\, b_i \,\}$ *and* $\hat{P}_{/C_i} = \{\, c_i \,\}$. *For every* $C \in \mathrm{SCC}(D_p)$ *the program* $\hat{P}_{/C} \in$ **no-DBC**. *Consequently,* $\{\, r \in \hat{P}_{/C} - \mathrm{at}^+(\hat{P}_{/C}), C \in \mathrm{SCC}(D_p), \hat{P}_{/C} \notin$ **no-DBC** $\,\} = \emptyset$ *and we obtain* $\mathrm{wfw}(P^n_4) = 0$.

5. *Consider program* P^n_{51} *and let* $P = P^n_{51}$. *The partition* $\mathrm{SCC}(D_P)$ *contains exactly the sets* $C_i = \{a_i, b_i\}$, *for* $1 \leq i \leq n$, *and thus* $\hat{P}_{/C_i} = \{\, a_i \leftarrow \neg b_i;\ b_i \leftarrow \neg a_i \,\}$. *Since* $\mathrm{db}_{\textbf{no-C}}(\hat{P}_{/C_i}) = 1$ *and there are* n *components we obtain* $\mathrm{wfw}(P^n_{51}) = n$.

6. *Consider program* P^n_{52} *and let* $P = P^n_{52}$. *We observe that the partition* $\mathrm{SCC}(D_P)$ *contains exactly the sets* $C_i = \{a_i, b_i\}$. *For every atom* $a \in C_i$ *where* $1 \leq i \leq n$ *there is a bad* $\{a\}$*-cycle in the dependency graph of* $\hat{P}_{/C_i}$ *and thus* $\mathrm{at}^+(\hat{P}_{/C_i}) = \emptyset$. *Consequently,* $\hat{P}_{/C_i} - \mathrm{at}^+(\hat{P}_{/C_i}) = \hat{P}_{/C_i}$. *Since* $\hat{P}_{/C_i} \notin$ **no-DBC**, $\{\, r \in \hat{P}_{/C} - \mathrm{at}^+(\hat{P}_{/C}) : C \in \mathrm{SCC}(D_p), \hat{P}_{/C} \notin$ **no-DBC** $\,\} = P$. *We observe that* $\mathrm{db}_{\textbf{no-C}}(P) = n$ *and according to Observation 8.21, we obtain* $\mathrm{wfw}(P^n_{52}) = n$.

7. *Consider program* P^n_6 *and let* $P = P^n_6$. *The partition* $\mathrm{SCC}(D_p)$ *contains exactly the sets* $A = \{a\}$, $B_i = \{b_i\}$, *and* $C_i = \{c_i\}$ *where* $1 \leq i \leq n$. *Hence* $\hat{P}_{/A} = \{\, a \,\}$ *and* $\hat{P}_{/B_i} = \hat{P}_{/C_i} = \emptyset$ *where* $1 \leq i \leq n$. *Since* $\mathrm{db}_{\textbf{no-C}}(\hat{P}_{/C}) = 0$ *for every* $C \in \mathrm{SCC}(D_P)$, *we obtain* $\mathrm{wfw}(P^n_6) = 0$.

8. *Consider program* P^n_7 *and let* $P = P^n_7$. *Since the partition* $\mathrm{SCC}(D_p)$ *contains exactly the sets* $\{a_i\}$ *where* $1 \leq i \leq n$, $\hat{P}_{/\{a_i\}} = \{\, a_i \,\}$ *and thus* $\mathrm{wfw}(\hat{P}_{/\{a_i\}}) = 0$. *We obtain* $\mathrm{wfw}(P^n_7) = 0$.

9. *Consider program* $P_8^{m,n}$ *and let* $P = P_8^{m,n}$. *The partition* $\mathrm{SCC}(D_P)$ *contains exactly the sets* $A_i = \{a_i\}$ *for* $1 \leq i \leq m$, $B = \{b\}$, *and* $C = \{\, c_i : 1 \leq i \leq n \,\}$. *Hence* $\hat{P}_{/A_i} = \emptyset$ *for* $1 \leq i \leq m$, $\hat{P}_{/B} = \emptyset$, *and* $\hat{P}_{/C} = \{\, c_i \leftarrow c_{i+1} : 1 \leq i \leq n \,\} \cup \{\, c_{n+1} \leftarrow c_1 \,\}$. *The programs* $\hat{P}_{/A_i}$, $\hat{P}_{/B}$, *and* $\hat{P}_{/C}$ *belong to the class* **no-DBC** *for* $1 \leq i \leq m$. *Consequently* $\{\, r \in \hat{P}_{/C} - \mathrm{at}^+(\hat{P}_{/C}) : C \in \mathrm{SCC}(D_p), \hat{P}_{/C} \notin \textbf{no-DBC} \,\} = \emptyset$. *Hence we conclude that* $\mathrm{wfw}(P_8^{m,n}) = 0$.

10. *Consider program* P_9^n *and let* $P = P_9^n$. *The partition* $\mathrm{SCC}(D_P)$ *contains only the set* $C = \mathrm{at}(P)$. *For every atom* $a \in C$ *there is a bad* $\{a\}$*-cycle in the dependency graph of* P *and thus* $\mathrm{at}^+(\hat{P}_{/C}) = \emptyset$. *Consequently,* $\hat{P}_{/C} - \mathrm{at}^+(\hat{P}_{/C}) = \hat{P}_{/C} = P$. *Since* $P \notin$ **no-DBC**, $\{\, r \in \hat{P}_{/C} - \mathrm{at}^+(\hat{P}_{/C}) : C \in \mathrm{SCC}(D_p), \hat{P}_{/C} \notin \textbf{no-DBC} \,\} = P$. *By Observation 8.7* $\mathrm{db}_{\textbf{no-C}}(P) \leq 1$ *and according to Observation 8.21, we obtain* $\mathrm{wfw}(P_9^n) \leq 1$.

11. *Consider program* P_{11}^n *and let* $P = P_{11}^n$. *The set* $X = \{b\}$ *is a deletion* **Normal***-backdoor of* P_{11}^n *by Observation 8.7 and* $P = P_{11}^n - X = \{\, a_i \leftarrow c;\ c;\ \leftarrow a_i : 1 \leq i \leq n \,\}$. *The partition* $\mathrm{SCC}(D_P)$ *contains exactly the sets* $\{a_i\}$ *for* $1 \leq i \leq n$ *and* $\{c\}$. *Hence* $\hat{P}_{/\{a_i\}} = \{\, a_i \,\}$ *for* $1 \leq i \leq n$ *and* $\hat{P}_{/\{c\}} = \{\, c \,\}$. *We observe that* $\mathrm{db}_{\textbf{no-C}}(\hat{P}_{/C}) = 0$ *for every* $C \in \mathrm{SCC}(D_P)$ *and according to Observation 8.21, we obtain* $\mathrm{wfw}(P) = 0$. *Consequently,* $\mathrm{wfw}^{\uparrow}(P_{11}^n) = |X| + \mathrm{wfw}(P_{11}^n - X) = 1$.

In the following proposition we state the relationship between the parameter wfw and our backdoor-based Asp parameters. The first result ($\mathrm{db}_{\textbf{no-DBC}}$ strictly dominates wfw) was anticipated by Gottlob, Scarcello, and Sideri [GSS02].

Proposition 8.25. wfw *strictly dominates* $\mathrm{db}_{\textbf{no-C}}$ *and* $\mathrm{db}_{\textbf{no-DBC}}$ *strictly dominates* wfw. *Moreover,* $\mathrm{db}_{\mathcal{C}}$ *and* wfw *are incomparable for the remaining target classes namely* $\mathcal{C} \in \{$**Horn**, **no-BC**, **no-DC**, **no-DC2**, **no-EC**, **no-BEC**, **no-DEC**$\}$.

Proof. We first show that wfw strictly dominates $\mathrm{db}_{\textbf{no-C}}$. Let P be a normal program and X be a deletion **no-C**-backdoor of P. Define $\hat{P} = \{\, \hat{P}_{/C} - \mathrm{at}^+(\hat{P}_{/C}) : C \in \mathrm{SCC}(D_p), \hat{P}_{/C} \notin \textbf{no-DBC} \,\}$. Since $\hat{P} \subseteq P$ and **no-C** is hereditary (Observation 4.11), $\hat{P} - X \in$ **no-C** and hence X is a deletion **no-C**-backdoor of $\hat{P}$. Consequently, $\mathrm{wfw}(P) \leq \mathrm{db}_{\textbf{no-C}}(\hat{P})$. To show that wfw is strictly more general than $\mathrm{db}_{\textbf{no-C}}$, consider the program P_4^n where $\mathrm{wfw}(P_4^n) = 0$ and $\mathrm{db}_{\textbf{no-C}}(P_4^n) = n$. Hence wfw $\prec$ $\mathrm{db}_{\textbf{no-C}}$ by Observations 8.3 and 8.24.

Next, we show that $\mathrm{db}_{\textbf{no-DBC}}$ strictly dominates wfw. Let P be a normal program and $\hat{P} = \{\, \hat{P}_{/C} - \mathrm{at}^+(\hat{P}_{/C}) : C \in \mathrm{SCC}(D_p), \hat{P}_{/C} \notin \textbf{no-DBC} \,\}$. According to Observation 8.21, $\mathrm{wfw}(P) = \mathrm{db}_{\textbf{no-C}}(\hat{P})$ and thus it is sufficient to show that $\mathrm{db}_{\textbf{no-DBC}}(P) < \mathrm{db}_{\textbf{no-C}}(\hat{P})$. Let X be an arbitrary deletion **no-C**-backdoor of $\hat{P}$. Since **no-C** $\subseteq$ **no-DBC** Observation 8.3 yields that X is also a deletion **no-DBC**-backdoor of $\hat{P}$. Let c be an arbitrary directed bad cycle of D_p. As all vertices of c belong to the same partition $C \in \mathrm{SCC}(D_p)$, $\mathrm{at}(\hat{P}_{/C}) \subseteq C$, and $D_{\hat{p}_{/C}}$ is an induced subdigraph of D_p on $\mathrm{at}(\hat{P}_{/C})$, we obtain c is a directed bad cycle in $D_{\hat{p}_{/C}}$. Since

$\hat{P} = \{\, \hat{P}_{/C} - \mathrm{at}^+(\hat{P}_{/C}) : C \in \mathrm{SCC}(D_p), \hat{P}_{/C} \notin \textbf{no-DBC} \,\}$ and by definition there is no $\mathrm{at}^+(\hat{P}_{/C})$-cycle in U_p, there is no directed bad $\mathrm{at}^+(\hat{P}_{/C})$-cycle in D_p and hence c is also a directed bad cycle in $D_{\hat{p}_{/C}}$. Since X is a deletion **no-DBC**-backdoor of $D_{\hat{p}_{/C}}$ and c is a directed bad X-cycle in $D_{\hat{p}_{/C}}$, X is also a deletion **no-DBC**-backdoor of the program P. Consequently, $\mathrm{db}_{\textbf{no-DBC}}(P) \leq \mathrm{db}_{\textbf{no-C}}(\hat{P}) = \mathrm{wfw}(P)$. To show that $\mathrm{db}_{\textbf{no-DBC}}$ is strictly more general than the parameter wfw, consider the program P^n_{33} where $\mathrm{db}_{\textbf{no-DBC}}(P^n_{33}) \leq 1$ and $\mathrm{wfw}(P^n_{33}) = n$ by Observations 8.7 and 8.24. Hence $\mathrm{db}_{\textbf{no-DBC}} \prec \mathrm{lstr}$.

The third statement follows from considering the programs P^n_{33}, P^n_{34}, and P^n_4 where $\mathrm{db}_{\mathcal{C}}(P^n_{33}) \leq 1$ for $\mathcal{C} \in \{$**Horn**, **no-BC**, **no-DC**, **no-DC2**, **no-BEC**, **no-DEC**$\}$ and $\mathrm{db}_{\textbf{no-EC}}(P^n_{34}) \leq 1$ and $\mathrm{wfw}(P^n_{33}) = \mathrm{wfw}(P^n_{34}) = n$; and $\mathrm{wfw}(P^n_4) = 0$ and $\mathrm{db}_{\mathcal{C}}(P^n_4) = n$ by Observations 8.7 and 8.24. Hence $\mathrm{db}_{\mathcal{C}} \bowtie \mathrm{wfw}$ for $\mathcal{C} \in \{$**Horn**, **no-BC**, **no-DC**, **no-DC2**, **no-EC**, **no-BEC**, **no-DEC**$\}$. □

Observation 8.26. *If $p \in \{\#\mathrm{neg}, \#\text{non-Horn}, \mathrm{lstr}\}$, then p and* wfw *are incomparable.*

Proof. To show that p and wfw are incomparable consider the programs P^n_{31} and P^n_{35} where $p(P^n_{31}) \geq n+1$ and $\mathrm{wfw}(P^n_{31}) = 1$; and $p(P^n_{35}) \leq 1$ and $\mathrm{wfw}(P^n_{35}) = n$ by Observations 8.11, 8.17 and 8.24. □

8.5 Incidence Treewidth

Treewidth is graph parameter introduced by Robertson and Seymour [RS84; RS85; RS86] that measures in a certain sense the tree-likeness of a graph. For further background and examples on treewidth we refer to other sources [Bod93; Bod97; Bod05; GPW10]. Treewidth has been widely applied in knowledge representation, reasoning, and artificial intelligence [Dun07; GPW10; JPW09; MW12; PRW09].

Definition 8.27. *Let $G = (V, E)$ be a graph, $T = (N, E_T)$ a tree, and χ a labeling that maps any node t of T to a subset $\chi(t) \subseteq V$. We call the sets $\chi(\cdot)$* bags *and denote the vertices of T as* nodes*. The pair (T, χ) is a* tree decomposition *of G if the following conditions hold:*

1. *for every vertex $v \in V$ there is a node $t \in N$ such that $v \in \chi(t)$ ("vertices covered");*
2. *for every edge $vw \in E$ there is a node $t \in N$ such that $v, w \in \chi(t)$ ("edges covered"); and*
3. *for any three nodes $t_1, t_2, t_3 \in N$, if t_2 lies on the unique path from t_1 to t_3, then $\chi(t_1) \cap \chi(t_3) \subseteq \chi(t_2)$ ("connectivity").*

The width *of the tree decomposition (T, χ) is $\max\{\, |\chi(t)| - 1 : t \in V(T) \,\}$. The* treewidth *of G, denoted by $\mathrm{tw}(G)$, is the minimum taken over the widths of all possible tree decompositions of G.*

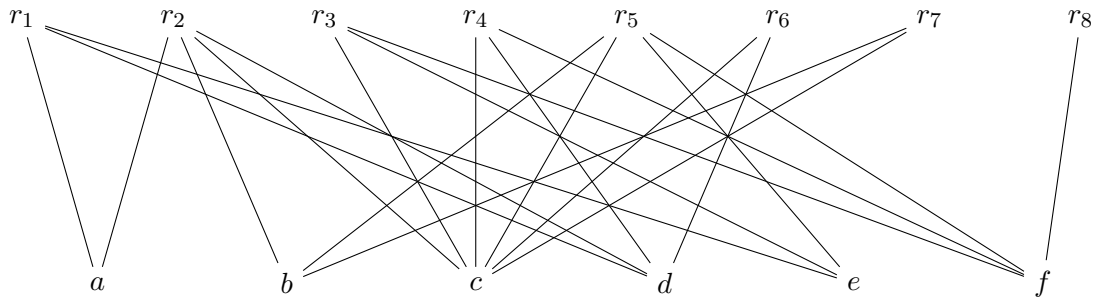


Figure 8.2: Incidence graph I_P of program P from Example 2.1.

We will use the following basic properties of treewidth.

Lemma 8.28 (Folklore, e.g., [RS86])**.** *Let G be a graph and $C_1, \ldots, C_l$ its connected components, then* $\mathrm{tw}(G) = \max\{\, \mathrm{tw}(C_j) : 1 \leq i \leq l \,\}$.

Lemma 8.29 (Folklore, e.g., [BK08])**.** *Let G be a graph. If G has a feedback vertex set size at most k, then* $\mathrm{tw}(G) \leq k + 1$.

Treewidth can be applied to programs by means of various graph representations.

Definition 8.30 (Jakl, Pichler, and Woltran [JPW09])**.** *Let P be a normal program. The* incidence graph *I_P of P is the bipartite graph which has as vertices the atoms and rules of P and where a rule and an atom are joined by an edge if and only if the atom occurs in the rule. Then* $\mathrm{inctw}(P) := \mathrm{tw}(I_P)$. *The parameter* $\mathrm{inctw}(P)$ *is called the* incidence treewidth *of P.*

Figure 8.2 illustrates the incidence graph I_P of program P from Example 2.1.

Proposition 8.31 (Jakl, Pichler, and Woltran [JPW09])**.** *For each $L \in \mathcal{AspFull} \setminus \{\textsc{Enum}\}$, $L[\mathrm{inctw}]_{\mathrm{N}} \in \mathrm{FPT}$ and for $\textsc{Enum}[\mathrm{inctw}]_{\mathrm{N}}$ the solutions can be enumerated with fixed-parameter linear delay between any two consecutive solutions.*

Observation 8.32. *We make the following observations about programs from Example 8.1.*

1. *Consider the programs P^n_{32} and P^n_{51}. We observe that incidence graph of P^n_{32} consists of the cycles a, r_i, b_i, r_{2i} for $1 \leq i \leq n$ and the incidence graph of P^n_{51} consists of the cycles a_i, r_i, b_i, r_{2i} for $1 \leq i \leq n$. According to Lemma 8.29, a cycle has treewidth at most 2 and according to Lemma 8.28, we have* $\mathrm{inctw}(P^n_{32}) \leq 2$ *and* $\mathrm{inctw}(P^n_{51}) \leq 2$.

2. *Consider the programs P^n_6 and P^n_7. Its incidence graph contains a clique on n vertices. Thus, by definition* $\mathrm{inctw}(P^n_6) \geq n - 1$ *and* $\mathrm{inctw}(P^n_7) \geq n - 1$.

3. *Consider program* $P_8^{m,n}$. *The incidence graph consists of a tree on the vertices* $r_0, b, a_1, \ldots, a_m$ *and a cycle* $r_1, c_1, \ldots, r_n, c_n, r_{n+1}, c_{n+1}, r_{n+2}$. *By definition a tree has treewidth* 1, *according to Lemma 8.29, a cycle has treewidth at most* 2, *and according to Lemma 8.28, we obtain* $\mathrm{inctw}(P_8^{m,n}) \leq 2$.

The following observation states why we cannot apply our lifting theorem and extend the parameter treewidth from normal to disjunctive programs.

Observation 8.33. $\textsc{Enum}[\mathrm{inctw}]_{\mathrm{N}} \notin \mathrm{FPT}$.

Proof. Consider the program P_{51}^n where $\mathrm{inctw}(P_{51}^n) \leq 2$. Let $M \subseteq \mathrm{at}(P)$ such that either $a_i \in M$ or $b_i \in M$. According to the definitions, we obtain the GL-reduct $P^M = \{\, a_i : a_i \in M \,\} \cup \{\, b_i : b_i \in M \,\}$. Since M is a minimal model of P^M, M is also an answer set of P. Thus, the program P has 2^n many answer sets. Consequently, enumerating the answer sets of P takes time $\Omega(2^n)$. □

Proposition 8.34. *If* $\mathcal{C} \in \{\textbf{Horn}\} \cup \mathcal{Acyc}$ *and* $p \in \{\mathrm{db}_{\mathcal{C}}, \#\mathrm{neg}, \#\text{non-Horn}, \mathrm{lstr}, \mathrm{wfw}\}$, *then* p *and* inctw *are incomparable.*

Proof. We observe incomparability from the programs P_{51}^n and P_6^n where $p(P_{51}^n) \geq n$ and $\mathrm{inctw}(P_{51}^n) = 2$; and $p(P_6^n) \leq 1$ and $\mathrm{inctw}(P_6^n) \geq n - 1$ by Observations 8.7, 8.11, 8.17, 8.24, and 8.32. □

8.6 Dependency Treewidth

One might ask whether it makes sense to consider restrictions on the treewidth of the dependency graph. In this section we show that the dependency treewidth strictly dominates the incidence treewidth and backdoors into the target class **no-C**, but unfortunately parameterizing the main Asp problems by the dependency treewidth does not yield fixed-parameter tractability.

Definition 8.35. *Let* P *be a program and* U_p *its dependency graph, then* $\mathrm{deptw}(P) := \mathrm{tw}(U_p)$. *We call* $\mathrm{deptw}(P)$ *the* dependency treewidth *of* P.

Observation 8.36. *We make the following observations about programs from Example 8.1.*

1. *Consider programs* P_{32}^n *and* P_6^n *where the dependency graph is a tree. Thus,* $\mathrm{deptw}(P_{32}^n) = \mathrm{deptw}(P_6^n) = 1$.

2. *Consider program* P_{51}^n. *We observe that its dependency graph consists of* n *disjoint cycles* $b_i, v_{b_i,a_i}, a_i, v_{a_i,b_i}$ *for* $1 \leq i \leq n$. *According to Lemma 8.29, a cycle has treewidth at most* 2 *and according to Lemma 8.28, we obtain* $\mathrm{deptw}(P_{51}^n) \leq 2$.

3. *Consider program* P_7^n*. Its dependency graph contains a clique on* n *vertices as a subgraph. Hence* $\mathrm{deptw}(P_7^n) \geq n-1$.

Proposition 8.37. deptw *strictly dominates* inctw *and* $\mathrm{db}_{\textbf{no-C}}$*. Let* $\mathcal{C} \in \{\textbf{Horn}\} \cup \mathcal{Acyc} \setminus \{\textbf{no-C}, \textbf{no-EC}\}$ *and* $p \in \{\mathrm{db}_{\mathcal{C}}, \#\mathrm{neg}, \#\text{non-Horn}, \mathrm{lstr}, \mathrm{wfw}\}$*, then* p *and* deptw *are incomparable.*

Proof. Let P be a normal program, and I_P its incidence graph. Let (T, χ) be an arbitrary tree decomposition of I_P. We create a tree decomposition (T, χ') for U_p as follows: For every $r \in P$ let v_r be the corresponding vertex in I_P. We replace the occurrence of a $v_r \in \chi(t)$ by $H(r)$ for all nodes $t \in V(T)$. Then the pair (T, χ') satisfies Conditions 1 and 2 of a tree decomposition of U_p. Since all edges of I_P are covered in (T, χ) for every $r \in P$ there is a $t \in V(T)$ such that $v_r \in \chi(T)$ and $h \in \chi(T)$ where $H(r) = \{h\}$. Because all v_r are connected in the bags of the tree decomposition (T, χ) and all corresponding elements h are connected in (T, χ), Condition 3 holds for (T, χ'). Thus, (T, χ') is a tree decomposition of the dependency graph U_p. Since the width of (T, χ') is less or equal to the width of (T, χ) it follows $\mathrm{tw}(U_p) \leq \mathrm{tw}(I_P)$ for a normal program P. To show that deptw strictly dominates inctw, consider the program P_6^n where $\mathrm{deptw}(P_6^n) \leq 1$ and $\mathrm{inctw}(P_6^n) \geq n$. Hence deptw $\prec$ inctw.

Let P be a normal program and X a deletion **no-C**-backdoor of P. Thus, X is a feedback vertex set of the dependency graph U_p. According to Lemma 8.29, $\mathrm{tw}(U_p) \leq k+1$. Hence deptw $\preceq \mathrm{db}_{\textbf{no-C}}$. To show that deptw strictly dominates $\mathrm{db}_{\textbf{no-C}}$ consider the program P_{51}^n where $\mathrm{deptw}(P_{51}^n) \leq 2$ and $\mathrm{db}_{\textbf{no-C}}(P_{51}^n) \geq n$. Consequently, deptw $\prec \mathrm{db}_{\textbf{no-C}}$ and the proposition sustains.

To show the last statement, consider again the programs P_{51}^n and P_7^n where $\mathrm{deptw}(P_{51}^n) \leq 2$ and $p(P_{51}^n) \geq n$; and $\mathrm{deptw}(P_7^n) \geq n-1$ and $p(P_7^n) = 0$ by Observations 8.7, 8.17, 8.24, and 8.36. □

Proposition 8.38. *For each* $L \in \mathcal{AspReason}$*,* L_{N} *is* NP*-hard, even for programs that have dependency treewidth* 2*.*

Proof. First consider the problem Consistency. From a 3-CNF formula F with k variables we construct a program P as follows: Among the atoms of our program P will be two atoms a_x and $a_{\bar{x}}$ for each variable $x \in var(F)$ and a new atom f. We add the rules $a_{\bar{x}} \leftarrow \neg a_x$ and $a_x \leftarrow \neg a_{\bar{x}}$ for each variable $x \in var(F)$. For each clause $\{l_1, l_2, l_3\} \in F$ we add the rule $f \leftarrow h(l_1), h(l_2), h(l_3), \neg f$ where $h(\neg x) = a_x$ and $h(x) = a_{\bar{x}}$. Now it is easy to see that the formula F is satisfiable if and only if the program P has an answer set. Let U_p be the undirected dependency graph of P. We construct the following tree decomposition (T, χ) for U_p: the tree T consists of the node t_f and for each $x \in var(F)$ of the nodes t_{fx}, $t_{x\bar{x}}$, and $t_{\bar{x}x}$ along with the edges $t_f t_{fx}$, $t_{fx} t_{x\bar{x}}$, and $t_{x\bar{x}} t_{\bar{x}x}$. We label the nodes by $\chi(t_f) := \{f, v_f\}$ and for each $x \in var(F)$ by $\chi(t_{fx}) := \{a_x, a_{\bar{x}}, f\}$, $\chi(t_{x\bar{x}}) := \{a_x, a_{\bar{x}}, v_{a_x \bar{a}_x}\}$, and $\chi(t_{\bar{x}x}) := \{a_x, a_{\bar{x}}, v_{\bar{a}_x a_x}\}$. We observe that the pair (T, χ) satisfies Condition 1. The rules $a_{\bar{x}} \leftarrow \neg a_x$ and $a_x \leftarrow \neg a_{\bar{x}}$ yield the edges $a_x v_{a_x \bar{a}_x}$, $v_{a_x \bar{a}_x} \bar{a}_x$,

$a_x v_{\bar{a}_x a_x}$, and $v_{\bar{a}_x a_x} \bar{a}_x$ in U_p which are all "covered" by $\chi(t_{x\bar{x}})$ and $\chi(t_{\bar{x}x})$. The rule $f \leftarrow h(l_1), h(l_2), h(l_3), \neg f$ yields the edge $f v_f$ which is covered by $\chi(t_f)$ and yields the edges $f a_x$ or $f a_{\bar{x}}$ which are covered by $\chi(t_{fx})$. Thus, Condition 2 is satisfied. We easily observe that Condition 3 also holds for the pair (T, χ). Hence (T, χ) is a tree decomposition of the dependency graph U_p. Since $\max\{\, |\chi(t)|-1 : t \in V(T) \,\} = 2$, the tree decomposition (T, χ) is of width 2 and $\mathrm{deptw}(P) = 2$. Hence the problem Consistency$[\mathrm{deptw}]_{\mathrm{N}}$ is NP-hard, even for programs that have dependency treewidth 2. We observe hardness for the problems Brave Reasoning and Skeptical Reasoning by the very same argument as in the proof of Theorem 6.4 and the proposition holds. □

8.7 Interaction Treewidth

In this section we consider two parameters investigated by Ben-Eliyahu and Dechter [BD94]: the interaction treewidth introduced under the term "clique width"[1], and the feedback width of the interaction graph introduced under the term "cycle-cutset size". The interaction graph represents "interactions" between head atoms and related body atoms (similar to the Gaifman graph). The interaction treewidth measures in a certain sense the tree-likeness of the interaction graph and the feedback width the distance of the interaction graph from being acyclic. Both parameters are considered together with the length of the longest cycle in the positive dependency digraph (which states dependencies between atoms in the head and atoms in the positive body).

Definition 8.39 (Ben-Eliyahu and Dechter [BD94])**.** *Let P be a normal program. The* interaction graph *is the graph A_P which has as vertices the atoms of P and an edge xy between any two distinct atoms x and y for which there are rules $r, r' \in P$ such that $x \in \mathrm{at}(r)$, $y \in \mathrm{at}(r')$, and $H(r) \cap H(r') \neq \emptyset$.*[2]

Definition 8.40 (Kanchanasut and Stuckey [KS92], Ben-Eliyahu and Dechter [BD94])**.** *Let P be a program. The* positive dependency digraph *D_P^+ of P has as vertices the atoms $\mathrm{at}(P)$ and a directed edge (x, y) between any two atoms $x, y \in \mathrm{at}(P)$ for which there is a rule $r \in P$ with $x \in H(r)$ and $y \in B^+(r)$.*[3]

Let $G = (V, E)$ be a graph and $c = (v_1, \ldots, v_l)$ a cycle of length l in G. A *chord* of c is an edge $v_i v_j \in E$ where v_i and v_j are not connected by an edge in c (non-consecutive vertices). G is *chordal* (triangulated) if every cycle in G of length at least 4 has a chord.

[1] Today the term "clique-width" is predominately used to refer to a different graph parameter, see, e.g., [GP04].

[2] This definition is equivalent to the original definition in earlier work by Ben-Eliyahu and Dechter [BD94] which is given in terms of cliques: the interaction graph is the graph where each atom is associated with a vertex and for every atom a the set of all literals that appear in rules that have a in their heads are connected as a clique.

[3] Ben-Eliyahu and Dechter [BD94] used the term dependency graph while the term positive dependency graph was first used by Kanchanasut and Stuckey [KS92] and became popular by Erdem and Lifschitz [EL03].

Definition 8.41 (Ben-Eliyahu and Dechter [BD94]). *Let G be a digraph and G' a graph. Then*

$$\begin{aligned}\mathrm{lc}(G) &:= \max\{\{2\} \cup \{\, |c| : c \text{ is a cycle in } G \,\}\},\\ \mathrm{cs}(G') &:= \{\, w : G' \text{ is a subgraph of a chordal graph with all cliques of size at most } w \,\},\\ \mathrm{fw}(G') &:= \min\{\, |S| : S \text{ is a feedback vertex set of } G' \,\}.\end{aligned}$$

$\mathrm{lc}(G)$ *is the* length of the longest cycle *of* G. $\mathrm{cs}(G)$ *is called the* clique size.[4] *Let P be a normal program, A_P its interaction graph, and D_p^+ its positive dependency digraph. Then*

$$\begin{aligned}\mathrm{cluster}(P) &:= \mathrm{cs}(A_P) \cdot \log \mathrm{lc}(D_p^+) \text{ and}\\ \mathrm{cyclecut}(P) &:= \mathrm{fw}(A_P) \cdot \log \mathrm{lc}(D_p^+).\end{aligned}$$

$\mathrm{cluster}(P)$ *is called the* size of the tree clustering. $\mathrm{cyclecut}(P)$ *is called the* size of the cycle cutset decomposition.

In fact the definition of $\mathrm{cs}(G)$ is related to the treewidth:

Lemma 8.42 (Robertson and Seymour, 1986). *Let G be a graph. Then*

$$\mathrm{tw}(G) = \mathrm{cs}(G) + 1.$$

Corollary 8.43. *Let P be a normal program, A_P its interaction graph, and D_p^+ its dependency digraph. Then*

$$\mathrm{cluster}(P) = (\mathrm{tw}(A_P) - 1) \cdot \log \mathrm{lc}(D_p^+).$$

Proposition 8.44 (Ben-Eliyahu and Dechter [BD94]). *For each $L \in \mathcal{AspFull}$, $L[\mathrm{cluster}]_{\mathrm{N}} \in \mathrm{FPT}$ and $L[\mathrm{cyclecut}]_{\mathrm{N}} \in \mathrm{FPT}$.*

Observation 8.45. *We make the following observations about programs from Example 8.1.*

1. *Consider programs P_{51}^n and P_{53}^n and let $P \in \{P_{51}^n, P_{53}^n\}$. The interaction graph A_p contains n disjoint paths a_i, b_i, for $1 \leq i \leq m$. Hence A_P contains no cycles and $\mathrm{fw}(A_P) = 0$ and according to Lemma 8.29, we obtain $\mathrm{tw}(A_P) \leq 1$. Moreover, the positive dependency digraph D_P^+ contains no edges, n disjoint cycles of length exactly 2, respectively. Thus, $\mathrm{lc}(D_P^+) = 2$. Consequently, $\mathrm{cluster}(P_{51}^n) \leq 1$ and $\mathrm{cyclecut}(P_{51}^n) \leq 1$; and $\mathrm{cluster}(P_{53}^n) \leq 1$ and $\mathrm{cyclecut}(P_{53}^n) \leq 1$.*
2. *Consider program $P_8^{m,n}$ and let $P = P_8^{m,n}$. The interaction graph A_P contains a clique on m vertices and thus $\mathrm{tw}(A_P) \geq m-1$. According to Lemma 8.42, we obtain $\mathrm{cs}(A_P) \geq m-2$. According to Lemma 8.29, we have $\mathrm{fw}(A_P) \geq m-2$. Moreover, the positive dependency digraph D_P^+ contains the cycle $c_1, c_2, \ldots, c_n, c_{n+1}$. Thus, $\mathrm{lc}(D_P^+) = n$. Consequently, $\mathrm{cluster}(P_8^{m,n}) \geq (m-2) \cdot \log n$ and $\mathrm{cyclecut}(P_8^{m,n}) \geq (m-2) \cdot \log n$.*

[4] The original definition is based on the length of the longest acyclic path in any component of G instead of the length of the longest cycle and the term clique width is used instead of clique size.

Observation 8.46. cluster *strictly dominates* cyclecut.

Proof. Let P be a normal program and A_P its interaction graph. According to Lemma 8.29, we obtain $\mathrm{tw}(A_P) \leq \mathrm{fw}(A_P) + 1$. Hence $\mathrm{cluster}(P) \prec \mathrm{cyclecut}(P)$. □

Proposition 8.47. inctw *strictly dominates* cluster. *If* $\mathcal{C} \in \{\textbf{Horn}\} \cup \mathcal{Acyc}$ *and* $p \in \{\mathrm{db}_{\mathcal{C}}$, #neg, #non-Horn, lstr, wfw$\}$, *then* p *and* cluster *are incomparable; and* p *and* cyclecut *are incomparable.*

Proof. We first show that inctw dominates cluster. Let P be a normal program, I_P its incidence graph, and A_P its interaction graph. Let (T, χ) be an arbitrary tree decomposition of A_P. We create a tree decomposition (T, χ') for I_P as follows: For every $r \in P$ let v_r be the corresponding vertex in I_P. By definition for every $r \in P$ there is a bag $\chi(t)$ where $t \in V(T)$ such that $\mathrm{at}(r) \subset \chi(t)$. We set $\chi'(t) = \chi(t) \cup \{v_r\}$. Then the pair (T, χ') clearly satisfies Conditions 1 and 2 of a tree decomposition of I_P by definition. Since every v_r occurs in exactly one bag Condition 3 holds for (T, χ'). Thus, (T, χ') is a tree decomposition of the interaction graph A_P. Since the width of (T, χ') is less or equal to the width of (T, χ) plus one it follows $\mathrm{tw}(I_P) \leq \mathrm{tw}(A_P) + 1$. To show that inctw strictly dominates cluster, consider the program $P_8^{m,n}$ where $\mathrm{inctw}(P_8^{m,n}) \leq 2$ and $\mathrm{cluster}(P_8^{m,n}) = (m-2)\log n$ by Observations 8.32 and 8.45. Hence inctw $\prec$ cluster.

Let $p \in \{\mathrm{db}_{\mathcal{C}}$, #neg, #non-Horn, lstr, wfw$\}$ and $\mathcal{C} \in \{\textbf{Horn}\} \cup \mathcal{Acyc}$. We show the incomparability of the parameter p and cyclecut. In fact we show something stronger, there are programs P where p is of constant size, but both $\mathrm{tw}(D_P^+)$ and $\mathrm{fw}(D_P^+)$, respectively, and $\mathrm{lc}(I_P)$ can be arbitrarily large, and there are programs where the converse sustains. Therefore we consider the programs P_{51}^n and $P_8^{m,n}$ where $p(P_{51}^n) \geq n$ and $\mathrm{cluster}(P_{51}^n) \leq 1$ and $\mathrm{cyclecut}(P_{51}^n) \leq 1$; and $p(P_8^{m,n}) \leq 1$ and $\mathrm{cyclecut}(P_8^{m,n}) \geq (m-2) \cdot \log n$ and $\mathrm{cluster}(P_8^{m,n}) \geq (m-2) \cdot \log n$ by Observations 8.7, 8.11, 8.17, 8.24, and 8.45. Consequently, the second statement holds. □

8.8 Number of Bad Even Cycles

Lin and Zhao [LZ04a] have considered the number of directed bad even cycles of a given program as a parameter which measures in a certain sense the distance of a program from being acyclic with respect to bad even cycles. This parameter relates to our notion of deletion **no-DEC**-backdoors and deletion **no-DBEC**-backdoors.

Definition 8.48 (Lin and Zhao, 2004)**.** *Let P be a normal program. Then*

$$\#\mathrm{badEvenCycles}(P) := |\{\, c : c \text{ is a directed bad even cycle of } P \,\}|$$

Proposition 8.49. *For each $L \in \mathcal{AspFull}$, $L[\#\mathrm{badEvenCycles}]_{\mathrm{N}} \in \mathrm{FPT}$.*

Observation 8.50. *We make the following observations about programs from Example 8.1.*

1. *Consider program* P_4^n *which contains no directed bad even cycle. Hence* $\#\text{badEvenCycles}(P_4^n) = 0$.

2. *Consider program* P_{51}^n *which contains* n *disjoint directed bad even cycles. Thus,* $\#\text{badEvenCycles}(P_{51}^n) = n$.

3. *Consider programs* P_{52}^n, P_7^n, *and* $P_8^{m,n}$ *which contain no directed bad even cycle. Consequently we obtain* $\#\text{badEvenCycles}(P_{52}^n) = \#\text{badEvenCycles}(P_7^n) = \#\text{badEvenCycles}(P_8^{m,n}) = 0$.

4. *Consider program* P_9^n *which contains the directed bad even cycles* a_1, a_2, a_3, b_i *for* $1 \leq i \leq n$. *Since there are* n *of those directed bad even cycles we obtain* $\#\text{badEvenCycles}(P_9^n) = n$.

Proposition 8.51. $\text{db}_{\textbf{no-DBEC}}$ *strictly dominates* #badEvenCycles. *Moreover,* $\text{db}_{\mathcal{C}}$ *and* #badEvenCycles *are incomparable for the remaining target classes* $\mathcal{C} \in \mathit{Acyc} \setminus \{\textbf{no-DBEC}\} \cup \{\textbf{Horn}\}$. *If* $p \in \{$#neg, #non-Horn, lstr, wfw, inctw, deptw, cluster, cyclecut$\}$, *then* p *and* #badEvenCycles *are incomparable.*

Proof. To see that $\text{db}_{\textbf{no-DBEC}}$ strictly dominates #badEvenCycles. Let P be a normal program. If P has at most k directed bad even cycles, we can construct a deletion **no-DBEC**-backdoor X for P by taking one element from each directed bad even cycle into X. Thus, $\text{db}_{\textbf{no-DBEC}}(P) \leq \#\text{badEvenCycles}(P)$. If a program P has a deletion **no-DBEC**-backdoor of size 1, it can have arbitrarily many even cycles that run through the atom in the backdoor, e.g. program P_9^n where $\text{db}_{\textbf{no-DBEC}}(P_9^n) \leq 1$ and $\#\text{badEvenCycles}(P_9^n) = n$ by Observations 8.7 and 8.50. It follows that $\text{db}_{\textbf{no-DBEC}} \prec$ #badEvenCycles and the proposition holds.

To show the second statement, consider the programs P_4^n, P_{52}^n, and P_9^n where $\text{db}_{\mathcal{C}}(P_9^n) = 1$ for $\mathcal{C} \in \mathit{Acyc} \cup \{\textbf{Horn}\}$ and $\#\text{badEvenCycles}(P_9^n) = n$; conversely $\text{db}_{\mathcal{C}}(P_4^n) \geq n$ for $\mathcal{C} \in \{\textbf{Horn}, \textbf{no-C}, \textbf{no-BC}, \textbf{no-DC}, \textbf{no-DC2}, \textbf{no-EC}, \textbf{no-DEC}, \textbf{no-BEC}\}$, $\text{db}_{\textbf{no-DBC}}(P_{52}^n) \geq n$, and $\#\text{badEvenCycles}(P_4^n) = \#\text{badEvenCycles}(P_{52}^n) = 0$. Hence $\text{db}_{\mathcal{C}} \bowtie$ #badEvenCycles for $\mathcal{C} \in \mathit{Acyc} \setminus \{\textbf{no-DBEC}\} \cup \{\textbf{Horn}\}$ by Observations 8.7 and 8.50.

To show the third statement, consider the programs P_{51}^n, P_{52}^n, P_7^n, and $P_8^{m,n}$, P_9^n where $\text{inctw}(P_7^n) \geq n - 1$ and $\text{deptw}(P_7^n) \geq n - 1$, $p(P_{52}^n) \geq n$ for $p \in \{$#neg, #non-Horn, lstr, wfw$\}$, $\text{cyclecut}(P_8^{m,n}) \geq (m-2)\log n$, $\text{cluster}(P_8^{m,n}) \geq (m-2)\log n$, and $\#\text{badEvenCycles}(P_7^n) = \#\text{badEvenCycles}(P_8^{m,n}) = \#\text{badEvenCycles}(P_{52}^n) = 0$; conversely $p(P_{51}^n) \leq 2$ for $p \in \{$inctw, deptw, cluster, cyclecut$\}$, $p(P_9^n) \leq 2$ for $p \in \{$#neg, #non-Horn, lstr, wfw$\}$, and $\#\text{badEvenCycles}(P_{51}^n) = \#\text{badEvenCycles}(P_9^n) = n$ by Observations 8.7, 8.11, 8.17, 8.24, 8.32, 8.36, 8.45, and 8.50. Hence $p \bowtie$ #badEvenCycles for $p \in \{$#neg, #non-Horn, lstr, wfw, inctw, deptw, cluster, cyclecut$\}$. □

8.9 Number of Positive Cycles (Loop Formulas)

Fages [Fag94], Lin and Zhao [LZ04a], and Lee and Lifschitz [LL03] have introduced transformations of normal programs and disjunctive programs into Sat, respectively. Fages [Fag94] has established the notion of being acyclic with respect to the positive dependency digraph of a given program, so-called tight programs. Lin and Zhao [LZ04a] have extended this to non-tight programs by adding additional formulas that prevent cycles in the positive dependency graph, so-called loop formulas. We would like to mention that loop formulas are used in some Asp solvers, e.g., ClaspD [Dre+08a] and Cmodels3 [Lie05]. The concept of loop formulas is based on the observation that cycles in the positive dependency digraph yield additional models in the Sat formula which are in fact not answer sets and can be eliminated by forbidding a "circular justification" of atoms without having a "justification from outside". The number of loop formulas depends on the number of cycles of the positive dependency digraph and yields the following parameter that measures in a certain sense the distance of a program from being tight.

Definition 8.52 (Fages, 1994)**.** *Let P be a normal program and D_P^+ its positive dependency digraph. Then*

$$\#\text{posCycles} := |\{\, c : c \text{ is a directed cycle in } D_P^+ \,\}|$$

The program P is called tight *if* $\#\text{posCycles} = 0$.[5]

The parameter has been generalized to disjunctive programs by Lee and Lifschitz [LL03].

Proposition 8.53 (Fages, 1994)**.** *For $L \in \mathcal{AspReason}$, $L[\#\text{posCycles}]_{\text{N}}$ is* NP*-hard or* co-NP*-hard, even for tight programs.*

Observation 8.54. *We make the following observations about programs from Example 8.1.*

1. *Consider programs P_{32}^n and P_{53}^n where the positive dependency digraphs contain n directed cycles. Hence $\#\text{posCycles}(P_{32}^n) = \#\text{posCycles}(P_{53}^n) = n$.*
2. *Consider program P_{51}^n and P_7^n where the positive dependency digraphs contain no cycle. Hence $\#\text{posCycles}(P_{51}^n) = \#\text{posCycles}(P_7^n) = 0$.*
3. *Consider program $P_8^{m,n}$. Its positive dependency digraph contains only the cycle c_1, c_2, …, c_n, c_{n+1}; thus, $\#\text{posCycles}(P_8^{m,n}) = 1$.*

Proposition 8.55. *If $\mathcal{C} \in \{\textbf{Horn}\} \cup \mathcal{Acyc}$ and $p \in \{\text{db}_{\mathcal{C}}, \#\text{neg}, \#\text{non-Horn}, \text{lstr}, \text{wfw}, \text{inctw}, \text{deptw}, \text{cluster}, \text{cyclecut}, \#\text{badEvenCycles}\}$, then p and* #posCycles *are incomparable.*

[5] Fages [Fag94] used the term positive-order consistent instead of tight.

Proof. We observe incomparability from the programs P_{32}^n, P_{51}^n, P_{53}^n, P_7^n, and $P_8^{m,n}$. We have $p(P_{51}^n) \geq n$ for $p \in \{\text{db}_\mathcal{C}\ \#\text{neg}, \#\text{non-Horn}, \text{lstr}, \text{wfw}, \#\text{badEvenCycles}\}$, $\text{inctw}(P_7^n) \geq n-1$, $\text{deptw}(P_7^n) \geq n-1$, $\text{cyclecut}(P_8^{m,n}) \geq (m-2) \cdot \log n$, $\text{cluster}(P_8^{m,n}) \geq (m-2) \cdot \log n$, and $\#\text{posCycles}(P_{51}^n) = \#\text{posCycles}(P_7^n) = 0$ as well as $\#\text{posCycles}(P_8^{m,n}) = 1$; conversely for $p \in \{\text{db}_\mathcal{C}, \#\text{neg}, \#\text{non-Horn}, \text{lstr}, \text{wfw}, \text{inctw}, \text{deptw}\}$ we have $p(P_{32}^n) \leq 2$, for $p \in \{\text{cluster}, \text{cyclecut}\}$ we have $p(P_{53}^n) \leq 2$ and $\#\text{posCycles}(P_{32}^n) = \#\text{posCycles}(P_{53}^n) = n$ by Observations 8.7, 8.11, 8.17, 8.24, 8.32, 8.36, 8.45, 8.50, and 8.54. Consequently, the proposition holds. □

8.10 Head-Cycles

Ben-Eliyahu and Dechter [BD94] have considered programs that do not contain certain cycles in their positive dependency digraph, so-called head-cycle-free programs. Head-cycle-free programs can be transformed into normal programs in polynomial time. We would like to mention that connections to head-cycle-free programs are exploited in the implementation of ASP solvers (see e.g., work by Leone, Rullo, and Scarcello [LRS97]). In the following, we consider the number of head cycles as a parameter which then measures in a certain sense the distance of a program from being head-cycle-free.

Definition 8.56 (Ben-Eliyahu and Dechter [BD94])**.** *Let P be a program and D_p^+ its positive dependency digraph. A* head-cycle *of D_p^+ is an $\{x, y\}$-cycle*[6] *where $x, y \in H(r)$ for some rule $r \in P$. The program P is* head-cycle-free *if D_P^+ contains no head-cycle.*

One might consider the number of head-cycles as a parameter to tractability.

Definition 8.57. *Let P be a program and D_p^+ its positive dependency digraph. Then*

$$\#\text{headCycles} := |\{\, c : c \text{ is a head-cycle of } D_p^+ \,\}|.$$

But as the following proposition states that the ASP reasoning problems are already NP-complete for head-cycle-free programs.

Proposition 8.58 (Ben-Eliyahu and Dechter [BD94])**.** *Each $L \in \mathcal{AspReason}$ is* NP*-hard or* co-NP*-hard, even for head-cycle-free programs.*

[6] See Section 4.2.2 for the definition of a W-cycle.

Observation 8.59. *We make the following observations about programs from Example 8.1.*

1. *Consider program P_{51}^n. Since the positive dependency digraph of P_{51}^n contains no cycle,* $\#\text{headCycles}(P_{51}^n) = 0$.
2. *Consider program P_{11}^n. The positive dependency digraph of P_{11}^n contains the head cycles $a_i bc$ for $1 \leq i \leq n$. Thus,* $\#\text{headCycles}(P_{11}^n) = n$.

Even though the parameter #headCycles does not yield tractability for the Asp reasoning problems we are interested in the relationship between our lifted parameters and the parameter #headCycles. We will first restrict the input programs to normal programs in Observation 8.60 and then consider disjunctive programs Observation 8.61.

Observation 8.60. *If $\mathcal{C} \in \{\textbf{Horn}\} \cup \mathcal{Acyc}$ and $p \in \{\text{db}_\mathcal{C}, \#\text{neg}, \#\text{non-Horn}, \text{lstr}, \text{wfw}, \text{inctw}, \text{deptw}, \text{cluster}, \text{cyclecut}, \#\text{badEvenCycles}, \#\text{posCycles}\}$, then* #headCycles *strictly dominates $p^{\downarrow}$.*

Proof. By definition every normal program is head-cycle-free, hence #headCycles strictly dominates p. □

Observation 8.61. *If $\mathcal{C} \in \{\textbf{Horn}\} \cup \mathcal{Acyc}$, then $\text{db}_\mathcal{C}$ and* #headCycles *are incomparable. Moreover, if $p \in \{\#\text{neg}, \#\text{non-Horn}, \text{lstr}, \text{wfw}\}$, then $p^{\uparrow}$ and* #headCycles *are incomparable.*

Proof. To see that the parameters are incomparable consider the programs P_{51}^n and P_{11}^n where $\text{db}_\mathcal{C} \geq n$ as well as $p(P_{51}^n) \geq n$ and $\#\text{headCycles}(P_{51}^n) = 0$; and $p(P_{11}^n) = 1$ and $\#\text{headCycles}(P_{11}^n) = n$ by Observations 8.7, 8.11, 8.17, 8.24, and 8.59. □

Contribution and Discussion

We have studied the size of smallest strong and deletion backdoors into various tractable target classes as Asp parameters and compared them with each other. In that way we obtain certain distance measures from tractable target classes of a program. We have demonstrated that several structural restrictions considered in the literature can be stated in terms of backdoors. As a result we have obtained a unifying approach that accommodates several known restrictions. Moreover, we have studied known Asp parameters like incidence treewidth and fundamental concepts to Asp solving like loop formulas which yield the natural way an Asp parameter. The main contribution of this chapter is a detailed theoretical comparison of the various parameters in terms of their generality and the domination lattice in Figure 8.1 which illustrates the relationship between the various Asp parameters.

CHAPTER 9

Complexity Barrier Breaking Reductions

In this chapter we propose a new exact method for solving the Asp reasoning problems for propositional disjunctive logic programs. Our method exploits the small distance of a disjunctive programs from being normal by means of deletion **Normal**-backdoors. The distance is measured in terms of the size of a smallest "backdoor to normality" which is the smallest number of atoms whose deletion makes the program normal.

Our method proceeds in three phases.

1. We compute a smallest backdoor to normality of the given program. We show that this is fixed-parameter tractable when parameterized by an upper bound k on the backdoor size, using an efficient algorithm for computing a smallest vertex cover of a graph [CKX06].
2. Next we use the backdoor to transform the logic program into a quantified Boolean formula (QBF) where the number of universally quantified variables equals the number k of atoms in the backdoor and the size of the QBF is *quasilinear* in the size of the given logic program, more precisely, the size is $\mathcal{O}(m \cdot \log n)$ where m is the size of the program and n is the total number of atoms. The quasilinearity is achieved by means of the characterization of the least model of a Horn program in terms of level numberings [Jan06].
3. Finally, we eliminate the universal variables using universal expansion [AB02; Bie04], which results in a propositional formula whose size is at most by a factor of 2^k larger than the QBF.

In consequence, the combinatorial explosion, which is expected when transforming a problem from the second level of the Polynomial Hierarchy to the first level, can be

confined to a function of the parameter k and thus utilizes closeness to normality. In that way we provide an fpt-reduction that reduces disjunctive Asp to normal Asp. Our reductions *break complexity barriers* as they move problems form the second to the first level of the Polynomial Hierarchy.

In Chapter 3 we used deletion backdoors into enumerable target classes to make the problem fixed-parameter tractable. In Chapter 5 we used deletion backdoors into **Normal** to lift Asp parameters that are defined for normal programs to disjunctive programs which also yields fixed-parameter tractability. In contrast, the results in this chapter do not provide fixed-parameter tractability of the main Asp reasoning problems, and hence are not directly comparable to the previous results.

In Section 9.1 we provide a special QBF encoding and a Sat encoding for brave and skeptical reasoning. The basis are the concepts and methods of Section 3.3 where we have shown that Strong $\mathcal{C}$-Backdoor Asp Check is fixed-parameter tractable. In Section 9.1.3 we study strong backdoor detection into the target class **Normal**. In Section 9.2 we investigate whether deletion backdoors to tightness allow a similar QBF encoding and Sat encoding for brave and skeptical reasoning. We conclude the chapter with background and related work, a summary of our contribution, and a discussion of the results. This chapter is based on published work [FS13; FS15a].[1]

9.1 Backdoors to Normality

In this section we present a special quasilinear-time algorithm that, given a program P a strong **Normal**-backdoor X of P, and an atom $a \in \mathrm{at}(P)$, produces two QBFs that are quasilinear in the input size, whose satisfiability answers brave or skeptical acceptance of a, respectively. Special about this QBF encoding is that the number of universally quantified variables of the produced QBFs is exactly the size of a strong **Normal**-backdoor X in contrast to the straightforward encoding to QBF (used, e.g., in work by Egly et al. [Egl+00]) where the numbers of universally and existentially quantified variables are exactly the number of input atoms. The number k of universally quantified variables of a QBF is a measure of its hardness: if k is small, then the we can use universal expansion to eliminate these variables and produce an equivalent propositional formula that is by a factor of 2^k larger than the given QBF. A QBF with few universally quantified variables is "easier" than one with an arbitrary number [DS14b].

Theorem 9.1. *Given a disjunctive logic program P, a strong* **Normal**-*backdoor X of P, and an atom $a \in \mathrm{at}(P)$, let $k = |X|$ and $t = \|P\| \log |at(P)|$. We can produce in time $\mathcal{O}(t)$ quantified Boolean formulas $Q_{Brave}(a)$ and $Q_{Skept}(a)$ of size $\mathcal{O}(t)$ with exactly k many universally quantified variables such that*

1. *$Q_{Brave}(a)$ evaluates to true if and only if a is in some answer set of P and*

[1] The fpt-reduction to Sat as proposed in the conference paper [FS13] does not use Qbf-Sat as an intermediate step and does therefore not support the more succinct Qbf-Sat encoding as presented in this thesis and in the new version [FS15a].

2. *$Q_{Skept}(a)$ evaluates to false if and only if a is in all answer sets of P.*

Before we give a proof of this result, we state some direct consequences.

Corollary 9.2. *The following statements are true:*

1. *The problem* Brave Reasoning *is* para-NP*-complete when parameterized by the size of a strong* **Normal***-backdoor, assuming that the backdoor is given as input.*
2. *The problem* Skeptical Reasoning *is* co-para-NP*-complete when parameterized by the size of a strong* **Normal***-backdoor, assuming that the backdoor is given as input.*

Proof. We generate the quantified Boolean formulas Q_{Brave} and Q_{Skept} according to Theorem 9.1 and then eliminate universal quantifiers one after the other by *universal quantifier expansion* [AB02; Bie04]. Eliminating k many universally quantified variables in this manner leads to an existentially quantified formula (i.e., a propositional formula) that is at most by a factor of 2^k larger than the original formula. This provides fpt-reductions from Brave Reasoning to Sat and from Skeptical Reasoning to UnSat, respectively, hence Brave Reasoning is in para-NP when parameterized by the size of a strong **Normal**-backdoor and Skeptical Reasoning is in co-para-NP when parameterized by the size of a strong **Normal**-backdoor. If a parameterized problem is NP-hard when we fix the parameter to a constant, then it is para-NP-hard [FG06, Theorem 2.14]. Now Brave Reasoning when parameterized by the size of a strong **Normal**-backdoor for backdoor size 0 is exactly the Brave Reasoning problem for normal programs, which is NP-complete [BF91; MT91b], hence we conclude that Brave Reasoning is para-NP-complete when parameterized by the size of a strong **Normal**-backdoor. A similar argument shows that Skeptical Reasoning is co-para-NP-complete when parameterized by the size of a strong **Normal**-backdoor. □

9.1.1 Deterministic Approach

An important part of our QBF encoding is a *non-deterministic* implementation of an fpt-algorithm for checking whether a model M of the given program P is also a minimal model of P^M and hence an answer set of P. The special attention to the minimality check is attributed to the fact that using backdoors to solve the reasoning problems ensures minimality with respect to the partial truth assignment reduct but not with respect to the additional backdoor atoms (cf. Definition 3.2 and the following property). In other words, only backdoor atoms may "destroy" minimality. The partial truth assignment reduct in Definition 3.2 makes the evaluation of a partial truth assignment to a program explicit, but is not useful for checking whether a backdoor atom has a justification in a model. Therefore, we need a relaxed notion of Definition 3.2 which is implicitly in the proof of Theorem 3.10.

Definition 9.3 (*Backdoor Reduct*)**.** *Let P be a disjunctive program and $M, X \subseteq \mathrm{at}(P)$. For a set $X_1 \subseteq M \cap X$ we construct a program $P_{X_1 \subseteq X}$ as follows:*

1. *Remove all rules r for which $H(r) \cap X_1 \neq \emptyset$ and*
2. *Replace for all remaining rules r*
   a) *the head $H(r)$ with $H(r) \setminus X$ and*
   b) *the positive body $B^+(r)$ with $B^+(r) \setminus X_1$.*

The underlying idea of Definition 9.3 is to consider backdoor atoms X_1 that have been set to true by a model candidate M and remove only rules if a backdoor atom in X_1 occurs in the head and remove only literals from the rules that do not effect minimality. In that way we can still use the reduct to verify minimality later on.

Recall that by definition we exclude programs with tautological rules. Since X is a strong **Normal**-backdoor of P, it is also a deletion **Normal**-backdoor of P by Lemma 9.9. Hence $P - X$ is normal. Let r be an arbitrarily chosen rule in P. Then there is a corresponding rule $r' \in P - X$ and a corresponding rule $r'' \in P_{X_1 \subseteq X}$. Since we remove in both constructions exactly the same literals from the head of every rule, $H(r') = H(r'')$ holds. Consequently, $P_{X_1 \subseteq X}$ is normal and $P^M_{X_1 \subseteq X}$ is Horn (here $P^M_{X_1 \subseteq X}$ denotes the GL reduct of $P_{X_1 \subseteq X}$ under M).

For any program P' let $\mathrm{Constr}(P')$ denote the set of constrains of P' and $\mathrm{DH}(P') = P' \setminus \mathrm{Constr}(P')$. If P' is Horn, $\mathrm{DH}(P')$ has a least model L and P' has a model if and only if L is a model of $\mathrm{Constr}(P')$ [DG84].

We will use Definition 9.3 and the observations above to construct Algorithm 9.1 for verifying minimality of a model M with respect to a subset $X_1 \subseteq X \cap M$ for a backdoor X. The underlying idea is that we can check minimality effectively for a normal program (since its GL reduct is a Horn program and we can carry out the minimality check in linear time), however the atoms of the backdoor that are set to true by M need special attention. Hence, we check for a subset X_1 of the backdoor X whether there is a least model $L \cup X_1 \subsetneq M$ of $P^M_{X_1 \subseteq X}$. If yes, M cannot be a minimal model of P^M. If not, M is minimal with respect to X_1. The individual conditions of (4) from Algorithm 1 are obtained by systematically excluding the potential causes of M not being minimal. The following lemma states that running the algorithm successfully for each subset of the backdoor X ensures minimality.

Lemma 9.4. *Let X be a strong* **Normal***-backdoor. A model $M \subseteq \mathrm{at}(P)$ of P^M is a minimal model of P^M if and only if the algorithm* MinCheck(P, X_1, M) *returns True for each set $X_1 \subseteq X$.*

Proof. The proof follows from the proof of the second claim of Lemma 3.10. The algorithm there is stated slightly differently to Algorithm 9.1 (MinCheck). Hence, it remains to

ALGORITHM 9.1: MinCheck(P, X_1, M)

Input: A disjunctive program P, a strong **Normal**-backdoor X of P, sets $X_1 \subseteq X$ and $M \subseteq \text{at}(P)$
Output: {*True*, *False*}
Result: Is there no $L \cup X_1 \subsetneq M$ such that $L \cup X_1$ is a model of P^M?

1. Return *True* if X_1 is not a subset of M.
2. Compute the Horn program $P^M_{X_1 \subseteq X}$.
3. Compute the least model L of $\text{DH}(P^M_{X_1 \subseteq X})$.
4. Return *True* if at least one of the following conditions holds:
   (a) L is not a model of $\text{Constr}(P^M_{X_1 \subseteq X})$.
   (b) L is not a subset of $M \setminus X$,
   (c) $L \cup X_1$ is not a proper subset of M,
   (d) $L \cup X_1$ is not a model of P^M.
5. Otherwise return *False*.

observe that both algorithms are equivalent. Condition (1) of Algorithm 9.1 is there stated as part of the second claim. Condition (4a) of Algorithm 9.1 is stated there as "$P^M_{X_1 \subseteq X}$ has no model". This is equivalent since L is not a model of $\text{Constr}(P^M_{X_1 \subseteq X})$ if and only if $P^M_{X_1 \subseteq X}$ no model. Hence, the lemma is established. □

Example 9.1. Again, consider the program R from Example 2.2 and the backdoor $X = \{b, c, h\}$ from Example 3.2. Let $M = \{b, c, g\} \subseteq \text{at}(R)$. Since M satisfies all rules in R, the set M is a model of R. We apply the Algorithm MinCheck for each subset of $\{b, c, h\}$. For $X_1 = \emptyset$ we obtain $R^M_{X_1 \subseteq X} = \{\, a \leftarrow b;\ \leftarrow a;\ \leftarrow e;\ a;\ g;\ \leftarrow \}$. The set $L = \{a, g\}$ is the least model of $\text{DH}(R^M_{X_1 \subseteq X})$. Since Condition (4a) holds, the algorithm returns *True* for X_1. For $X_2 = \{b\}$ we have $R^M_{X_2 \subseteq X} = \{\, a;\ \leftarrow a;\ g;\ \leftarrow \}$ and the least model $L = \{a, g\}$ of $\text{DH}(R^M_{X_2 \subseteq X})$. Since Condition (4a) holds, MinCheck returns *True* for X_2. For $X_3 = \{c\}$ we obtain $R^M_{X_3 \subseteq X} = \{\, a;\ g \,\}$ and the least model $L = \{a, g\}$ of $\text{DH}(R^M_{X_3 \subseteq X})$. Since Condition (4c) holds, the algorithm returns *True* for X_3. For $X_4 = \{b, c\}$ we have $R^M_{X_4 \subseteq X} = \{g\}$. The set $L = \{g\}$ is the least model of $\text{DH}(R^M_{X_4 \subseteq X})$. Since Condition (4c) holds, the algorithm returns *True* for X_4. For all remaining subsets of X the Algorithm MinCheck returns *True* according to Condition (1). Consequently, M is a minimal model of R^M and thus an answer set of R. ⊣

In our QBF encoding we will implement Algorithm 9.1 non-deterministically in contrast to a previous approach in Section 3.2 where a slightly different version of Algorithm 9.1 is used deterministically. The approach there is based on the property stated below Definition 1. By means of the truth assignments τ which we obtain from a strong backdoor X we can construct $2^{|X|}$ simplified programs P_τ. We know that the answer sets of the partial truth assignment reducts P_τ provide "answer set candidates"

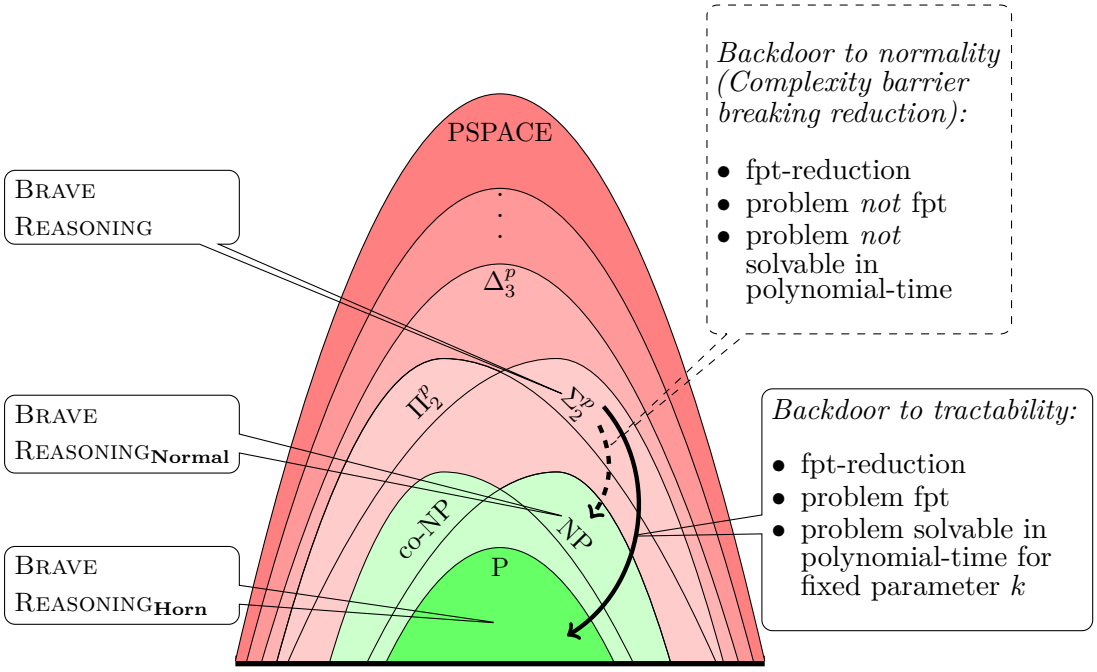


Figure 9.1: Illustration of the Polynomial Hierarchy and the location of the problem Brave Reasoning when disjunctive programs are allowed as input, when the input is restricted to normal programs and when the input is restricted to Horn programs (left). Illustration of the reductions (right) from Brave Reasoning to a computationally easier problem, when we apply backdoor evaluation as introduced in Section 3.3 by means of a backdoor into a tractable target class (backdoor to tractability) and by means of a backdoor into a normal program (backdoor to normality).

of the original program and we can check these candidates in time $\mathcal{O}(f(|X|)\, n^c)$ for some computable function f and a constant c. Such an approach works deterministically: (i) find a backdoor X (ii) apply X to P and obtain $2^{|X|}$ many partial truth assignment reducts $P_{\tau_1}, \ldots, P_{\tau_{2^{|X|}}}$ (iii) find the answer sets in polynomial-time (by assumption we can find all answer sets for each truth assignment reduct P_τ in polynomial time) (iv) check whether a candidate is also an answer set of the original program.

9.1.2 Non-deterministic Approach

Unfortunately, the assumption for (iii) does not hold for the class of normal programs, since a normal program can have exponentially many answer sets. Hence, we cannot simply check the answer sets of P_τ one by one using Algorithm 9.1 deterministically. Furthermore, M is part of the input of Algorithm 9.1 so we need to know it in advance. Therefore, we need a "non-deterministic" construction that does not require to fix M in advance.

In view of Theorem 9.1, the fixed-parameter tractable reduction to Qbf-Sat, and experience in modern Sat and Qbf-Sat solving techniques we obtain a much more optimistic perspective on Asp reasoning problems located on the second level of the Polynomial Hierarchy. Figure 9.1 illustrates this perspective and the difference between the deterministic approach (as established in Chapter 3) and the non-deterministic approach (as established in this chapter) for the problem Brave Reasoning.

We will show that it is possible to implement MinCheck(P, X_1, M) for each set $X_1 \subseteq X$ non-deterministically by means of k universal quantified variables in such a way that we do not need to know M in advance. Possible sets M will be represented by the truth values of certain variables, and since the truth values do not need to be known in advance this will allow us to consider all possible sets M without enumerating them.

To achieve a quasilinear encoding of the computation of the least model of a definite Horn program, we make use of the technique of *level numberings*, which is due to Janhunen [Jan06]. This improves our previous quadratic encoding [FS13], which is based on Dowling and Gallier's algorithm.

Definition 9.5 (Fages [Fag94])**.** *Let P be a program and M a set of atoms. Then M is a* supported model *of P if (i) M is a model of P and (ii) for every $a \in M$ there is a rule $r \in P$ such that $a \in H(r)$ and $B^+(r) \subseteq M$.*

Definition 9.6 (Janhunen [Jan06])**.** *Let P be a definite Horn program and $M \subseteq \mathrm{at}(P)$ a supported model of P. Let $\mathrm{SR}(P, M) = \{\, r \in P : B^+(r) \subseteq M \,\}$. A mapping $\# : M \cup \mathrm{SR}(P, M) \to \mathbb{N}_0$ is a* level numbering with respect to M *if*

1. *For every $r \in P$ with $B^+(r) = \emptyset$ we have $\#(r) = 0$;*
2. *For every $r \in \mathrm{SR}(P, M)$ with $B^+(r) \neq \emptyset$ we have $\#(r) = \max_{b \in B^+(r)} \#(b) + 1$;*
3. *For every $a \in M$ we have $\#(a) = \min_{r \in \mathrm{SR}(P,M), H(r) = \{a\}} \#(r)$;*

The following is a direct consequence of an earlier result by Janhunen [Jan06, The. 55],

Lemma 9.7 (Janhunen [Jan06])**.** *Let P be a definite Horn program and $M \subseteq \mathrm{at}(P)$ a supported model of P. Then M is the least model of P if and only if there exists a level numbering with respect to M.*

Next, we describe the construction of the formulas in detail by defining several formulas which are then put together. To that aim, let us fix a given program P and a strong **Normal**-backdoor X of P.

Among the variables of our formulas will be disjoint sets

$$\begin{aligned} V_m &:= \{\, m[a] : a \in \mathrm{at}(P) \,\}, \\ V_\ell &:= \{\, \ell[a] : a \in \mathrm{at}(P) \,\}, \\ V_x &:= \{\, x[a] : a \in \mathrm{at}(P) \,\}, \text{and} \\ V_s &:= \{\, s[r] : r \in P \,\}. \end{aligned}$$

For a subset $M \subseteq \mathrm{at}(P)$ we let $\tau_M : V_m \to \{0,1\}$ be the truth assignment that sets $m[a]$ to 1 if and only if $a \in M$. Similarly, for a subset $X_1 \subseteq X$ we let $\tau_{X_1} : V_x \to \{0,1\}$ be the truth assignment that sets $x[a]$ to 1 if and only if $a \in X_1$.

Now we can state the main lemma for the proof of Theorem 9.1.

Lemma 9.8. *Given a disjunctive logic program P, a strong* **Normal**-*backdoor X of P, we can construct in polynomial time a propositional formula F^{MinCheck} of size $\mathcal{O}(\|P\| \log |\mathrm{at}(P)|)$ such that for each model M of P and each subset $X_1 \subseteq X$ it holds that the formula $F^{\mathrm{MinCheck}}[\tau_M \cup \tau_{X_1}]$ is satisfiable if and only if Algorithm* MinCheck(P, X_1, M) *returns True.*

Proof. To simplify the notation, we put

$$P' := P^M_{X_1 \subseteq X}.$$

We define the formula F^{MinCheck} as the conjunction of five formulas

$$F^{\mathrm{MinCheck}} := \neg F^{\mathrm{subset}} \vee \big(F^{\mathrm{stays}} \wedge F^{\mathrm{supp}} \wedge F^{\mathrm{level}} \wedge F^{\mathrm{cond}}\big),$$

where F^{subset} enforces Condition (1) of MinCheck; F^{stays} enforces a certain truth assignment to the variables in V_s; F^{supp} enforces that for every satisfying assignment τ of $F^{\mathrm{MinCheck}}[\tau_M \cup \tau_{X_1}]$, the set $L := \{\, a \in \mathrm{at}(P) : \ell[a] \in \tau^{-1}(1) \,\}$ is a supported model of $\mathrm{DH}(P')$; F^{level} enforces that $\mathrm{DH}(P')$ admits a level numbering with respect to L; and F^{cond} enforces that L satisfies Conditions (4a)–(4d) of MinCheck.

We will use propositional constants $T^{\mathrm{condition}}$ that are true if and only if the specified condition holds for the input P and X.

First we define the formula F^{subset} which expresses that "$X_1 \subseteq X \cap M$".

$$F^{subset} := \bigwedge_{a \in X} x[a] \to m[a]$$

Then, we define the formula F^{stays} which expresses that for each $r \in P$ the variable $s[r]$ is true if and only if there is some $r' \in P'$ which has been obtained from r, i.e., r "stays" in P'.

$$F^{\mathrm{stays}} := \bigwedge_{r \in P} \left(s[r] \leftrightarrow \left[\bigwedge_{a \in H(r) \cap X} \neg x[a] \wedge \bigwedge_{a \in B^-(r)} \neg m[a] \right] \right)$$

("$s[r]$ is true if and only if $H(r) \cap X_1 = \emptyset$ and $M \cap B^-(r) = \emptyset$").

We note that $r' \in P'$ is a constraint if and only if $H(r) \setminus X = \emptyset$.

For convenience, we also define the formulas

$$\begin{aligned}
F_r^{\text{stays-pos}} &:= s[r] \wedge T^{H(r)\setminus X \neq \emptyset} && (\text{“}r' \in \mathrm{DH}(P')\text{”}),\\
F_r^{\text{stays-const}} &:= s[r] \wedge T^{H(r)\setminus X = \emptyset} && (\text{“}r' \in \mathrm{Constr}(P')\text{”}).\\
F_{r,a}^{\text{stays-head}} &:= s[r] \wedge T^{a \in H(r)\setminus X} && (\text{“}a \in H(r')\text{”}),\\
F_{r,b}^{\text{stays-body}} &:= s[r] \wedge T^{b \in B^+(r)} \wedge \neg x[b] && (\text{“}b \in B^+(r')\text{”}),\\
F_r^{\text{SR}} &:= s[r] \wedge \bigwedge_{b \in B^+(r)} (F_{r,b}^{\text{stays-body}} \rightarrow \ell[b]) && (\text{“}r' \in \mathrm{DH}(P'),\ B^+(r') \subseteq L\text{”}).
\end{aligned}$$

We observe that $\|F^{\text{stays}}\| \in \mathcal{O}(\|P\|)$, $\|F_r^{\text{stays-pos}}\|$, $\|F_r^{\text{stays-const}}\|$, $\|F_{r,a}^{\text{stays-head}}\|$, $\|F_{r,b}^{\text{stays-body}}\| \in \mathcal{O}(1)$, and $\|F_r^{\text{SR}}\| \in \mathcal{O}(\|r\|)$, and that all these formulas can be constructed in a time that is linear in their respective size.

Next we define the formula F^{supp} which expresses that L is a supported model of $\mathrm{DH}(P')$. A straightforward encoding would yield the following term

$$\bigwedge_{r \in P} \left[\left(F_r^{\text{SR}} \rightarrow \bigvee_{a \in H(r)} \left[F_{r,a}^{\text{stays-head}} \wedge \ell[a] \right] \right) \wedge \bigwedge_{a \in \mathrm{at}(P)} \left(\ell[a] \rightarrow \bigvee_{r \in P, a \in H(r)} \left[F_{r,a}^{\text{stays-head}} \wedge \bigwedge_{b \in B^+(r)} F_{r,b}^{\text{stays-body}} \wedge \ell[b] \right] \right) \right],$$

which is not linear in the size of the program. However, it is easy to observe that the second conjunctive term is logically equivalent to

$$\bigwedge_{a \in \mathrm{at}(P)} \left(\ell[a] \rightarrow \bigvee_{r \in P, a \in H(r)\setminus X} \left[F_{r,a}^{\text{stays-head}} \wedge \bigwedge_{b \in B^+(r)} F_{r,b}^{\text{stays-body}} \wedge \ell[b] \right] \right).$$

Thus, we define

$$F^{\text{supp}} := \bigwedge_{r \in P} \left[\left(F_r^{\text{SR}} \rightarrow \bigvee_{a \in H(r)} \left[F_{r,a}^{\text{stays-head}} \wedge \ell[a] \right] \right) \wedge \bigwedge_{a \in \mathrm{at}(P)} \left(\ell[a] \rightarrow \bigvee_{r \in P, a \in H(r)\setminus X} \left[s[r] \wedge \bigwedge_{b \in B^+(r)} F_{r,b}^{\text{stays-body}} \wedge \ell[b] \right] \right) \right].$$

The first conjunctive term of F^{supp} is clearly of size $\mathcal{O}(\|P\|)$. Since $P - X$ is normal, it follows that for each $r \in P$ there is at most one atom $a \in \mathrm{at}(P)$ such that $a \in H(r) \setminus X$.

Hence, the size of the second conjunctive term of F^{supp} is $\mathcal{O}(\|P\|)$ as well, and so $\|F^{\text{supp}}\| \in \mathcal{O}(\|P\|)$. This is also an upper bound for the time required to construct F^{supp}.

Next we define

$$F^{\text{level}} := F^{+} \wedge F_1^{\text{level}} \wedge F_2^{\text{level}} \wedge F_3^{\text{level}},$$

we will specify F^{+} and F_i^{level} below.

We denote a sequence of $l := \lceil \log_2 |\text{at}(P)| \rceil$ variables $x_1, \ldots, x_l$ by $\vec{x}$. The truth values of the variables $\vec{x}$ encode in binary the numbers between 0 and $|\text{at}(P)| - 1$. For a truth assignment τ to $\vec{x}$ let $n_\tau(\vec{x})$) denote the corresponding number. One can now define the following auxiliary formulas of size $\mathcal{O}(\log |\text{at}(P)|)$ (see, e.g., [EF99]).

$$\begin{aligned} F^{+1}(\vec{x},\vec{y}) \quad & (\text{“}n_\tau(\vec{x}) = n_\tau(\vec{y}) + 1\text{”}) \\ F^{\leq}(\vec{x},\vec{y}) \quad & (\text{“}n_\tau(\vec{x}) \leq n_\tau(\vec{y})\text{”}) \\ F^{=0}(\vec{x}) \quad & (\text{“}n_\tau(\vec{x}) = 0). \end{aligned}$$

In the construction of these formulas we can use some auxiliary variables that do not occur outside these formulas.

In addition to the variables in $V_l \cup V_m \cup V_x$, the following formulas will contain for every atom $a \in \text{at}(P)$ the variables $\vec{a}$ and for every rule $r \in P$ the variables $\vec{r}$ and $\vec{r}_+$.

The first formula ensures that $n_\tau(\vec{r}_+) = n_\tau(\vec{r}) + 1$ holds for all $r' \in \text{SR}(\text{DH}(P'), L)$.

$$F^{+} := \bigwedge_{r \in P} \Big(F_r^{\text{SR}} \rightarrow F^{+1}(\vec{r}_+, \vec{r}) \Big).$$

The next three formulas correspond to the respective parts of Definition 9.6 and ensure that setting $\#(r') = n_\tau(r)$ and $\#(a) = n_\tau(a)$ defines a level numbering of $\text{DH}(P')$ with respect to L.

$$F_1^{\text{level}} := \bigwedge_{r \in P} \Big[\Big(F_r^{\text{stays-pos}} \wedge \bigwedge_{b \in B^+(r)} \neg F_{r,b}^{\text{stays-body}} \Big) \rightarrow F_0^{=}(\vec{r}) \Big],$$

“For every $r' \in \text{DH}(P')$ with $B^+(r') = \emptyset$ we have $\#(r') = 0$;”

$$\begin{aligned} F_2^{\text{level}} := \bigwedge_{r \in P} \Big[\Big(F_r^{\text{SR}} \wedge \bigvee_{b \in B^+(r)} F_{r,b}^{\text{stays-body}} \Big) \rightarrow \\ \Big[\Big(\bigwedge_{b \in B^+(r)} F_{r,b}^{\text{stays-body}} \rightarrow F^{\leq}(\vec{b}, \vec{r}_+) \Big) \wedge \\ \Big(\bigvee_{b \in B^+(r)} F_{r,b}^{\text{stays-body}} \wedge F^{\leq}(\vec{r}_+, \vec{b}) \Big) \Big] \Big], \end{aligned}$$

"For every $r' \in \mathrm{SR}(\mathrm{DH}(P'), L)$ with $B^+(r') \neq \emptyset$ we have $\#(r') = \max_{b \in B^+(r')} \#(b) + 1$;"

$$F_3^{\text{level}} := \bigwedge_{a \in \mathrm{at}(P)} \ell[a] \rightarrow \Big[\Big(\bigwedge_{r \in P} F_r^{\text{stays-pos}} \wedge F_{r,a}^{\text{stays-head}} \rightarrow F^{\leq}(\vec{a}, \vec{r})\Big) \wedge \Big(\bigvee_{r \in P} F_r^{\text{stays-pos}} \wedge F_{r,a}^{\text{stays-head}} \wedge F^{\leq}(\vec{r}, \vec{a})\Big)\Big]$$

"For every $a \in L$ we have $\#(a) = \min_{r' \in \mathrm{SR}(\mathrm{DH}(P'), L), H(r') = \{a\}} \#(r')$;"

Finally we define

$$F^{\text{cond}} := F^{(\mathrm{a})} \vee F^{(\mathrm{b})} \vee F^{(\mathrm{c})} \vee F^{(\mathrm{d})}$$

where the auxiliary formulas check whether at least one of the respective Condition (4a)–(4d) of Algorithm MinCheck(P, X_1, M) holds for L:

$$F^{(\mathrm{a})} := \bigvee_{r \in P} \Big[F_r^{\text{stays-constr}} \wedge \bigvee_{b \in B^+(r)} \Big(F_{r,b}^{\text{stays-body}} \wedge \neg \ell[b]\Big)\Big],$$

"there is a rule in $\mathrm{Constr}(P^M_{X_1 \subseteq X})$ that is not satisfied by L";

$$F^{(\mathrm{b})} := \bigvee_{a \in \mathrm{at}(P)} \Big[\ell[a] \wedge \Big(\neg m[a] \vee T^{a \in X}\Big)\Big],$$

"L contains an atom that belongs to X or does not belong to M;"

$$F^{(\mathrm{c})} := \Big[\bigwedge_{a \in \mathrm{at}(P)} m[a] \leftrightarrow \Big(\ell[a] \vee x[a]\Big)\Big] \vee \Big[\bigvee_{a \in \mathrm{at}(P)} \Big(\ell[a] \vee x[a]\Big) \wedge \neg m[a]\Big],$$

"$L \cup X_1$ equals M or $L \cup X_1$ contains an atom that is not in M;"

$$F^{(\mathrm{d})} := \bigvee_{r \in P} \Big[\bigwedge_{a \in B^-(r)} \neg m[a] \wedge \bigwedge_{a \in H(r)} \Big(\neg \ell[a] \wedge \neg x[a]\Big) \wedge \bigwedge_{b \in B^+(r)} \Big(\ell[b] \vee x[b]\Big)\Big],$$

"P^M contains a rule that is not satisfied by $L \cup X_1$."

By definition of F^{supp} and F^{level}, and by Lemma 9.7, it follows that L is the least model of $\mathrm{DH}(P')$. Furthermore, by Lemma 9.4 it follows that $F^{\text{MinCheck}}[\tau_M \cup \tau_{X_1}]$ is satisfiable if and only if Algorithm MinCheck(P, X_1, M) returns *True*.

By construction, $\|F^{\text{supp}}\| = \mathcal{O}(\|P\|)$, $\|F^{\text{level}}\| = \mathcal{O}(\|P\| \log |\mathrm{at}(P)|)$, and $\|F^{\text{cond}}\| = \mathcal{O}(\|P\|)$. Hence $\|F^{\text{MinCheck}}\| = \mathcal{O}(\|P\| \log |\mathrm{at}(P)|)$ as claimed. Clearly F^{MinCheck} can be obtained in polynomial time. □

Proof of Theorem 9.1. It is now easy to define the required QBFs by setting

$$Q_{\text{Brave}}(a) := \exists V_m.\forall V_x.\exists(V_\ell \cup A).\ F^{\text{mod}} \wedge F^{\text{MinCheck}} \wedge m[a] \quad \text{and}$$

$$Q_{\text{Skept}}(a) := \exists V_m.\forall V_x.\exists(V_\ell \cup A).\ F^{\text{mod}} \wedge F^{\text{MinCheck}} \wedge \neg m[a],$$

where $A = var(F^{\text{MinCheck}} \setminus (V_m \cup V_\ell \cup V_x))$ contains the auxiliary variables of F^{MinCheck} such as vectors and auxiliary variables in $F^{+1}(\vec{x}, \vec{y})$. It only remains to define F^{mod}, a propositional formula on the variables in V_m such that $F^{\text{mod}}[\tau_M]$ is satisfiable if and only if M is a model of P,

$$F^{\text{mod}} := \bigwedge_{r \in P} \Big[\bigwedge_{b \in B^-(r)} \neg m[b] \rightarrow \Big(\bigvee_{b \in B^+(r)} \neg m[b] \vee \bigvee_{b \in H(r)} m[b] \Big) \Big].$$

By Lemmas 9.4 and 9.8 it follows that $Q_{\text{Brave}}(a)$ evaluates to true if and only if a is in some answer set of P; and $Q_{\text{Skept}}(a)$ evaluates to false if and only if a is in all answer sets of P. The formulas $Q_{\text{Brave}}(a)$ and $Q_{\text{Skept}}(a)$ can clearly be constructed in polynomial time. Since $\|F^{\text{mod}}\| = \|P\|$, it follows from Lemma 9.8 that $\|Q_{\text{Brave}}(a)\| = \|Q_{\text{Skept}}(a)\| = \mathcal{O}(\|P\| \log |at(P)|)$. Hence the theorem is shown true. □

We would like to note that Theorem 9.1 remains true if we require that the formulas $Q_{\text{Skept}}(a)$ and $Q_{\text{Brave}}(a)$ are in prenex CNF form, i.e., of the form $\exists V_m.\forall V_x.\exists V_\ell \cup A.\ F_{\text{cnf}}$ where the matrix F_{cnf} is a propositional formula in conjunctive normal form (CNF). This is a direct consequence of the fact that one can transform in linear time any propositional formula F into a satisfiability-equivalent formula F_{cnf} in CNF using the well-known transformation due to Tseitin [Tse68], see also [KL99]. The transformation produces new variables, called extension variables, and we have $var(F) \subseteq var(F_{\text{cnf}})$. The transformation clearly works also for QBF formulas, where the extension variables are existentially quantified in the innermost quantifier block.

Our approach could be of practical use, at least for certain classes of instances, and hence might fit into a portfolio-based solver. This is certainly an interesting future research topic, since it requires additional considerations, e.g., preprocessing techniques to reduce the size of **Normal**-backdoors like *shifting* [Jan+09]; more sophisticated encodings of our formulas Q_{Brave} and Q_{Skept} like improvements carried out in [Jan06]; or compilations to Smt like Sat(Dl) [JNS09] and Sat(Acyclicity) [GJR14].

We would like to point out that our approach directly extends to more general problems, when we look for answer sets that satisfy a certain global property which can be expressed by a propositional formula F^{prop} on the variables in V_m. We just check the satisfiability of $F^{\text{mod}} \wedge F^{\text{MinCheck}} \wedge F^{\text{prop}}$.

9.1.3 Normality Backdoor Detection

In this section we study the problem of finding strong **Normal**-backdoors. We exploit a connection between strong **Normal**-backdoors of a program and vertex covers of a

certain graph representation associated with the program. We first observe the following connection between deletion **Normal**-backdoors and strong **Normal**-backdoors.

Lemma 9.9. *A set X is a strong* **Normal***-backdoor of a program P if and only if it is a deletion* **Normal***-backdoor of P.*

Proof. We observe that the class **Normal** is hereditary. Thus the if direction holds by Proposition 3.7. We proceed to show the only-if direction. Assume X is a strong **Normal**-backdoor of P. Consider a rule $r' \in P - X$ which is not tautological. Let $r \in P$ be a rule from which r' was obtained in forming $P - X$. We define $\tau \in 2^X$ by setting all atoms in $X \cap (H(r) \cup B^-(r))$ to 0, all atoms in $X \cap B^+(r)$ to 1, and all remaining atoms in $X \setminus \mathrm{at}(r)$ arbitrarily to 0 or 1. Since r is not tautological, this definition of τ is sound. It remains to observe that $r' \in P_\tau$. Since X is a strong **Normal**-backdoor of P, the rule r' is normal. Hence, the lemma follows. □

Recall the definition of a head dependency graph from Definition 5.6 (see p. 55). The following is an immediate consequence from Lemmas 9.9 and 5.7.

Lemma 9.10. *Let P be a program. A set $X \subseteq \mathrm{at}(P)$ is a strong* **Normal***-backdoor of P if and only if X is a vertex cover of H_p.*

Proof. Let X be a strong **Normal**-backdoor of P which is also a deletion **Normal**-backdoor according to Lemma 9.9. Consider an edge uv of H_p, then there is a rule $r \in P$ with $u, v \in H(r)$ and $u \neq v$. Since X is a deletion **Normal**-backdoor of P, we have $\{u, v\} \cap X \neq \emptyset$. We conclude that X is a vertex cover of H_p.

Conversely, assume that X is a vertex cover of H_p. We proceed indirectly and assume that X is not a strong **Normal**-backdoor of P. By Lemma 9.9, X is neither a deletion **Normal**-backdoor of P. It follows that there is a rule $r \in P - X$ with two atoms $u, v \in H(r)$; clearly $u, v \notin X$. Consequently there is an edge uv of H_p such that $\{u, v\} \cap X = \emptyset$, contradicting the assumption that X is a vertex cover. Hence the lemma prevails. □

Theorem 9.11. *Given a program P and an integer k, we can find in time $\mathcal{O}(1.2738^k + k\|P\|)$ a strong* **Normal***-backdoor of P with a size $\leq k$ or decide that no such backdoor exists.*

Proof. In order to find a strong **Normal**-backdoor of a given program P, we use Lemma 9.10 and find a vertex cover of size at most k in the head dependency graph (which has $n = |\mathrm{at}(P)| \leq \|P\|$ many vertices). A vertex cover of size k, if it exists, can be found in time $\mathcal{O}(1.2738^k + kn)$ [CKX06]. Thus, the theorem holds. □

In Theorem 9.1 we assume that a strong **Normal**-backdoor of size at most k is given when solving the problems Strong **Normal**-Backdoor-Brave-Reasoning and

Skeptical-Reasoning. As a direct consequence of Theorem 4.4, this assumption can be dropped, and we obtain the following corollary.

Corollary 9.12. *The results of Theorem 9.1 and Corollary 9.2 still hold if the backdoor is not given as part of the input.*

9.2 Backdoors to Tightness

In the previous section, we established an fpt-transformation from Asp into Sat that takes advantage of small backdoors to normal programs. The main Asp reasoning problems and the problem of determining an answer set are NP-complete and co-NP-complete, respectively. Another class of programs where these problems are also NP-complete and co-NP-complete, respectively, are programs where certain cyclic dependencies between atoms in the head and the positive body are forbidden, so-called tight programs [Fag94; BEL00]. Further, tight programs allow for a compact transformation into SAT [Fag94; Cla78]. Hence, the class of tight programs is a natural candidate for a target class, similar to the class of normal programs. In this section, we investigate whether small backdoors into tight programs also admit an fpt-transformation from Asp into Sat.

We first give a definition for tightness. We associate with each program P its *positive dependency digraph* D_P^+. Recall the definition of a positive dependency digraph (see also Definition 8.40), which has the atoms of P as vertices and a directed edge (x, y) between any two distinct atoms $x, y \in \mathrm{at}(P)$ for which there is a rule $r \in P$ with $x \in H(r)$ and $y \in B^+(r)$. A program is called *tight* if D_P^+ is acyclic [LL03]. We denote the class of all tight programs by **Tight**.

It is well known that the main Asp reasoning problems are in NP and co-NP for tight programs; in fact, a reduction to Sat based on the concept of *loop formulas* has been proposed by Lin and Zhao [LZ04b]. This was then generalized by Lee and Lifschitz [LL03] with a reduction that takes as input a disjunctive program P together with the set S of all directed cycles in the positive dependency digraph of P, and produces a CNF formula F such that answer sets of P correspond to the satisfying assignments of F. This provides an fpt-reduction from the problems Brave Reasoning and Skeptical Reasoning to Sat, when parameterized by the number of all cycles in the positive dependency digraph of a given program P, assuming that these cycles are given as part of the input.

The number of cycles does not seem to be a very practical parameter, as this number can quickly become very large even for very simple programs. Lifschitz and Razborov [LR06] have shown that already for normal programs an exponential blowup may occur, since the number of cycles in a normal program can be arbitrarily large. Hence, it would be interesting to generalize the result of Lee and Lifschitz [LL03] to a more powerful parameter. In fact, the size k of a deletion **Tight**-backdoor would be a candidate for such a parameter, as it is easy to see, it is at most as large as the number of cycles, but can be exponentially smaller. This is a direct consequence of the following two observations:

Observation 9.13. *If a program P has exactly k cycles in D_P^+, we can construct a deletion* **Tight***-backdoor X of P by taking one element from each cycle into X.*

Observation 9.14. *If a program P has a deletion* **Tight***-backdoor of size* 1*, it can have arbitrarily many cycles that run through the atom in the backdoor.*

In the following, we show that this parameter k is of little use, as the reasoning problems already reach their full complexity for programs with a deletion **Tight**-backdoor of size 1.

Theorem 9.15. *The problems* Brave Reasoning *and* Skeptical Reasoning *when parameterized by the size of a strong* **Tight***-backdoor are Σ_2^p-hard and Π_2^p-hard, respectively, even for programs that admit a strong* **Tight***-backdoor of size* 1*, and the backdoor is provided with the input. The problems remain hard when we consider a deletion* **Tight***-backdoor instead of a strong* **Tight***-backdoor.*

Proof. Consider the reduction from Eiter and Gottlob [EG95] which reduces the Σ_2^p-hard problem $\exists\forall$-QBF Model Checking to the problem Consistency (which decides whether given a program P has an answer set). A $\exists\forall$ quantified Boolean formula (QBF) has the form $\exists x_1 \cdots \exists x_n \forall y_1 \cdots \forall y_m D_1 \vee \ldots \vee D_r$ where each $D_i = l_{i,1} \wedge l_{i,2} \wedge l_{i,3}$ and $l_{i,j}$ is either an atom $x_1, \ldots, x_n, y_1, \ldots, y_m$ or its negation. Their construction yields a program $P := \{x_i \vee v_i;\ y_i \vee z_j;\ y_j \leftarrow w;\ z_j \leftarrow w;\ w \leftarrow y_j, z_j;\ w \leftarrow g(l_{k,1}), g(l_{k,2}), g(l_{k,3});\ w \leftarrow \neg w\}$ for each $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, m\}$, $k \in \{1, \ldots, r\}$, and g maps as follows $g(\neg x_i) = v_i$, $g(\neg y_j) = z_j$, and otherwise $g(l) = l$. Since $P_{w=0} = \{x_i \vee v_i;\ y_j \vee z_j\}$ and $P_{w=1} = \{x_i \vee v_i;\ y_j \vee z_j;\ y_j;\ z_j\}$ are both in **Tight**, the set $X = \{w\}$ is a strong **Tight**-backdoor of P of size 1. Thus the restriction does not yield tractability. The intractability of Skeptical Reasoning follows directly by the reduction of Eiter and Gottlob [EG95] from the problem Consistency. Hardness of the other problems can be observed easily. Since $P - \{w\} := \{x_i \vee v_i;\ y_i \vee z_j;\ y_j;\ z_j;\ \leftarrow y_j, z_j;\ \leftarrow g(l_{k,1}), g(l_{k,2}), g(l_{k,3})\}$ for each $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, m\}$, $k \in \{1, \ldots, r\}$ is tight, we obtain a deletion **Tight**-backdoor of size 1. In consequence we established the theorem. □

Recall Definition 8.56 which states that a program is head-cycle-free if its positive dependency graph contains no cycles where atoms on the cycle occur together in the head of a rule. We denote the class of all head-cycle-free programs by **HCF**. As an immediate consequence of Theorem 9.15 we obtain the following result.

Corollary 9.16. *The problems* Brave Reasoning *and* Skeptical Reasoning *when parameterized by the size of a strong* **HCF***-backdoor are Σ_2^p-hard and Π_2^p-hard, respectively, even for programs that admit a strong* **HCF***-backdoor of size* 1*, and the backdoor is provided with the input. The problems remain hard when we consider a deletion* **HCF***-backdoor instead of a strong* **HCF***-backdoor.*

Proof. The proof is easy and follows from the fact that every tight program is also head-cycle-free. □

Background and Related Work

Transformations from Asp problems into Sat have been explored by several authors; existing research mainly focuses on transforming programs for which the reasoning problems already belong to NP or co-NP. In particular, transformations have been considered for head-cycle-free programs [BD94], tight programs [Fag94], and normal programs [LZ04b; Jan06].

Some authors have generalized the above transformations to capture programs for which the reasoning problems are outside NP and co-NP. Janhunen et al. [Jan+06] considered programs where the number of disjunctions in the heads of rules is bounded. They provided a transformation that allows a Sat encoding of the test whether a candidate set of atoms is indeed an answer set of the input program. Lee and Lifschitz [LL03] considered programs with a bounded number of cycles in the positive dependency graph. They suggested a transformation that, similar to ours, transforms the input program into an exponentially larger propositional formula whose satisfying assignments correspond to answer sets of the program. As pointed out by Lifschitz and Razborov [LR06], this transformation produces an exponential blowup already for normal programs (we note that by way of contrast, our transformation is in fact quasilinear for normal programs).

A general theoretical framework to classify parameterized problems on whether they admit an fpt-reduction to Sat or not has lately been introduced by DeHaan and Szeider [DS14b]. Our approach has recently been applied to abduction by Pfandler, Rümmele, and Szeider [PRS13].

Over the last years several transformations have been implemented into various Asp solvers. Disjunctive solvers that are mainly based on compilations into Sat and use the logical characterizations of loop formulas [LL03; Lee05] are for example Cmodels3 [Lie05; Lie11], ClaspD [Dre+08a; GKS12b], and ClaspD-2 [GKS13]. ClaspD and ClaspD-2 are also Sat-solvers while Cmodels3 uses other Sat-solvers (e.g., Zchaff [MFM05; Mal07]) as blackbox. In contrast to ClaspD the solver ClaspD-2 does not explicitly generate loop formulas, instead it uses certain encodings of unfounded sets where variables to represent relevant parts of candidate assignments to check and allows to reuse constraints that characterize unfounded sets. The solver GnT [Jan+06; Jan13b] uses a guess and check approach where a candidate is guessed by an instance and checked by another instance. DLV [Leo+06; DLV12] is a native Asp solver that also uses a guess and check approach, the main algorithm is DPLL-based, however it uses a four-valued interpretation for the intermediate truth assignments of atoms [GLM06]. Reductions to Qbf-Sat have so far been only of prototypical interest [Egl+00; Egl+01]. To our knowledge recent solving techniques in Qbf-Sat [LB10; BLS11; Lon12; LEV13] have not been applied to Asp solving.

Contribution and Discussion

In this chapter we have established a method to utilize certain structural aspects of disjunctive logic programs to transform the Asp reasoning problems brave and skeptical reasoning into Qbf-Sat. It is known that the two reasoning problems, when restricted to so-called *normal programs*, drop to NP and co-NP [BF91; MT91b; MT91a], respectively. Our method exploits a small distance of a given program from being normal by means of a structural parameter k which seems quite natural and has to our knowledge not been considered in the literature before. We measure the distance in terms of smallest strong backdoors into normal programs. Note that we already used deletion backdoors into normal programs in Chapter 5 to lift parameters that are based on the deletion of atoms and have been defined for normal programs only to disjunctive programs. However, in contrast to previous chapters and previous work on backdoors, where the considered reasoning problems are in fact tractable when parameterized by the size of a smallest strong backdoor into a fixed "tractable" target class, the problems remain intractable when parameterized by the size of a smallest strong backdoor into normal programs. Hence, we can see the class **Normal** as an "intractable target class" where the backdoor approach still pays off as the reduction is fixed-parameter tractable and gives rise to a computationally easier class of programs. The parameter size of a smallest strong **Normal**-backdoor is less restrictive and more general than the parameters that yield tractability of the considered reasoning problems.

The backdoor approach used in this chapter is in fact similar to Chapter 3. We first detect a backdoor (find a backdoor of size at most k of the given program P, or decide that a backdoor of size k does not exist) and then evaluate the backdoor (apply the found backdoor to the instance and determine the solution). However in the second step we transform the given program P in time $\mathcal{O}(\|P\| \log |\mathrm{at}(P)|)$ into quantified Boolean formulas $Q_{\mathrm{Brave}}(a^*)$ and $Q_{\mathrm{Skept}}(a^*)$, respectively, and solve the formulas using a Qbf-Sat-solver. Note that the Qbf-Sat encoding is more succinct than a Sat encoding (as presented in our earlier version [FS13] of the transformation). The formula $Q_{\mathrm{Brave}}(a^*)$ evaluates to true if and only if the given atom a^* belongs to some answer set of P. The formula $Q_{\mathrm{Skept}}(a^*)$ evaluates to false if and only if the given atom a^* belongs to all answer sets of P. Both formulas are of size $\mathcal{O}(\|P\| \log |\mathrm{at}(P)|)$ and contain exactly k many universally quantified variables where k is the size of the given strong **Normal**-backdoor of P. As a result the combinatorial explosion, which is expected when transforming problems from the second level of the Polynomial Hierarchy to the first level, is confined to the parameter k, while the running time is polynomial in the input size n and the order of the polynomial is independent of k. In that way we provide an alternative approach to the concept loop formulas [LL03; Lee05] which allows us to process classes of programs that are larger than normal or head-cycle-free programs. We would like to note that loop formulas can be seen as an fpt-reduction to Sat, where the parameter is the number of loops. Generally, we call such fixed-parameter tractable transformations *complexity barrier breaking reductions* which is to our knowledge the first positive result from the perspective of parameterized complexity theory beyond NP. While our transformation

is fixed-parameter tractable we can neither expect it to be polynomial-time tractable nor that the problems brave and skeptical reasoning are fixed-parameter tractable when parameterized by strong **Normal**-backdoors, since it would simply imply the collapse of the Polynomial Hierarchy which is considered highly unlikely in standard complexity theory. Interestingly, our para-NP-completeness or co-para-NP-completeness, respectively, results for Brave Reasoning and Skeptical Reasoning when parameterized by the size of a strong **Normal**-backdoor are then still positive results. Moreover, we have established limits of such fixed-parameter tractable reductions by considering backdoors to tightness. We have shown that the reasoning problems already reach their full complexities (i.e., completeness for the second level of the Polynomial Hierarchy) with programs of distance one from being tight. Hence, a fixed-parameter tractable transformation into Sat for programs of distance $k > 0$ from being tight is not possible unless the Polynomial Hierarchy collapses.

Finally, we would like to point out that out approach could be of practical use, at least for certain classes of instances, and hence might fit into a portfolio-based solver. This is certainly an interesting future research topic, since it requires additional considerations, e.g., preprocessing techniques to reduce the size of **Normal**-backdoors like *shifting* [Jan+09]; or compilations to Smt like Sat(Dl) [JNS09] and Sat(Acyclicity) [GJR14].

CHAPTER 10

Practical Considerations

In this chapter we discuss some practical considerations and present first empirical data, although our main focus is theoretical. The underlying idea for fixed-parameter tractability is that problem instances for which the parameter is small can be solved efficiently. It is therefore natural to ask how the values of a parameter are distributed in various problem instances. Hence, we investigate in Section 10.1 the size of backdoors for various benchmark programs, focusing on the target classes **Horn**, **no-DC**, and **Normal**. As expected, structured programs, originating from application domains, have smaller backdoors than random instances. In Section 10.2 we sketch some ideas on how backdoors into tractable target classes could be used within a heuristic of an Asp solver. A conclusive evaluation of these ideas requires a rigorous experimental setup, which is way beyond the scope of this thesis.

10.1 Backdoor Detection

10.1.1 Encodings and Setup

We have determined strong **Horn**-backdoors and strong **Normal**-backdoors for various benchmark programs by means of encodings into answer set programming, integer linear programming (Ilp), local search (LS), and propositional satisfiability (Sat). We use the connection stated in Lemma 4.3 and compile the problem of finding a minimum vertex cover (k-vertex cover) into the respective domain. The encodings are straightforward: Let P be a program (without constraints or tautological rules) and let $N_p = (V, E)$ be its negation dependency graph. For the Asp encoding we proceed as follows: Among the atoms of our Asp program will be atoms $\{ e_{uw} : uw \in E \}$ and atoms $C = \{ c_v : v \in V \}$. The truth values of the atoms in C represent a subset $S \subseteq \mathrm{at}(P)$ such that c_v is true if and only if $v \in S$ (a vertex cover). We introduce for every edge $vw \in E$ a constraint $\leftarrow e_{vw}, \neg c_v, \neg c_w$ and a choice rule $\{c_u, c_v\} \leftarrow e_{u,v}$. Moreover, we add a

statement to minimize the number of atoms in C that belong to an answer set (see [SNS02; Geb+12] for choice rules and minimize statements). For the Ilp encoding we proceed as follows: We introduce for every vertex $v \in V$ a binary variable b_v, we add for every edge $vw \in E$ a constraint $b_v + b_w \geq 1$, and minimize the sum $\sum_{v \in V} b_v$. For LS we ran designated local search based vertex cover solvers [CSC10; CSS11; Cai+13] on the graph N_p. For the Sat encoding we used an encoding similar to the encoding presented in [Gar12]. We introduce for every edge $uv \in E$ a binary clause and add a sequential unary counter [Sin05] to express that at most k vertices belong to a vertex cover. Moreover, we used an incremental encoding into integer linear programming (Ilp) of SageMath [Wil12] to compute small deletion **no-DC**-backdoors.

The answer set program that solves backdoor detection was generated by means of Asp meta programming [GKS11] and solved using Clasp [Geb+11c; GKS12b] version 3.0.5 with an unsatisfiable-core based optimization strategy (the command line parameter "–opt-strategy=5" yields the behavior) [And+12]. The integer linear program was generated using the open source mathematics framework SageMath [Wil12] with Python [Ros95], solved using ILOG CPLEX 12 [ILO11] and Gurobi [Gur14].

The results obtained with Ilp methods using modern solvers like CPLEX and Gurobi come along with a certain inaccuracy (see e.g., [Coo+13; Chv83; Sch98]). Therefore, we ran Clasp on the structured instances for backdoors into **Horn** and **Normal** using the encoding of k-vertex cover problem described above to obtain optimality.

10.1.2 Instances

We mainly used benchmark sets from the first three Asp competitions [CIR14; Den+09; Geb+07], because most of the instances contain only normal and/or disjunctive rules and no extended rules (cardinality/weight-constraints). We are aware that one can preprocess extended rules and compile them into normal rules. Even though recent versions of the solver Clasp provide such an option [Geb+10] and more sophisticated implementations like lp2normal [BJ13; BGJ14] are available, those compilations blow up the instances significantly. Hence we omitted it for pragmatic reasons.

- `ConformantPlanning`: secure planning under incomplete initial states [TPS09]; instances provided by Gebser and Kaminski [GK12].
- `Factoring`: factorization of a number where an efficient algorithm would yield a cryptographic attack by Gebser [Dre+08b]; for instances see [GS09].
- `HanoiTower`: classic Towers of Hanoi puzzle by Truszczynski, Smith and Westlund; for instances see [CIR14].
- `GraphColoring`: classic graph coloring problem by Lierler and Balduccini; for instances see [CIR14].

- `KnightTour`: finding a tour for the knight piece traveling any square following the rules of chess by Zhou, Calimeri, and Santoro; for instances see [CIR14].
- `Labyrinth`: classical Ravensburger's Labyrinth puzzle by Gebser; for instances see [CIR14].
- `MinimalDiagnosis`: an application in systems biology [Geb+11d]; for instances see [CIR14].
- `MSS/MUS`: problem whether a clause belongs to some minimal unsatisfiable subset [JM11]; instances provided by Gebser and Kaminski [GK12].
- `Solitaire`: classical Peg Solitaire puzzle by Lierler and Balduccini; for instances see [CIR14].
- `StrategicCompanies`: encoding the Σ_2^p-complete problem of producing and owning companies and strategic sets between the companies [Geb+07].
- `RandomQBF`: transformations of randomly generated 2-QBF instances using the method by Chen and Interian [CI05]; for instances see [Geb+07].
- `RLP`: Randomly generated normal programs, of various density (number of rules divided by the number of atoms) [ZL03]; for instances see [Geb+07].
- `RandomNonTight`: Randomly generated normal programs in [GS09] with $n = 40$, 50, and 60 variables, respectively with 40 instances per step instances.

domain	instance set (#instances)	disj.	#atoms	horn bd(%)	stdev
AI	`HanoiTower (60)`	–	32956.7	4.28	0.08
	`StrategicCompanies (15)`	+	2002.0	6.03	0.04
	`MinimalDiagnosis (551)`	+	111856.5	10.74	1.71
Graph	`GraphColoring (60)`	–	3544.4	19.47	0.79
Planning	`MSS/MUS (38)`	+	49402.3	3.80	0.70
	`ConformantPlanning (24)`	+	1378.2	8.43	2.12
Cryptography	`Factoring (10)`	–	3336.8	25.76	1.30
Puzzle	`Labyrinth (261)`	–	55604.9	3.42	0.82
	`KnightTour (10)`	–	23156.9	33.08	0.20
	`Solitaire (25)`	–	11486.8	38.88	0.20
Random	`RandomQBF (15)`	+	160.1	49.69	0.00
	`RLP (572)`	–	174.7	68.89	5.18
	`RandomNonTight (220)`	–	200.0	89.85	0.24

Table 10.1: Size of smallest strong **Horn**-backdoors (bd) for various benchmark sets, given as % of the total number of atoms (#atoms) by the mean over the instances.

10.1.3 Backdoors to Tractability

Table 10.1 illustrates our results on the size of small strong **Horn**-backdoors of the considered benchmark instances. The structured instances have, as expected, significantly smaller strong **Horn**-backdoors than the random instances. We would like to mention that the random programs from the Asp competitions contain a rather small number of atoms. So far we have no good evidence why in particular the sets `KnightTour` and `Solitaire` have rather large strong **Horn**-backdoors compared to other structured instances.

For the acyclicity based target classes $\mathcal{C} \in \mathcal{Acyc}$ we have computed small deletion $\mathcal{C}$-backdoors only for instances from the sets `HanoiTower`, `MinimalDiagnosis`, `GraphColoring`, `MSS/MUS`, `ConformantPlanning`, `Factoring`, `Labyrinth` `KnightTour`, and `Solitaire` since the currently available algorithms did not scale on the other sets. In fact, we set a quite permissive timeout of one day. However, we were able to compute a backdoor for instances from the sets `HanoiTower`, `GraphColoring`, and `ConformantPlanning` mostly within 600 seconds. We have obtained similar sized small deletion **Horn**-backdoors and deletion **no-DC**-backdoors for instances from the sets `HanoiTower` and `GraphColoring`, respectively. The size of small deletion **no-DC**-backdoors of instances of `MinimalDiagnosis` was almost twice the size of small strong **Horn**-backdoors. The size of small deletion **no-DC**-backdoors of instances of the sets `MSS/MUS`, `Labyrinth`, and `Solitaire` was about half of the size of small strong **Horn**-backdoors. The size of small deletion **no-DC**-backdoors of instances of the set `Factoring` was significantly smaller, more precisely, about 6% of the size of small strong **Horn**-backdoors. The size of small deletion **no-DC**-backdoors of instances of the

domain	instance set (#instances)	#atoms	normal bd (%)	stdev
AI	`MinimalDiagnosis (551)`	111856.5	10.74	1.71
	`StrategicCompanies (15)`	2002.0	6.03	0.04
Planning	`MSS/MUS (38)`	49402.3	1.90	0.35
	`ConformantPlanning (24)`	1378.2	0.69	0.39
Random	`RandomQBF` (15)	160.1	49.69	0.00

Table 10.2: Size of smallest strong **Normal**-backdoor (bd) for various benchmark sets, given as % of the total number of atoms by the mean over the instances.

set `KnightTour` was about 80% and of the set `ConformantPlanning` about 88% of the size of small strong **Horn**-backdoors.

10.1.4 Backdoors to Normality

Table 10.2 illustrates our results on the size of small strong **Normal**-backdoors of the considered benchmark instances (note that we skipped programs that are already normal). Similar to Table 10.1 in the previous section structured instances have smaller backdoors than random instances. When we compare the size of a smallest strong **Normal**-backdoor with the size of a smallest strong **Horn**-backdoor for selected sets it seems at first surprising that for `MinimalDiagnosis`, `StrategicCompanies`, and `RandomQBF` small backdoors into **Horn** and into **Normal** are of the same size. However, instances from both sets contain various disjunctions in the head but only a few constraints with non-negative bodies. Hence, a strong **Horn**-backdoor is also a strong **Normal**-backdoor and vice versa. When we consider the set `ConformantPlanning` it turns out that smallest strong **Normal**-backdoors are significantly smaller (0.7% vs. 8.8% of the total number of atoms). As instances from `ConformantPlanning` have rather small backdoors our transformation into QBF-SAT (see Section 9) seems to be feasible for these instances.

10.2 Backdoor Evaluation

Instead of applying the algorithm from Section 3.3 directly, one can possibly use backdoors to improve modern heuristics in ASP solvers to obtain a speed-up. Most modern solver heuristics work independently from the current truth assignment. They assign to each atom in the program a score and incorporate into the score the learned knowledge based on derived conflicts (history of the truth assignments). Various studies on the effect of restricting decision heuristics to a subset of variables based on structural properties have been carried out in the context of SAT, where both positive [GMS98; GMT02; Str00] and negative effects [JN08] have been observed depending on the domain of the instances. Järvisalo and Junttila [JJ09] proved that branching only on the input variables

of an underlying problem (before translating it into CNF) can imply a super-polynomial increase in the length of optimal proofs for learning-based heuristics. However, very recent results by Gebser et al. [Geb+13a] suggest that modern Asp solvers with a clause learning heuristic can benefit from additional structural information when a relaxed form of restricted branching is used, namely increasing the score of atoms if a certain structural property prevails. Those properties have to be manually identified. Since backdoor atoms are of structural importance for the problem it seems reasonable to initially increase the score of the atoms if the atom is contained in the considered backdoor. As strong **Horn**-backdoors are relatively easy to compute and very easy to approximate one could occasionally update the heuristic based on a newly computation of a backdoor. So a solver could benefit from backdoors in both the initial state and while learning new atoms. [Geb+13a]. However, we do not expect a practically competitive heuristic that is purely based on backdoors and does not take other aspects of Asp solving into account. Therefore, a conclusive evaluation requires a rigorous experimental setup that takes the interaction of various heuristic methods into account (e.g., interaction of solver techniques [KSM11] and tuning of parameter values [Hoo12], modifying the branching heuristic by caching truth values of atoms and reusing them [PD07], modifying the atom score initially, or while learning conflict clauses, or at a certain depth of the search).

Contribution and Future Work

In this chapter we gave an idea about the size of small deletion backdoors into **Horn**, **no-DC**, and **Normal** in realistic instances. The backdoor approach as established in Chapters 3 and 4 is only effective when we can detect a backdoor within a reasonable amount of time and the size of the backdoor is small enough. However, it turns out that backdoors into **Horn** or **no-DC** are still orders of magnitudes to large to exploit them without further improvements by means of the backdoor approach. We observed that one can detect small (not necessarily optimal) backdoors into **Horn** relatively fast using an Ilp solver. Our results revealed that the discovered small backdoors were almost always optimal, but detecting optimality requires significantly more computation time. Unfortunately, our incremental approach to compute small backdoors into **no-DC** seems to be rather ineffective. Moreover, we cannot effectively detect small backdoors into one of the other acyclicity-based target classes, since the currently available algorithms can only deal with rather small instances within a reasonable computation time. Hence, our backdoor approach seems currently without further improvements only of theoretical interest. However, we have discovered that there are practical instances where small deletion backdoors into **Normal** are relatively small. Similar to backdoors into **Horn** we can find small (not necessarily optimal) backdoors into **Normal** relatively fast. Hence, our transformation into Qbf-Sat seems feasible for these instances and could provide an alternative approach to loop formulas. Therefore, we think that a practical implementation is an interesting subject for further investigations. Further, we have provided generic ideas of how one can possibly use backdoors into target classes where the reasoning problems are tractable to improve certain aspects of modern heuristics.

A conclusive evaluation seems to be a quite interesting topic for further research, in particular since also positive effects on speeding up solvers have been observed when structural information has been made available to solvers by means of restrictions on the decision heuristic [Geb+13a].

CHAPTER 11

Summary

We hope that this thesis has made contribution towards a better theoretical understanding of answer set programming and its computational complexity. We used parameterized complexity theory and mainly the notion of backdoors to carry out a detailed theoretical worst-case complexity analysis, which takes also certain structural properties (e.g., size of a backdoor) of the input instance into account and is hence closer to the practical hardness of problems on real-world instances.

11.1 Technical Results

We have introduced the backdoor approach (backdoor detection and backdoor evaluation), which has mainly been used in Sat and Csp for theoretical analysis, to the domain of propositional answer set programming. In a certain sense, the backdoor approach allows us to augment known tractable classes and makes efficient solving methods for tractable classes generally applicable. We have studied various target classes and have established tractability results for backdoor detection as well as hardness results depending on the considered target class and the kind of the backdoor (Chapters 3–4). We have considered approaches to extend known parameters using backdoors and more precise approaches to the evaluation of strong and deletion backdoors (Chapters 5–7). We have established a detailed theoretical comparison of the various Asp parameters, which are based on backdoors and parameters that are known from the literature, in terms of their generality (Chapter 8). We have introduced the notion of a complexity barrier breaking reduction, which is a fixed-parameter tractable complexity reduction from a problem that is located on the second level (or higher) of the Polynomial Hierarchy to itself when the input is restricted (or another problem) such that the complexity drops (e.g., NP-complete) without making the problem itself fixed-parameter tractable. We have established a complexity barrier breaking reduction for the main reasoning problems of disjunctive Asp using backdoors into a normal programs resulting in a reduction of the problem

complexity. A backdoor into normal programs of small size hence captures structural properties of disjunctive Asp instances. We have also shown that we cannot establish a complexity barrier breaking reduction using backdoors into tightness under standard assumptions (Chapter 9). Finally, we have discussed some practical considerations and presented some empirical data on how the size of a backdoor is distributed in various benchmark instances (Chapter 10).

11.2 Overall Results

We have introduced backdoors to the domain of answer set programming and have established a detailed theoretical worst-case complexity analysis (including polynomial-time preprocessing and more detailed approaches to the evaluation of backdoors) considering the size of a backdoor as a parameter. The analysis yields positive results for the problem backdoor evaluation as well as both positive and negative results for the backdoor detection problems. Some results reveal connections to and provide interesting applications of very recent combinatorial results in parameterized complexity theory.

Structure has been studied earlier in answer set programming from various perspectives. However, we have provided a detailed theoretical comparison of various Asp parameters in terms of their generality and the relationship between the parameters. Furthermore, our framework of answer set backdoors also generalizes several structural parameters known from the literature.

We have proposed a novel method to solve Asp reasoning problems on disjunctive programs. The main part is a transformation which takes advantage of a small backdoor into normal programs and is fixed-parameter tractable (whereas the problems themselves are *not* fixed-parameter tractable). Our method provides an alternative approach to the concept of loop formulas. We have shown that backdoors into tight programs, which provide the basis to solve programs using succinct compilations into a Sat formulas, can not be used to establish a similar method, since the reasoning problems remain complete for the second level of the Polynomial Hierarchy when parameterized by a backdoor into tight programs.

Finally, our complexity barrier breaking reductions provide a new way of using fixed-parameter tractability and enlarges its applicability. In fact, our approach as exemplified above for a method to solve Asp reasoning problems is very general and applicable to a wide range of other hard combinatorial problems that lie beyond NP or co-NP which has actually already been shown in subsequent results [PRS13; DS14a; DS14b; HS15].

11.3 Future Research

The results and concepts of this thesis give rise to several interesting research questions. For instance, it would be interesting to lift Theorem 3.12 to target classes that contain programs with an exponential number of answer sets and are not enumerable, but where

the set of all answer sets can be succinctly represented. A simple example is the class of programs that consist of (in)dependent components of bounded size. Our investigations on the various target classes could be extended to further classes of programs to obtain a wider picture similar to the target classes for satisfiability backdoors based on Schaefer's classes [GS12b]. In particular, it would be interesting to investigate on the classes considered by Truszczyński [Tru11] where the reasoning problems are tractable or the complexity drops at least to NP (e.g., dual-normal programs). Moreover, an interesting research question is whether the notion of a strong backdoor can be extended by allowing unions of classes as a heterogeneous target class [Gas+14], in other words, different partial truth assignment reducts may belong to different target classes. An obvious research direction is to extend our backdoor approach to extended rules and investigate on the effect of normalizations [BGJ14] on the size of a backdoor. An interesting question regarding preprocessing is whether backdoor detection into the various acyclicity-based target classes admits a polynomial kernel. Since backdoor evaluation does not admit a polynomial kernel, it would be interesting to investigate whether there is a kernel that consists of several polynomially sized kernels instead of a single polynomial kernel. A direction for future work is to consider the parameterized complexity of the co-NP check for disjunctive answer set programs when parameterized by various different parameters.

From the practical perspective it would be interesting to study whether and how backdoors can be used to control modern heuristics in Asp solvers to obtain a speed-up. Even so we expect that a practically competitive heuristic is not purely based on backdoors, one might use backdoors to improve heuristics when taking other aspects of Asp solving into account. An empirical study following these considerations is subject of current research.

Our complexity barrier breaking reduction provides an alternative approach to efficiently solve reasoning problems on disjunctive answer set programs if the instance has a small backdoor to normality. Thus another interesting research direction is to consider whether the approach can be of practical use, at least for certain classes of instances.

We think that it would also be interesting to rigorously determine various parameters of known benchmark instances. Finally, an interesting research task would be to invent new structural parameters that correlate with practical hardness of known benchmark instances.

List of Figures

2.1 Illustration of the Polynomial Hierarchy 16
2.2 Illustration of the relationship of parameterized complexity classes 18

3.1 Exploit pattern of ASP backdoors 30

4.1 Negation dependency graph N_p of the program P from Example 2.1 38
4.2 Dependency digraph D_p and dependency graph U_p of program P from Example 2.1 40
4.3 Relationship between classes of programs with respect to their generality . . 42
4.4 Known complexity of the problem STRONG $\mathcal{C}$-BACKDOOR DETECTION 45
4.5 Known complexity of the problem DELETION $\mathcal{C}$-BACKDOOR DETECTION . . 46

7.1 Illustration of truth assignment reducts of the program P and the strong **Horn**-backdoors X and Y from Example 7.2 69
7.2 A **Horn**-backdoor tree $BT = (T, \chi)$ of program P^n from Example 7.2 70

8.1 Relationship between ASP parameters when restricted to normal programs . 88
8.2 Incidence graph I_P of program P from Example 2.1 97

9.1 Illustration of our complexity barrier breaking reductions for the main reasoning problems of disjunctive ASP using backdoors to normality 112

List of Tables

10.1 Size of smallest strong **Horn**-backdoors for various benchmark sets 128
10.2 Size of smallest strong **Normal**-backdoor for various benchmark sets 129

List of Algorithms

7.1 $\mathcal{C}$-BdTreeComp(P, g, l) 78

9.1 MinCheck(P, X_1, M) 111

Index

$\mathrm{AS}(P)$, 8
$\mathrm{AS}(P, BT)$, 72
$\mathrm{AS}(P, X)$, 26
$\mathrm{AS}(P, \tau)$, 72
$\mathrm{at}(BT)$, 70
$\mathrm{at}(P)$, 7
$\mathrm{at}(r)$, 7
BT, 70
$B^+(r)$, 7
$B^-(r)$, 7
$\mathrm{Constr}(P)$, 8
$\mathrm{DH}(P)$, 8
F_τ, 12
$H(r)$, 7
H_p, 55, 119
I_P, 97
$LM(P)$, 9
$L[p^\uparrow]$, 55
$L[p]$, 55
$L[p]_N$, 55
N_p, 38
$P - X$, 25
$P_{X_1 \subseteq X}$, 110
P_τ, 25
$T_P(A)$, 9
U_p, 40
$X_1(t)$, 70
$\#(a)$, 113
$\#(r)$, 113
$\mathcal{O}(\cdot)$, 15
$\|I\|$, 15, 18
$|P|$, 7
Γ_0, 71
Γ_k, 71
$\textsc{MinCheck}(P, X_1, M)$, 114
NP, 62
$\Omega(\cdot)$, 15
Π_2^p, 15
Π_3^p, 62
$\mathrm{SR}(P, M)$, 113
S, 71
Σ, 15
Σ_3^p, 62
Σ_0, 71
Σ_k, 71
$\chi(t)$, 70
co-NP/poly, 62
$(\mathrm{db}_{\mathcal{C}}^{\downarrow})^{\uparrow}(P)$, 54
$\mathrm{db}_{\mathcal{C}}$, 54, 86
$\mathrm{gs}(BT)$, 71
#badEvenCycles(P), 102
#headCycles(P), 105
$\#\mathrm{leaves}(BT)$, 71
#neg, 54, 88
#non-Horn, 88
#posCycles, 104
$\mathrm{cluster}(P)$, 101
$\mathrm{cs}(G)$, 101
$\mathrm{cyclecut}(P)$, 101
$\mathrm{deptw}(P)$, 98
$\mathrm{fw}(G)$, 101
$\mathrm{fw}(P)$, 93
$\mathrm{inctw}(P)$, 97
$\mathrm{lc}(G)$, 101
$\mathrm{lstr}(P)$, 91
$\mathrm{wfw}(P)$, 93

$\mathrm{sb}_{\mathcal{C}}$, 54, 86
$\|P\|$, 7
τ, 12
$\tau_{\chi(t)}$, 70
$\mathrm{tw}(A_P)$, 101
$\mathrm{tw}(G)$, 96
$\mathrm{tw}(I_P)$, 97
$\mathrm{tw}(U_p)$, 98
$p \bowtie q$, 84
$p \preceq q$, 84
$p \sim q$, 84
$p^{\uparrow}$, 54
$p^{\downarrow}$, 53
$q \preceq p$, 84
2^X, 12

Acyc, 41, 43
algorithm
  Dowling and Gallier's, 113
  fpt, 18, *see also* fixed-parameter tractable
  MinCheck(P, X_1, M), 111
  non-deterministic, 15
  polynomial-time, 15
  quadratic, 113
  quasilinear-time, 108
answer set, 8
  candidates, 28, 112
Asp parameter, 53
  #badEvenCycles, 102–104, 106
  #headCycles, 105, 106
  #neg, 88–92, 96, 98, 99, 102–104, 106
  #non-Horn, 88–92, 96, 98, 99, 102–104, 106
  #posCycles, 104, 106
  cluster, 101–104, 106
  cyclecut, 101–104, 106
  $\mathrm{db}_{\mathcal{C}}$, 86
    Horn, 86, 87, 92, 95, 98, 99, 102, 104, 106
    no-BC, 87, 92, 95, 98, 99, 102–104, 106
    no-EC, 102–104, 106
    no-BEC, 87, 92, 95, 98, 99, 102–104, 106
    no-C, 87, 90, 92, 93, 98, 102–104, 106
    no-DBC, 87, 92, 98, 99, 102–104, 106
    no-DBEC, 87, 98, 99, 102–104, 106
    no-DC, 87, 90, 92, 95, 98, 99, 102–104, 106
    no-DC2, 87, 90, 92, 95, 98, 99, 102–104, 106
    no-DEC, 87, 90, 92, 95, 98, 99, 102–104, 106
    no-EC, 87, 90, 92, 95, 98
    Normal, 86, 87
deptw (dependency treewidth), 98, 99, 103, 104, 106
dominates, 84, 86
fw (feedback width), 91–95, 101
incomparable, 84
inctw (incidence treewidth), 97–99, 102, 103, 106
lift, 54
lstr (level of stratifiability), 91, 92, 96, 98, 99, 102–104, 106
normal, 53
number
  of bad even cycles, *see* #badEvenCycles
  of distinct negative literals, *see* #neg
  of head cycles, *see* #headCycles
  of non-Horn rules, *see* #neg
  of positive cycles, *see* #posCycles
of cycle cutset decomposition, *see* cyclecut
of tree clustering, *see* cluster
relationship, 88
$\mathrm{sb}_{\mathcal{C}}$, 86
  Horn, 86
  Normal, 86
similar, 84

strictly dominates, 84
wfw (weak feedback width), 91–96, 98, 99, 103, 104, 106
AspFull, 10, 31, 39, 62, 73, 80, 84
AspReason, 10
assignment, *see* truth assignment
atom, 7

backdoor
approach, 24
detection, 24, *see also* strong $\mathcal{C}$-backdoor, deletion $\mathcal{C}$-backdoor
evaluation, 24
tree, 71, *see also* $\mathcal{C}$-backdoor tree
Horn formulas, 71
$\mathcal{C}$-backdoor tree, 71, 73, 75, 76, 80
smallest, 76
$\mathcal{C}$-Backdoor Tree Asp Check(GS), 73
Horn, 73
no-BC, 73
no-BEC, 73
no-C, 73
no-DBC, 73
no-DBEC, 73
no-DC, 73
no-DC2, 73
no-EC, 73
Normal, 73
$\mathcal{C}$-Backdoor-Tree Detection(GS,Leaves), 77, 79, 80
bi-kernel, 60
bi-kernelization, 60
binary decision tree, *see* tree
Bound$[p]$, 55
Bound$[p]_{\mathrm{N}}$, 55
Brave Reasoning, 9, 48, 62, 73, 80

Checking, 9, 48, 62, 73, 80
child, 22
Clasp, 126

class of programs
closed under the union of disjoint copies, 85
enumerable, 31, *see also* enumerable
hereditary, 26, 42, 75, 76, 86
Horn, 7
incomparable, 42
linear-time enumerable, 31, *see also* enumerable
no-BC, 41
no-BEC, 41
no-C, 41
no-DBC, 41, 43
no-DBEC, 41, 43
no-DC, 41
no-DEC, 41
no-EC, 41
Normal, 8, 42
Strat, 43
clause, 12
cliquewidth, 100
complexity class
$\#P$, 16
co-NP, 15
co-para-NP, 45, 49, 109
FPT, 18
NC^1/poly, 14
non-uniform, 16
NP, 15
NP/poly, 16
P, 15
para-C, 19
para-NP, 19, 109
Π_2^p, 16
Π_3^p, 16
PSPACE, 17
Σ_2^p, 16
Σ_3^p, 16
W[1], 19
W[2], 19, 43, 44
XP, 19, 49
compressible, 62
`ConformantPlanning`, 127

Consistency, 9, 48, 62, 73, 80
Counting, 10, 48, 62, 73, 80
CPLEX, 126
cycle, 21
  bad, 41
  chord, 100
  directed, 22, 40
  even, 41
  good, 41
  head-cycle, 105
  odd, 41
  transversal, 47
  W-cycle, 47

$\mathcal{D}$-$\mathcal{A}$cyc, 41
deletion $\mathcal{C}$-backdoor, 26, 119, 121, 129
  small, 129
  smallest, 48, 128
Deletion $\mathcal{C}$-Backdoor Detection, 32
  Horn, 46, 61
  no-BC, 46
  no-BEC, 46
  no-C, 46
  no-DBC, 46
  no-DBEC, 46, 49
  no-DC, 46, 61
  no-DEC, 46, 49
  no-EC, 46
  Normal, 61
deletion backdoor
  CNF formula, 24
  program, 26, *see also* deletion $\mathcal{C}$-backdoor
dependency digraph, 40, *see also* dependency graph
dependency graph, 40
  directed, 40
  head, 55, 119
  interaction graph, 100
  negation, 38, 39, 125
  positive, 120
  undirected, 40
digraph, 21
  dependency, 40, *see also* dependency graph
  strongly connected component, 22
  subdigraph, 22
  underlying graph, 22
Directed Path via a Node, 45
domination lattice, 88

edge, 21
  directed, 21
Emden-Kowalski operator, 9
encoding
  $F^{\leq}(\vec{x},\vec{y})$, 116
  $F^{(\mathrm{a})}$, 117
  $F^{(\mathrm{b})}$, 117
  $F^{(\mathrm{c})}$, 117
  $F^{(\mathrm{d})}$, 117
  $F^{+1}(\vec{x},\vec{y})$, 116
  F^{+}, 116
  $F^{=0}(\vec{x})$, 116
  F^{MinCheck}, 114
  F^{cond}, 114
  F^{level}, 114, 116
  F^{mod}, 118
  $F^{\text{stays-body}}_{r,b}$, 115
  $F^{\text{stays-const}}_{r}$, 115
  $F^{\text{stays-head}}_{r,a}$, 115
  $F^{\text{stays-pos}}_{r}$, 115
  F^{stays}, 114, 115
  F^{subset}, 114
  F^{supp}, 114, 115
  $Q_{\mathrm{Brave}}(a)$, 109, 118
  $Q_{\mathrm{Skept}}(a)$, 109, 118
enumerable, 31, 73, 80
  linear-time, 31, 38, 43
  polynomial-time delay, 31
Enum, 10, 48, 62, 73, 80
Enum$[p]_{\mathrm{N}}$, 55
$\exists\forall$-QBF-Sat, 13, 17

`Factoring`, 127
False, 29, 74, 111
Feedback Vertex Set, 62
feedback vertex set, 47, 62

directed, 47
mixed, 47
subset, 47
W-feedback vertex set, 47
feedback width, *see* Asp parameter
fixed-parameter tractable, 17, 28, 31, 33, 39, 47, 55, 61, 73, 79, 80, 84, 88, 89, 91, 93, 97, 101, 102
∀∃-Qbf-Sat, 13, 17
formula
CNF, 12
conjunctive normal form, 12
Horn, 71
matrix, 13
prenex normal form, 13
QBF, 12
quantified Boolean formula, 12
quantifier block, 13
satisfiable, 12
fpt, 56, *see also* fixed-parameter tractable

graph
induced subgraph, 21
bipartite, 21
chordal, 100
clique, 21
complete, 21
connected component, 21
dependency, 40, *see also* dependency graph
directed, *see* digraph
head dependency, 55, *see also* dependency graph
incidence graph, 97
interaction, 100, *see also* dependency graph
negation dependency, 38, *see also* dependency graph
subdigraph
induced, 22
subgraph, 21
tree, 21
triangulated, *see* chordal
undirected, 21
undirected dependency, 40, *see also* dependency graph
vertex cover, 38, *see also* vertex cover
`GraphColoring`, 127
Gurobi, 126

`HanoiTower`, 127
head dependency graph, 55, *see also* dependency graph
Hierarchy
Polynomial, 15
Weft, 19
Hitting Set, 19, 44

Ilp, 125
incidence graph, 97, *see also* graph
instance, 15
main part, 18
no-instance, 15
parameter, 18
yes-instance, 15
integer linear programming, 125
interaction graph, 100, *see also* dependency graph
interaction treewidth, 101, *see also* Asp parameter

kernel, 60, 61
bi, 60
linear, 61
polynomial, 60–62
quadratic, 61
kernelization, 60, 61
bi, 60
Buss, 80
loss-free, 79, 80
`KnightTour`, 127

`Labyrinth`, 127
leaf, 22
length
of a cycle, 21, 47
of a path, 21
of the longest cycle, 101

level numbering, 113
lift, *see* self-lifting
literal
  of a formula, 12
  of a program, 7
local search, 125
loop formulas, 120
lp2normal, 126
LS, 125

Max-Lits-Sat, 20
Max-Occurrence-Consistency, 20
MinCheck(P, X_1, M), 110, 111
minimal
  strong $\mathcal{C}$-backdoor, 25
  deletion **Normal**-backdoor, 54
  vertex cover, 56
`MinimalDiagnosis`, 127
model, 8
  answer set, 8
  least model, 9
  minimal, 110
  stable model, 8
  supported, 113
modular, 14
`MSS/MUS`, 127

negation dependency graph, 38, *see also* dependency graph
node, 21, 22
number
  of answer sets, 10
  of atoms, 61
  of bad even cycles, *see* Asp parameter
  of distinct negative literals, *see* Asp parameter
  of head cycles, *see* Asp parameter
  of leaves, 71
  of non-Horn rules, *see* Asp parameter
  of positive cycles, *see* Asp parameter
  of rules, 7
  of variables, 62

parameterized by, 39
  Asp parameter p, 55
  feedback width, 93, *see also* Asp parameter
  Gallo-Scutellà parameter, 73
  Gallo-Scutellà parameter + size of $\mathcal{C}$-backdoor tree, 73, 77, 80
  incidence treewidth, 97, *see also* Asp parameter
  level of stratifiability, 91, *see also* Asp parameter
  maximum number of literals, 20
  maximum number of occurrences, 20
  number of bad even cycles, 102
  number of distinct negative literals, 88, *see also* Asp parameter, 89
  number of non-Horn rules, 88, *see also* Asp parameter, 89
  number of positive cycle, 104
  number of variables, 62
  size of strong $\mathcal{C}$-backdoor, 28, 31
  size of cycle cutset decomposition, 101, *see also* Asp parameter
  size of tree clustering, 101, *see also* Asp parameter
  solution size, 32, 60
  weak feedback width, 93, *see also* Asp parameter
parameterized complexity, 17
path
  directed, 22
  length, 21
predecessor, 22
preprocessing, 60
problem
  admits polynomial kernel, 60
  C-complete, 15
  C-hard, 15
  compressible, 60, 62
  co-NP-hard, 104, 105
  co-para-NP-complete, 19, 109
  co-para-NP-hard, 45

decidable, 61
decision, 15
fixed-parameter tractable, 18
NP-hard, 104, 105
parameterized, 18
para-NP-complete, 19, 109
para-Σ_2^P-complete, 20
Π_2^p-hard, 121
Σ_2^p-hard, 121
solvable
non-deterministically, 15
polynomial-time, 15
unparameterized version, 20
W[2]-hard, 43, 44
program, 7
constraint-free, 7
cycle, 41
definite Horn, 7, 113
disjoint copy, 85
disjunctive, 7
head-cycle-free, 105
Horn, 7
logic, 7
normal, 7
positive, 7
stratified, 43
subprogram, 8
tight, 104, 120

QBF-SAT, 13

RandomNonTight, 127
RandomQBF, 127
reduct
backdoor, 110
Gelfond-Lifschitz (GL), 8
truth assignment reduct, 25
reduction
complexity barrier breaking, 108
Cook, 15
fpt, 18
Karp, 15
many-to-one, 15, 18
polynomial-time, 15, 60
quasilinear-time, 108
self-reduction, 32
Turing, 15
RLP, 127
root, 22
rule, 7
body
negative, 7
positive, 7
constraint, 7
constraint-free, 7
definite Horn, 7
head, 7
Horn, 7
integrity, 7
normal, 7
tautological, 7

SageMath, 126
SAT, 12, 19, 109, 125
SAT[VARS], 62
satisfies
set of atoms, 8
truth assignment, 12
self-lifting, 54, 89, 91, 93, 106
self-transformation, *see* reduction
separating example
P_{11}^n, 85, 87, 89, 92, 95, 106
P_1^n, 85, 89, 90
P_2^n, 85, 89, 90
P_{31}^n, 85, 86, 89–91, 93, 94, 96
P_{32}^n, 85, 86, 89, 91, 94, 97, 98, 104, 105
P_{33}^n, 85, 86, 94, 96
P_{34}^n, 85, 86, 94, 96
P_{35}^n, 85, 89, 91, 94, 96
P_3^n, 93
P_4^n, 85, 86, 89, 91–96, 103
P_{51}^n, 85, 87, 89, 91, 94, 97–99, 101–106
P_{52}^n, 85, 89, 91, 94, 103
P_{53}^n, 85, 101, 104, 105
P_{54}^n, 85, 87, 89, 90
P_6^n, 85, 87, 89, 91, 94, 97–99

P_7^n, 85, 89, 91, 94, 97, 99, 103–105
$P_8^{m,n}$, 85, 87, 89, 92, 95, 98, 101–105
P_9^n, 85, 87, 89, 92, 95, 103
P^n, 87
shifting, 118
size
of an instance, 18
of a strong $\mathcal{C}$-backdoor, 76
of a $\mathcal{C}$-backdoor tree, 76
of a clique, 101
of a cycle cutset decomposition, 101
of a deletion $\mathcal{C}$-backdoor, 54
of a parameterized instance, 15
of a program, 7
of a strong $\mathcal{C}$-backdoor, 54, 76
of a tree clustering, 101
Skeptical Reasoning, 10, 48, 62, 73, 80
`Solitaire`, 127
`StrategicCompanies`, 127
stratification, 43
level of stratifiability, *see* Asp parameter
strong $\mathcal{C}$-backdoor, 25, 75, 76, 119, 121
minimal, 79
smallest, 25, 76
Strong $\mathcal{C}$-Backdoor Asp Check, 28
Strong $\mathcal{C}$-Backdoor Detection, 32
Horn, 39, 45
no-BC, 43, 45, 47
no-BEC, 43, 45, 47
no-C, 43, 45, 47
no-DBC, 43–45, 47
no-DBEC, 43–45
no-DC, 43–45, 47
no-DEC, 43–45
no-EC, 43, 45, 47
strong backdoor
of a CNF formula, 24
of a program, 25, *see also* strong $\mathcal{C}$-backdoor
successor, 22

transversal, *see* cycle
tree, 21
backdoor, 71, *see also* $\mathcal{C}$-backdoor tree
binary, 22, 70
binary decision, 70, 72
treewidth, 96, *see also* Asp parameter
True, 29, 74, 111
truth assignment, 12
corresponding, 70

undirected dependency graph, 40, *see also* dependency graph
UnSat, 12, 20, 109

variable, 12
propositional, 12
Vertex Cover, 60, 61, 80
vertex, 21
negative, 40
vertex cover, 38, 55, 60, 61, 80, 119, 125

W-cycle transversal, *see* cycle
weak feedback width, *see* Asp parameter
width
clique, 100
dependency treewidth, 98, *see also* Asp parameter
feedback width, 93, *see also* Asp parameter
incidence treewidth, 97, *see also* Asp parameter
interaction treewidth, 101, *see also* Asp parameter
weak feedback width, 93, *see also* Asp parameter

Bibliography

[AB02] Abdelwaheb Ayari and David A. Basin. QUBOS: Deciding Quantified Boolean Logic Using Propositional Satisfiability Solvers. In: *Proceedings of the 4th International Conference on Formal Methods in Computer-Aided Design (FMCAD'02)*. Ed. by Mark Aagaard and John W. O'Leary. Vol. 2517. Lecture Notes in Computer Science. Portland, OR, USA: Springer Verlag, Nov. 2002, pp. 187–201. ISBN: 3-540-00116-6. DOI: 10.1007/3-540-36126-X_12 (cit. on pp. 107, 109).

[AB09] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, Cambridge, 2009, p. 594. ISBN: 9780521424264 (cit. on pp. 14, 15).

[Abu+07] Faisal N. Abu-Khzam et al. Kernelization Algorithms for the Vertex Cover Problem: Theory and Experiments. In: *Proceedings of the 6th Workshop on Algorithm Engineering & Expermients Workshop (ALENEX'04)*. Ed. by Lars Arge, Giuseppe F. Italiano, and Robert Sedgewick. New Orleans, LA, USA: Society for Industrial and Applied Mathematics (SIAM), Jan. 2007, pp. 62–69. ISBN: 0-89871-564-4 (cit. on p. 63).

[ABW88] Krzysztof R. Apt, Howard A. Blair, and Adrian Walker. Towards a theory of declarative knowledge. In: *Foundations of deductive databases and logic programming* (1988), pp. 89–148 (cit. on pp. 1, 10, 17, 40, 43).

[Adl+10] Isolde Adler et al. On the Boolean-Width of a Graph: Structure and Applications. In: *Proceedings of the 36th International Workshop on Graph Theoretic Concepts in Computer Science (WG'10)*. Ed. by Dimitrios M. Thilikos. Vol. 6410. Lecture Notes in Computer Science. Zarós, Crete, Greece: Springer Verlag, June 2010, pp. 159–170. ISBN: 978-3-642-16925-0. DOI: 10.1007/978-3-642-16926-7_16 (cit. on p. 21).

[AFT11] Albert Atserias, Johannes Klaus Fichte, and Marc Thurley. Clause-learning algorithms with many restarts and bounded-width resolution. In: *J. Artif. Intell. Res.* 40 (1 2011), pp. 353–373. ISSN: 1076-9757. DOI: 10.1613/jair.3152 (cit. on pp. 2, 6, 13).

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases: The Logical Level*. 1st. Boston, MA, USA: Addison-Wesley, 1995. ISBN: 0201537710. DOI: 10.020.153/7710 (cit. on p. 11).

[Alv+13] Mario Alviano et al. WASP: A Native ASP Solver Based on Constraint Learning. English. In: *Proceedings of the 12th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'13)*. Ed. by Pedro Cabalar and TranCao Son. Vol. 8148. Lecture Notes in Computer Science. Corunna, Spain: Springer Verlag, Sept. 2013, pp. 54–66. ISBN: 978-3-642-40563-1. DOI: 10.1007/978-3-642-40564-8_6 (cit. on pp. 1, 11).

[And+12] Benjamin Andres et al. Unsatisfiability-based optimization in clasp. In: *Technical Communications of the 28th International Conference on Logic Programming (ICLP'12)*. Ed. by Agostino Dovier and Vítor Santos Costa. Vol. 17. Leibniz International Proceedings in Informatics (LIPIcs). Budapest, Hungary: Dagstuhl Publishing, 2012, pp. 212–221. ISBN: 978-3-939897-43-9. DOI: 10.4230/LIPIcs.ICLP.2012.211 (cit. on pp. 12, 126).

[Ans+08] Carlos Ansótegui et al. Measuring the Hardness of SAT Instances. In: *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI '08)*. Ed. by Robert C. Holte and Adele E. Howe. Chicago, IL, USA: The AAAI Press, July 2008, pp. 222–228. ISBN: 978-1-57735-368-3 (cit. on pp. 1, 13).

[Ans+14] Carlos Ansótegui et al. The Fractal Dimension of SAT Formulas. English. In: *Proceedings of the 7th International Joint Conference on Automated Reasoning (IJCAR'14)*. Ed. by Stéphane Demri, Deepak Kapur, and Christoph Weidenbach. Vol. 8562. Lecture Notes in Computer Science. Held as Part of the Vienna Summer of Logic, VSL 2014. Vienna, Austria: Springer Verlag, 2014, pp. 107–121. ISBN: 978-3-319-08586-9. DOI: 10.1007/978-3-319-08587-6_8 (cit. on p. 13).

[APE12] Erdi Aker, Volkan Patoglu, and Esra Erdem. Answer set programming for reasoning with semantic knowledge in collaborative housekeeping robotics. In: *Proceedings of the 10th IFAC Symposium on Robot Control*. Ed. by Peter Korondi and Trygve Thomessen. Vol. 10. 1. Dubrovnik, Croatia: International Federation of Automatic Control, 2012, pp. 77–83. ISBN: 978-3-902823-11-3. DOI: 10.3182/20120905-3-HR-2030.00169 (cit. on p. 12).

[Bar03] Chitta Baral. *Knowledge representation, reasoning and declarative problem solving*. Cambridge, UK: Cambridge University Press, Cambridge, 2003. ISBN: 0-521-81802-8 (cit. on p. 10).

[BD94] Rachel Ben-Eliyahu and Rina Dechter. Propositional Semantics for Disjunctive Logic Programs. In: *Ann. Math. Artif. Intell.* 12.1 (1994), pp. 53–87. ISSN: 1012-2443. DOI: 10.1007/BF01530761 (cit. on pp. 1, 2, 13, 14, 21, 100, 101, 105, 122).

[BD98] Stefan Brass and Jürgen Dix. Characterizations of the Disjunctive Well-Founded Semantics: Confluent Calculi and Iterated GCWA. In: *J. Automated Reasoning* 20 (1 1998), pp. 143–165. ISSN: 0168-7433. DOI: 10.1023/A:1005952908693 (cit. on p. 9).

[BEL00] Yuliya Babovich, Esra Erdem, and Vladimir Lifschitz. Fages' Theorem and Answer Set Programming. In: *Proceedings of the 8th International Workshop on Non-Monotonic Reasoning (NMR'00)*. Ed. by Chitta Baral and Mirosław Truszczyński. arXiv:cs/0003042. 2000 (cit. on p. 120).

[Ben96] Rachel Ben-Eliyahu. A Hierarchy of Tractable Subsets for Computing Stable Models. In: *J. Artif. Intell. Res.* 5 (1996), pp. 27–52. DOI: 10.1613/jair.223 (cit. on pp. 21, 50, 57, 88, 90, 91).

[BET11] G. Brewka, T. Eiter, and M. Truszczyński. Answer set programming at a glance. In: *Communications of the ACM* 54.12 (2011), pp. 92–103. ISSN: 0001-0782. DOI: 10.1145/2043174.2043195 (cit. on p. 10).

[BF91] Nicole Bidoít and Christine Froidevaux. Negation by default and unstratifiable logic programs. In: *Theoretical Computer Science* 78.1 (1991), pp. 85–112. ISSN: 0304-3975. DOI: 10.1016/0304-3975(51)90004-7 (cit. on pp. 17, 109, 123).

[BG09] Jørgen Bang-Jensen and Gregory Z. Gutin. *Digraphs: theory, algorithms and applications*. Springer Monographs in Mathematics. London, UK: Springer Verlag, 2009, p. 798. ISBN: 978-1-84800-997-4. DOI: 10.1007/978-1-84800-998-1 (cit. on p. 22).

[BG93] J. Buss and J. Goldsmith. Nondeterminism within P^*. In: *SIAM J. Comput.* 22.3 (1993), pp. 560–572. DOI: 10.1137/0222038. eprint: http://epubs.siam.org/doi/pdf/10.1137/0222038 (cit. on pp. 63, 80).

[BGJ14] Jori Bomanson, Martin Gebser, and Tomi Janhunen. Improving the Normalization of Weight Rules in Answer Set Programs. English. In: *Proceedings of the 14th European Conference on Logics in Artificial Intelligence (JELIA'14)*. Ed. by Eduardo Fermé and João Leite. Vol. 8761. Lecture Notes in Computer Science. Funchal, Madeira, Portugal: Springer Verlag, Sept. 2014, pp. 166–180. ISBN: 978-3-319-11557-3. DOI: 10.1007/978-3-319-11558-0_12 (cit. on pp. 11, 64, 126, 135).

[Bie04] Armin Biere. Resolve and Expand. In: *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*. Ed. by Holger H. Hoos and David G. Mitchell. Lecture Notes in Computer Science 3542. Vancouver, BC, Canada: Springer Verlag, May 2004, pp. 59–70. ISBN: 3-540-27829-X. DOI: 10.1007/11527695_5 (cit. on pp. 107, 109).

[Bie+09] Armin Biere et al., eds. *Handbook of Satisfiability*. Vol. 185. Frontiers in Artificial Intelligence and Applications. Amsterdam, Netherlands: IOS Press, Feb. 2009. ISBN: 978-1-58603-929-5 (cit. on pp. 1, 2, 13).

[Bin+12] Daniel Binkele-Raible et al. Kernel(s) for problems with no kernel: On out-trees with many leaves. In: *ACM Transactions on Algorithms* 8.4 (2012), p. 38. DOI: 10.1145/2344422.2344428 (cit. on pp. 64, 65).

[BJ13] Jori Bomanson and Tomi Janhunen. Normalizing Cardinality Rules Using Merging and Sorting Constructions. In: *Proceedings of the 12th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'13)*. Ed. by Pedro Cabalar and Tran Cao Son. Vol. 8148. Lecture Notes in Computer Science. Corunna, Spain: Springer Verlag, Sept. 2013, pp. 187–199. ISBN: 978-3-642-40563-1. DOI: 10.1007/978-3-642-40564-8_19 (cit. on p. 126).

[BK08] Hans L. Bodlaender and Arie M. C. A. Koster. Combinatorial Optimization on Graphs of Bounded Treewidth. In: *The Computer Journal* 51.3 (2008), pp. 255–269. DOI: 10.1093/comjnl/bxm037 (cit. on p. 97).

[BL11] Paul Bonsma and Daniel Lokshtanov. Feedback Vertex Set in Mixed Graphs. In: *Proceedings of the 12th International Symposium on Algorithms and Data Structures (WADS'11)*. Ed. by Frank Dehne, John Iacono, and Jörg-Rüdiger Sack. Vol. 6844. Lecture Notes in Computer Science. New York, NY, USA: Springer Verlag, Aug. 2011, pp. 122–133. ISBN: 978-3-642-22299-3. DOI: 10.1007/978-3-642-22300-6_11 (cit. on p. 48).

[BLM14] Rémi Brochenin, Yuliya Lierler, and Marco Maratea. Abstract Disjunctive Answer Set Solvers. In: *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI'14)*. Ed. by Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan. Vol. 263. Frontiers in Artificial Intelligence and Applications. Prague, Czech Republic: IOS Press, Aug. 2014, pp. 165–170. DOI: 10.3233/978-1-61499-419-0-165 (cit. on p. 11).

[BLS11] Armin Biere, Florian Lonsing, and Martina Seidl. Blocked Clause Elimination for QBF. English. In: *Proceedings of the 23rd International Conference on Automated Deduction (CADE'11)*. Ed. by Nikolaj Bjørner and Viorica Sofronie-Stokkermans. Vol. 6803. Lecture Notes in Computer Science. Springer Verlag, 2011, pp. 101–115. ISBN: 978-3-642-22437-9. DOI: 10.1007/978-3-642-22438-6_10 (cit. on p. 122).

[BM08] John A. Bondy and U. S. R. Murty. *Graph theory*. Vol. 244. Graduate Texts in Mathematics. New York, USA: Springer Verlag, 2008, p. 655. ISBN: 978-1-84628-969-9 (cit. on p. 22).

[BN08] E. Ben-Sasson and J. Nordstrom. Short Proofs May Be Spacious: An Optimal Separation of Space and Length in Resolution. In: *Proceedings of the IEEE 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS'08)*. Ed. by R. Ravi. Philadelphia, PA, USA: IEEE Computer Soc., Oct. 2008, pp. 709–718. ISBN: 978-0-7695-3436-7. DOI: 10.1109/FOCS.2008.42 (cit. on p. 13).

[Bod05] Hans L. Bodlaender. Discovering Treewidth. In: *Proceedings of the 31st Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'05)*. Ed. by Peter Vojtáš et al. Vol. 3381. Lecture Notes in Computer Science. Liptovský Ján, Slovakia: Springer Verlag, Jan. 2005,

pp. 1–16. ISBN: 978-3-540-24302-1. DOI: 10.1007/978-3-540-30577-4_1 (cit. on p. 96).

[Bod93] Hans L. Bodlaender. A tourist guide through treewidth. In: *Acta Cybernetica* 11.1-2 (1993), pp. 1–22 (cit. on pp. 21, 96).

[Bod97] Hans L. Bodlaender. Treewidth: Algorithmic techniques and results. In: *Proceedings of the 22nd International Symposium on Mathematical Foundations of Computer Science (MFCS'97)*. Ed. by Igor Prívara and Peter Ružička. Vol. 1295. Lecture Notes in Computer Science. Bratislava, Slovakia: Springer Verlag, Aug. 1997, pp. 19–36. ISBN: 978-3-540-63437-9. DOI: 10.1007/BFb0029946 (cit. on p. 96).

[BS14] Paul Beame and Ashish Sabharwal. Non-Restarting SAT Solvers with Simple Preprocessing can Efficiently Simulate Resolution. In: *Proceedings of the 28rd AAAI Conference on Artificial Intelligence (AAAI'14)*. Ed. by Carla E. Brodley and Peter Stone. Québec, QC, Canada: The AAAI Press, July 2014 (cit. on p. 13).

[BTY11] Hans Bodlaender, Stéphan Thomassé, and Anders Yeo. Kernel bounds for disjoint cycles and disjoint paths. In: *Theoretical Computer Science* 412.35 (2011), pp. 4570–4578. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2011.04.039 (cit. on p. 64).

[Cai+13] Shaowei Cai et al. NuMVC: An Efficient Local Search Algorithm for Minimum Vertex Cover. In: *J. Artif. Intell. Res.* 46 (2013), pp. 687–716. DOI: 10.1613/jair.3907 (cit. on p. 126).

[CH85] Ashok K. Chandra and David Harel. Horn clause queries and generalizations. In: *J. Logic Programming* 2.1 (1985), pp. 1–15. ISSN: 0743-1066. DOI: 10.1016/0743-1066(85)90002-0 (cit. on p. 43).

[Che+08] Jianer Chen et al. A Fixed-Parameter Algorithm for the Directed Feedback Vertex Set Problem. In: *J. of the ACM* 55.5 (2008), pp. 1–19. ISSN: 0004-5411. DOI: 10.1145/1411509.1411511 (cit. on p. 48).

[Chi+12] Rajesh Chitnis et al. Directed Subset Feedback Vertex Set Is Fixed-Parameter Tractable. In: *Proceedings of the 39th International on Colloquium Automata, Languages, and Programming (ICALP'12)*. Ed. by Artur Czumaj et al. Vol. 7391. Lecture Notes in Computer Science. Warwick, UK: Springer Verlag, July 2012, pp. 230–241. ISBN: 978-3-642-31593-0. DOI: 10.1007/978-3-642-31594-7_20 (cit. on p. 48).

[Chv83] Vasek Chvatal. *Linear Programming*. Vol. 15. Series of Books in the Mathematical Sciences. New York, USA: W. H. Freeman and Company, New York, 1983, p. 393. ISBN: 0716715872. DOI: 10.1002/net.3230150310 (cit. on p. 126).

[CI05] Hubie Chen and Yannet Interian. A model for generating random quantified Boolean formulas. In: *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*. Ed. by Leslie Pack Kaelbling and Alessandro Saffiotti. Edinburgh, Scotland, UK: Professional Book Center, Aug. 2005, pp. 66–71. ISBN: 0938075934 (cit. on p. 127).

[CIR14] Francesco Calimeri, Giovambattista Ianni, and Francesco Ricca. The third open answer set programming competition. In: *Theory Pract. Log. Program.* 14 (01 Jan. 2014), pp. 117–135. ISSN: 1475-3081. DOI: 10.1017/S1471068412000105 (cit. on pp. 1, 126, 127).

[CKJ01] Jianer Chen, Iyad A. Kanj, and Weijia Jia. Vertex cover: further observations and further improvements. In: *J. Algorithms* 41.2 (2001), pp. 280–301. DOI: 10.1006/jagm.2001.1186 (cit. on p. 63).

[CKX06] J. Chen, I. Kanj, and G. Xia. Improved parameterized upper bounds for vertex cover. In: *Proceedings of the 31st International Symposium on Mathematical Foundations of Computer Science (MFCS'06)*. Ed. by Rastislav Královič and Paweł Urzyczyn. Vol. 4162. Lecture Notes in Computer Science. Stará Lesná, Slovakia: Springer Verlag, Aug. 2006, pp. 238–249. ISBN: 978-3-540-37791-7. DOI: 10.1007/11821069_21 (cit. on pp. 107, 119).

[CKX10] Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. In: *Theoretical Computer Science* 411.40–42 (Sept. 2010), pp. 3736–3756 (cit. on pp. 39, 60, 61, 63, 65).

[CL94] Marco Cadoli and Maurizio Lenzerini. The Complexity of Propositional Closed World Reasoning and Circumscription. In: *J. of Computer and System Sciences* 48.2 (1994), pp. 255–310. DOI: 10.1016/S0022-0000(05)80004-2 (cit. on pp. 17, 28).

[Cla78] Keith L. Clark. Negation as failure. In: *Logic and Data Bases* 1 (1978). Ed. by H. Gallaire and J. Minker, pp. 293–322 (cit. on pp. 13, 57, 120).

[CO00] Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. In: *Discr. Appl. Math.* 101.1–3 (2000), pp. 77–114. ISSN: 0166-218X. DOI: 10.1016/S0166-218X(99)00184-5 (cit. on p. 21).

[Coo+13] William Cook et al. A hybrid branch-and-bound approach for exact rational mixed-integer programming. In: *Mathematical Programming Computation* 5.3 (2013), pp. 305–344. ISSN: 1867-2949. DOI: 10.1007/s12532-013-0055-6 (cit. on p. 126).

[Coo71] Stephen A. Cook. The Complexity of Theorem-Proving Procedures. In: *Proceedings of the 3rd Annual Symposium on Theory of Computing (ACM STOC'71)*. Ed. by Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman. Shaker Heights, OH, USA: Assoc. Comput. Mach., New York, 1971, pp. 151–158. DOI: 10.1145/800157.805047 (cit. on pp. 13, 17).

[CR93] Alain Colmerauer and Philippe Roussel. The Birth of Prolog. In: *Proceedings on the 2nd ACM SIGPLAN Conference on the History of Programming Languages (HOPL-II)*. Ed. by John A. N. Lee and Jean E. Sammet. Cambridge, MA, USA: Assoc. Comput. Mach., New York, Mar. 1993, pp. 37–52. ISBN: 0-89791-570-4. DOI: 10.1145/154766.155362 (cit. on p. 10).

[CSC10] Shaowei Cai, Kaile Su, and Qingliang Chen. EWLS: A New Local Search for Minimum Vertex Cover. In: *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI'10)*. Ed. by Nestor Rychtyckyj and Daniel G. Shapiro. Atlanta, GA, USA: The AAAI Press, July 2010, pp. 45–50 (cit. on p. 126).

[CSS11] Shaowei Cai, Kaile Su, and Abdul Sattar. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. In: *Artificial Intelligence* 175 (9–10 2011), pp. 1672–1696. DOI: 10.1016/j.artint.2011.03.003 (cit. on p. 126).

[Cyg+13] Marek Cygan et al. Subset Feedback Vertex Set Is Fixed-Parameter Tractable. In: *SIAM J. Discrete Math.* 27.1 (2013), pp. 290–309. DOI: 10.1137/110843071 (cit. on pp. 47, 48).

[Cyg+15] Marek Cygan et al. *Parameterized Algorithms*. Unpublished. Springer Verlag, 2015, pp. XVII, 613. ISBN: 978-3-319-21274-6. DOI: 10.1007/978-3-319-21275-3 (cit. on p. 20).

[Dan+01] Evgeny Dantsin et al. Complexity and expressive power of logic programming. In: *ACM Computing Surveys (CSUR)* 33.3 (2001), pp. 374–425. ISSN: 0360-0300. DOI: 10.1145/502807.502810 (cit. on p. 17).

[Den00] Marc Denecker. Extending Classical Logic with Inductive Definitions. In: *Proceedings of the 1st International Conference on Computational Logic (CL'00)*. Ed. by John W. Lloyd et al. Vol. 1861. Lecture Notes in Computer Science. London, UK: Springer Verlag, July 2000, pp. 703–717. ISBN: 3-540-67797-6. DOI: 10.1007/3-540-44957-4_47 (cit. on p. 11).

[Den+09] Marc Denecker et al. The Second Answer Set Programming Competition. In: *Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*. Ed. by Esra Erdem, Fangzhen Lin, and Torsten Schaub. Vol. 5753. Lecture Notes in Computer Science. Potsdam, Germany: Springer Verlag, Sept. 2009, pp. 637–654. DOI: 10.1007/978-3-642-04238-6_75 (cit. on pp. 1, 126).

[DF13] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. London, UK: Springer Verlag, 2013. ISBN: 978-1-4471-5558-4. DOI: 10.1007/978-1-4471-5559-1 (cit. on pp. 3, 18–20, 28, 32, 47, 77).

[DF99] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. New York, NY, USA: Springer Verlag, 1999. ISBN: 978-1-4612-6798-0. DOI: 10.1007/978-1-4612-0515-9 (cit. on pp. 17–20, 32, 47, 77).

[DFS99] Rodney G. Downey, Michael R. Fellows, and Ulrike Stege. Parameterized Complexity: A Framework for Systematically Confronting Computational Intractability. In: *Contemporary Trends in Discrete Mathematics: From DIMACS and DIMATIA to the Future*. Ed. by Ronald L. Graham et al. Vol. 49. DIMACS: Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, 1999, pp. 49–99. ISBN: 0-8218-0963-6 (cit. on pp. 20, 60, 61, 63).

[DG84] William F. Dowling and Jean H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. In: *J. Logic Programming* 1.3 (1984), pp. 267–284. ISSN: 0743-1066. DOI: 10.1016/0743-1066(84)90014-1 (cit. on pp. 9, 30, 110).

[DGS07] Bistra N. Dilkina, Carla P. Gomes, and Ashish Sabharwal. Tradeoffs in the Complexity of Backdoor Detection. In: *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP'07)*. Ed. by Christian Bessiere. Vol. 4741. Lecture Notes in Computer Science. Providence, RI, USA: Springer Verlag, Sept. 2007, pp. 256–270. ISBN: 978-3-540-74969-1. DOI: 10.1007/978-3-540-74970-7_20 (cit. on pp. 33, 34).

[DGS09] Bistra Dilkina, Carla P. Gomes, and Ashish Sabharwal. Backdoors in the Context of Learning. English. In: *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT'09)*. Ed. by Oliver Kullmann. Vol. 5584. Lecture Notes in Computer Science. Swansea, UK: Springer Verlag, June 2009, pp. 73–79. ISBN: 978-3-642-02776-5. DOI: 10.1007/978-3-642-02777-2_9 (cit. on p. 33).

[DGS14] Bistra N. Dilkina, Carla P. Gomes, and Ashish Sabharwal. Tradeoffs in the complexity of backdoors to satisfiability: dynamic sub-solvers and learning during search. In: *Ann. Math. Artif. Intell.* 70.4 (2014), pp. 399–431. ISSN: 1012-2443. DOI: 10.1007/s10472-014-9407-9 (cit. on pp. 33, 34).

[Die12] Reinhard Diestel. *Graph Theory, 4th Edition*. Vol. 173. Graduate Texts in Mathematics. Springer Verlag, 2012, p. 410. ISBN: 978-3-642-14278-9 (cit. on p. 22).

[DLL62] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. In: *Commun. ACM* 5.7 (July 1962), pp. 394–397. ISSN: 0001-0782. DOI: 10.1145/368273.368557 (cit. on p. 13).

[DLV12] DLVSYSTEM s.r.l. *DLV*. http://www.dlvsystem.com/dlv. Dec. 2012 (cit. on pp. 11, 122).

[DM14] Holger Dell and Dieter van Melkebeek. Satisfiability Allows No Nontrivial Sparsification unless the Polynomial-Time Hierarchy Collapses. In: *J. of the ACM* 61.4 (2014), p. 23. DOI: 10.1145/2629620 (cit. on p. 64).

[DOS12] Wolfgang Dvořák, Sebastian Ordyniak, and Stefan Szeider. Augmenting tractable fragments of abstract argumentation. In: *Artificial Intelligence* 186 (2012), pp. 157–173. ISSN: 0004-3702. DOI: 10.1016/j.artint.2012.03.002 (cit. on pp. 2, 33).

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. In: *J. of the ACM* 7.3 (1960), pp. 201–215. DOI: 10.1145/321033.321034 (cit. on p. 13).

[Dre+08a] Christian Drescher et al. Conflict-Driven Disjunctive Answer Set Solving. In: *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR'08)*. Ed. by Gerhard Brewka and Jérôme Lang. Sydney, NSW, Australia: The AAAI Press, Sept. 2008, pp. 422–432. ISBN: 978-1-57735-384-3 (cit. on pp. 11, 104, 122).

[Dre+08b] Christian Drescher et al. Heuristics in Conflict Resolution. In: *Proceedings of the 12th International Workshop on Nonmonotonic Reasoning (NMR'08)*. Ed. by Maurice Pagnucco and Michael Thielscher. Vol. UNSW-CSE-TR-0819. The University of New South Wales, 2008, pp. 141–149 (cit. on p. 126).

[DS14a] Ronald DeHaan and Stefan Szeider. Fixed-Parameter Tractable Reductions to SAT. In: *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT'14)*. Ed. by Carsten Sinz and Uwe Egly. Vol. 8561. Lecture Notes in Computer Science. Held as Part of the Vienna Summer of Logic, VSL 2014. Vienna, Austria: Springer Verlag, July 2014, pp. 85–102. ISBN: 978-3-319-09283-6. DOI: 10.1007/978-3-319-09284-3_8 (cit. on p. 134).

[DS14b] Ronald DeHaan and Stefan Szeider. The Parameterized Complexity of Reasoning Problems Beyond NP. In: *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR'14)*. Ed. by Chitta Baral, Giuseppe De Giacomo, and Thomas Eiter. Held as Part of the Vienna Summer of Logic, VSL 2014. Full version CoRR: 1312.1672. Vienna, Austria: The AAAI Press, July 2014. ISBN: 978-1-57735-657-8 (cit. on pp. 20, 21, 108, 122, 134).

[Dun07] Paul E. Dunne. Computational properties of argument systems satisfying graph-theoretic constraints. In: *Artificial Intelligence* 171.10–15 (2007), pp. 701–729. ISSN: 0004-3702. DOI: 10.1016/j.artint.2007.03.006 (cit. on p. 96).

[EF99] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite model theory*. 2nd. Perspectives in Mathematical Logic. Springer Verlag, 1999. ISBN: 978-3-540-60149-4 (cit. on p. 116).

[EG95] Thomas Eiter and Georg Gottlob. On the computational cost of disjunctive logic programming: Propositional case. In: *Ann. Math. Artif. Intell.* 15.3–4 (1995), pp. 289–323. DOI: 10.1007/BF01536399 (cit. on pp. 1, 16, 121).

[Egl+00] Uwe Egly et al. Solving Advanced Reasoning Tasks using Quantified Boolean Formulas. In: *Proceedings of the 17th Conference on Artificial Intelligence (AAAI'00)*. Ed. by Henry Kautz and Bruce Porter. Austin, TX, USA: The AAAI Press, July 2000, pp. 417–422. ISBN: 978-0-262-51112-4 (cit. on pp. 108, 122).

[Egl+01] Uwe Egly et al. Computing Stable Models with Quantified Boolean Formulas: Some Experimental Results. In: *Proceedings of the 1st International Workshop on Answer Set Programming (ASP'01), Towards Efficient and Scalable Knowledge Representation and Reasoning*. Ed. by Alessandro Provetti and Tran Cao Son. AAAI Spring Symposium SS-01-01. Stanford, CA, USA: The AAAI Press, Mar. 2001 (cit. on p. 122).

[EIK09] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. Answer Set Programming: A Primer. In: *Reasoning Web. Semantic Technologies for Information Systems – 5th International Summer School*. Ed. by Sergio Tessaris et al. Vol. 5689. Lecture Notes in Computer Science. Brixen-Bressanone, Italy: Springer Verlag, Aug. 2009, pp. 40–110. ISBN: 978-3-642-03753-5. DOI: 10.1007/978-3-642-03754-2_2 (cit. on p. 10).

[Eit+04] T. Eiter et al. Simplifying logic programs under uniform and strong equivalence. In: *Proceedings 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'04)*. Ed. by Ilkka Niemelä and Vladimir Lifschitz. Vol. 2923. Lecture Notes in Computer Science. Tempe, AZ, USA: Springer Verlag, May 2004, pp. 87–99. ISBN: 978-3-540-72199-4. DOI: 10.1007/978-3-540-24609-1_10 (cit. on p. 9).

[Eit+12] Thomas Eiter et al. Exploiting Unfounded Sets for HEX-Program Evaluation. In: *Proceedings of the 13th European Conference on Logics in Artificial Intelligence (JELIA'12)*. Ed. by Andreas Cerro Luis Fariñas Herzig and Jérôme Mengin. Vol. 7519. Lecture Notes in Computer Science. Toulouse, France: Springer Verlag, Sept. 2012, pp. 160–175. ISBN: 978-3-642-33352-1. DOI: 10.1007/978-3-642-33353-8_13 (cit. on p. 12).

[EL03] Esra Erdem and Vladimir Lifschitz. Tight Logic Programs. In: *Theory Pract. Log. Program.* 3 (4+5 July 2003), pp. 499–518. ISSN: 1475-3081. DOI: 10.1017/S1471068403001765 (cit. on p. 100).

[Fab+99] Wolfgang Faber et al. Using database optimization techniques for nonmonotonic reasoning. In: *Proceedings of the 7th International Workshop on Deductive Databases and Logic Programming (DDLP'99)*. Ed. by Ulrich Geske, Carolina Ruiz, and Dietmar Seipel. Tokyo, Japan, Sept. 1999, pp. 135–139 (cit. on pp. 59, 64).

[Fag94] Francois Fages. Consistency of Clark's completion and existence of stable models. In: *J. on Methods of Logic in Computer Science* 1.1 (1994), pp. 51–60 (cit. on pp. 2, 13, 57, 104, 113, 120, 122).

[Fal+12] Andreas Falkner et al. Testing Object-Oriented Configurators with ASP. In: *Proceedings of the Workshop on Configuration at ECAI 2012 (ConfWS'12)*. Ed. by Wolfgang Mayer and Patrick Albert. Montpellier, France, Aug. 2012, pp. 21–26 (cit. on p. 12).

[FG03] Jörg Flum and Martin Grohe. Describing parameterized complexity classes. In: *Information and Computation* 187.2 (2003), pp. 291–319. ISSN: 0890-5401. DOI: 10.1016/S0890-5401(03)00161-5 (cit. on p. 19).

[FG06] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Vol. XIV. Theoretical Computer Science. Berlin: Springer Verlag, 2006, p. 495. ISBN: 978-3-540-29952-3. DOI: 10.1007/3-540-29953-X (cit. on pp. 18–20, 61, 63, 109).

[FHW80] Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. In: *Theoretical Computer Science* 10.2 (1980), pp. 111–121. DOI: 10.1016/0304-3975(80)90009-2 (cit. on p. 45).

[Fic12] Johannes K. Fichte. The Good, the Bad, and the Odd: Cycles in Answer-Set Programs. In: *Proceedings of the 23rd European Summer School in Logic, Language and Information (ESSLLI'11) and in New Directions in Logic, Language and Computation (ESSLLI'10 and ESSLLI'11 Student Sessions, Selected Papers Series)*. Ed. by Daniel Lassiter and Marija Slavkovik. Vol. 7415. Lecture Notes in Computer Science. Springer Verlag, 2012, pp. 78–90. ISBN: 978-3-642-31466-7. DOI: 10.1007/978-3-642-31467-4_6 (cit. on pp. 4, 5).

[Fic13a] Johannes K. Fichte. Backdoors to Tractability of Answer-Set Programming. In: *Proceedings of the 27th AAAI Conference on Artificial Intelligence (Proceedings of the 18th AAAI-SIGART Doctoral Consortium) (AAAI DC'13)*. Ed. by Peter McBurney and Ayanna Howard. Bellevue, WA, USA: The AAAI Press, 2013, pp. 1662–1663 (cit. on p. 6).

[Fic13b] Johannes K. Fichte. Backdoors to Tractability of Answer-Set Programming. In: *9th ICLP Doctoral Consortium (ICLP DC) – ICLP Technical Communications on-line supplement of Theory and Practice of Logic Programming (TPLP)*. Ed. by Martin Gebser and Marco Gavanelli. Vol. 13. 4-5. 2013, pp. 1–10 (cit. on p. 6).

[Fil+14] Yuval Filmus et al. From Small Space to Small Width in Resolution. In: *Proceedings of the 31st International Symposium on Theoretical Aspects of Computer Science (STACS'14)*. Ed. by Ernst W. Mayr and Natacha Portier. Vol. 25. Leibniz International Proceedings in Informatics (LIPIcs). Lyon, France: Dagstuhl Publishing, Mar. 2014, pp. 300–311. ISBN: 978-3-939897-65-1. DOI: 10.4230/LIPIcs.STACS.2014.300 (cit. on p. 13).

[FS11] Johannes K. Fichte and Stefan Szeider. Backdoors to Tractable Answer-Set Programming. In: *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*. Ed. by Toby Walsh. Barcelona, Catalonia, Spain: AAAI Press/IJCAI, July 2011, pp. 863–868. ISBN: 978-1-57735-516-8 (cit. on pp. 3–5).

[FS11] Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. In: *J. of Computer and System Sciences* 77.1 (2011), pp. 91–106. ISSN: 0022-0000. DOI: 10.1016/j.jcss.2010.06.007 (cit. on pp. 21, 62, 64).

[FS12] Johannes K. Fichte and Stefan Szeider. Backdoors to Normality for Disjunctive Logic Programs. In: *Proceedings of the 5th International Workshop Answer Set Programming and Other Computing Paradigms (ASPOCP'12)*. Ed. by Michael Fink and Yuliya Lierler. Vol. abs/cs/1301.1391v2. CoRR. Budapest, Hungary, Sept. 2012, pp. 99–114 (cit. on p. 5).

[FS13] Johannes K. Fichte and Stefan Szeider. Backdoors to Normality for Disjunctive Logic Programs. In: *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI'13)*. Ed. by Marie des Jardins and Michael Littman. A preliminary version of the paper was presented on the Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP'12). Bellevue, WA, USA: The AAAI Press, July 2013, pp. 320–327. ISBN: 978-1-57735-615-8 (cit. on pp. 4, 5, 108, 113, 123).

[FS15a] Johannes K. Fichte and Stefan Szeider. Backdoors to Normality for Disjunctive Logic Programs. In: (2015). Extended and updated version of a paper that appeared in *Proc. of the 27th AAAI Conference on Artificial Intelligence (AAAI'13)*. Submitted. (cit. on pp. 4, 5, 108).

[FS15b] Johannes K. Fichte and Stefan Szeider. Backdoors to Tractable Answer-Set Programming. In: *Artificial Intelligence* 220 (2015). Extended and updated version of a paper that appeared in *Proc. of the 22nd International Conference on Artificial Intelligence (IJCAI'11)*., pp. 64–103. ISSN: 0004-3702. DOI: 10.1016/j.artint.2014.12.001 (cit. on pp. 3–5, 23, 37, 53, 59, 84).

[FTW15] Johannes K. Fichte, Mirosław Truszczyński, and Stefan Woltran. Dual-normal programs – The forgotten class. In: *Theory Pract. Log. Program.* (2015). CoRR: 1507.05388. To appear in Proceedings of the 31st International Conference on Logic Programming (ICLP'15). (cit. on p. 6).

[Gar12] Marco Gario. Horn backdoor detection via Vertex Cover: Benchmark Description. In: *Proceedings of SAT Challenge 2012; Solver and Benchmark Descriptions*. Ed. by Adrian Balint et al. Helsinki, Finland: University of Helsinki, 2012. ISBN: 978-952-10-8106-4 (cit. on pp. 33, 126).

[Gas+13] Serge Gaspers et al. Backdoors to q-Horn. In: *Proceedings of the 30th International Symposium on Theoretical Aspects of Computer Science (STACS'13)*. Ed. by Natacha Portier and Thomas Wilke. Vol. 20. Leibniz International Proceedings in Informatics (LIPIcs). Kiel, Germany: Dagstuhl Publishing, Feb. 2013, pp. 67–79. ISBN: 978-3-939897-50-7. DOI: 10.4230/LIPIcs.STACS.2013.67 (cit. on p. 33).

[Gas+14] Serge Gaspers et al. Backdoors into Heterogeneous Classes of SAT and CSP. In: *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI'14)*. Ed. by Carla E. Brodley and Peter Stone. Québec City, QC, Canada: The AAAI Press, July 2014, pp. 2652–2658. ISBN: 978-1-57735-661-5 (cit. on p. 135).

[Geb+07] Martin Gebser et al. The First Answer Set Programming System Competition. In: *Proceedings of the 9th Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*. Ed. by Chitta Baral, Gerhard Brewka, and John Schlipf. Vol. 4483. Lecture Notes in Computer Science. Tempe, AZ, USA: Springer Verlag, May 2007, pp. 3–17. ISBN: 978-3-540-72199-4. DOI: 10.1007/978-3-540-72200-7_3 (cit. on pp. 126, 127).

[Geb+08] Martin Gebser et al. Advanced Preprocessing for Answer Set Solving. In: *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI'08)*. Ed. by Malik Ghallab et al. Vol. 178. Frontiers in Artificial Intelligence and Applications. Patras, Greece: IOS Press, July 2008, pp. 15–19. ISBN: 978-1-58603-891-5. DOI: 10.3233/978-1-58603-891-5-15 (cit. on pp. 59, 64).

[Geb+10] Martin Gebser et al. *A User's Guide to gringo, clasp, clingo, and iclingo*. Tech. rep. University Potsdam, 2010 (cit. on p. 126).

[Geb+11a] Martin Gebser et al. Challenges in Answer Set Solving. In: *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning – Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*. Ed. by Marcello Balduccini and TranCao Son. Vol. 6565. Lecture Notes in Artificial Intelligence. Springer Verlag, 2011, pp. 74–90. ISBN: 978-3-642-20831-7. DOI: 10.1007/978-3-642-20832-4_6 (cit. on p. 59).

[Geb+11b] Martin Gebser et al. Multi-Criteria Optimization in Answer Set Programming. In: *Technical Communications of the 27th International Conference on Logic Programming (ICLP'11)*. Ed. by John Gallagher and Michael Gelfond. Vol. 11. Leibniz International Proceedings in Informatics (LIPIcs). Lexington, KY, USA: Dagstuhl Publishing, July 2011, pp. 1–10. ISBN: 978-3-939897-31-6. DOI: 10.4230/LIPIcs.ICLP.2011.1 (cit. on pp. 1, 12).

[Geb+11c] Martin Gebser et al. Potassco: The Potsdam Answer Set Solving Collection. In: *AI Communications* 24.2 (2011), pp. 107–124. ISSN: 1875-8452. DOI: 10.3233/AIC-2011-0491 (cit. on pp. 1, 11, 126).

[Geb+11d] M. Gebser et al. Detecting Inconsistencies in Large Biological Networks with Answer Set Programming. In: *Theory Pract. Log. Program.* 11.2-3 (2011), pp. 323–360 (cit. on p. 127).

[Geb+12] Martin Gebser et al. *Answer Set Solving in Practice*. Morgan & Claypool, 2012. DOI: 10.2200/S00457ED1V01Y201211AIM019 (cit. on pp. 1, 10, 126).

[Geb+13a] Martin Gebser et al. Domain-specific Heuristics in Answer Set Programming. In: *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI'13)*. Ed. by Marie des Jardins and Michael Littman. Bellevue, WA, USA: The AAAI Press, July 2013, pp. 350–356. ISBN: 978-1-57735-615-8 (cit. on pp. 130, 131).

[Geb+13b] Martin Gebser et al. Matchmaking with Answer Set Programming. In: *Proceedings of 12th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'13)*. Ed. by Pedro Cabalar and Tran Cao Son. Vol. 8148. Lecture Notes in Computer Science. Corunna, Spain: Springer Verlag, Sept. 2013, pp. 342–347. ISBN: 978-3-642-40564-8. DOI: 10.1007/978-3-642-40564-8_34 (cit. on pp. 1, 12).

[Gel89] Allen Van Gelder. Negation as failure using tight derivations for general logic programs. In: *J. Logic Programming* 6.1–2 (1989), pp. 109–133. ISSN: 0743-1066. DOI: 10.1016/0743-1066(89)90032-0 (cit. on pp. 10, 43).

[GJR14] Martin Gebser, Tomi Janhunen, and Jussi Rintanen. Answer Set Programming as SAT modulo Acyclicity. In: *Proceedings of the 21st Eureopean Conference on Artificial Intelligence (ECAI'14)*. Ed. by Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan. Vol. 263. Frontiers in Artificial Intelligence and Applications. Prague, Czech Republic: IOS Press, Aug. 2014, pp. 351–356. ISBN: 978-1-61499-418-3. DOI: 10.3233/978-1-61499-419-0-351 (cit. on pp. 11, 14, 118, 124).

[GK12] Martin Gebser and Roland Kaminski. *Personal communication*. 2012 (cit. on pp. 126, 127).

[GKS11] Martin Gebser, Roland Kaminski, and Torsten Schaub. Complex optimization in answer set programming. In: *Theory Pract. Log. Program.* 11.4-5 (2011), pp. 821–839. DOI: 10.1017/S1471068411000329 (cit. on p. 126).

[GKS12a] Martin Gebser, Roland Kaufmann, and Torsten Schaub. Gearing up for effective ASP planning. In: *Correct Reasoning – Essays on Logic-Based AI in Honour of Vladimir Lifschitz*. Ed. by Esra Erdem et al. Vol. 7265. Lecture Notes in Computer Science. Springer Verlag, 2012, pp. 296–310. ISBN: 978-3-642-30742-3. DOI: 10.1007/978-3-642-30743-0_20 (cit. on p. 12).

[GKS12b] M. Gebser, B. Kaufmann, and T. Schaub. Conflict-Driven Answer Set Solving: From Theory to Practice. In: *Artificial Intelligence* 187-188 (2012), pp. 52–89. DOI: 10.1016/j.artint.2012.04.001 (cit. on pp. 11, 122, 126).

[GKS13] Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. Advanced conflict-driven disjunctive answer set solving. In: *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI'13)*. Ed. by Francesca Rossi and Sebastian Thrun. Beijing, China: The AAAI Press, Aug. 2013, pp. 912–918. ISBN: 978-1-57735-633-2 (cit. on pp. 11, 122).

[GL88] Michael Gelfond and Vladimir Lifschitz. The Stable Model Semantics For Logic Programming. In: *Proceedings of the 5th International Conference and Symposium on Logic Programming (ICLP/SLP'88)*. Ed. by Robert A. Kowalski and Kenneth A. Bowen. Vol. 2. Seattle, WA, USA: MIT Press, Aug. 1988, pp. 1070–1080. ISBN: 0-262-61056-6 (cit. on pp. 1, 9, 10, 17, 57).

[GL91] Michael Gelfond and Vladimir Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. In: *New Generation Comput.* 9.3/4 (1991), pp. 365–386. DOI: 10.1007/BF03037169 (cit. on pp. 8, 10, 57).

[GLL11] Martin Gebser, Joohyung Lee, and Yuliya Lierler. On elementary loops of logic programs. In: *Theory Pract. Log. Program.* 11 (06 Nov. 2011), pp. 953–988. ISSN: 1475-3081. DOI: 10.1017/S1471068411000019 (cit. on p. 14).

[GLM06] Enrico Giunchiglia, Yuliya Lierler, and Marco Maratea. Answer set programming based on propositional satisfiability. In: *J. Automated Reasoning* 36.4 (2006), pp. 345–377. DOI: 10.1007/s10817-006-9033-2 (cit. on pp. 11, 122).

[GMS98] E. Giunchiglia, A. Massarotto, and R. Sebastiani. Act, and the rest will follow: Exploiting determinism in planning as satisfiability. In: *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI'98)*. Ed. by Jack Mostow and Charles Rich. Madison, WI, USA: The AAAI Press, 1998, pp. 948–953. ISBN: 0-262-51098-7 (cit. on p. 129).

[GMT02] E. Giunchiglia, M. Maratea, and A. Tacchella. Dependent and independent variables in propositional satisfiability. In: *Proceedings of the 8th European Conference on Logics in Artificial Intelligence (JELIA'02)*. Ed. by Sergio Flesca et al. Vol. 2424. Lecture Notes in Computer Science. Springer Verlag, 2002, pp. 296–307. ISBN: 3-540-44190-5. DOI: 10.1007/3-540-45757-7_25 (cit. on p. 129).

[Gom+08] Carla P. Gomes et al. Chapter 2: Satisfiability Solvers. In: *Handbook of Knowledge Representation*. Ed. by Vladimir Lifschitz Frank van Harmelen and Bruce Porter. Vol. 3. Foundations of Artificial Intelligence. Amsterdam, Netherlands: Elsevier Science Publishers, North-Holland, 2008. Chap. 2, pp. 89–134. ISBN: 978-0-444-52211-5. DOI: 10.1016/S1574-6526(07)03002-7 (cit. on pp. 1, 13).

[GP04] Georg Gottlob and Reinhard Pichler. Hypergraphs in model checking: acyclicity and hypertree-width versus clique-width. In: *SIAM J. Comput.* 33.2 (2004), pp. 351–378. ISSN: 0097-5397. DOI: 10.1137/S0097539701396807 (cit. on p. 100).

[GPW10] Georg Gottlob, Reinhard Pichler, and Fang Wei. Bounded treewidth as a key to tractability of knowledge representation and reasoning. In: *Artificial Intelligence* 174.1 (2010), pp. 105–132. ISSN: 0004-3702. DOI: 10.1016/j.artint.2009.10.003 (cit. on pp. 2, 96).

[GS05] M. Gebser and T. Schaub. Loops: Relevant or redundant? In: *Proceedings of the 8th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05)*. Ed. by Chitta Baral et al. Vol. 3662. Lecture Notes in Computer Science. Diamante, Italy: Springer Verlag, Sept. 2005, pp. 53–65. ISBN: 3-540-28538-5. DOI: 10.1007/11546207_5 (cit. on p. 13).

[GS08] Georg Gottlob and Stefan Szeider. Fixed-parameter Algorithms For Artificial Intelligence, Constraint Satisfaction and Database Problems. In: *The Computer Journal* 51.3 (2008), pp. 303–325. ISSN: 0010-4620. DOI: 10.1093/comjnl/bxm056. eprint: http://comjnl.oxfordjournals.org/content/51/3/303.full.pdf+html (cit. on pp. 2, 18, 21).

[GS09] Martin Gebser and Torsten Schaub. *Asparagus*. http://asparagus.cs.uni-potsdam.de. 2009 (cit. on pp. 126, 127).

[GS12a] Serge Gaspers and Stefan Szeider. Backdoors to Acyclic SAT. In: *Proceedings of the 39th International Colloquium on Automata, Languages and Programming (ICALP'12)*. Ed. by Artur Czumaj et al. Vol. 7391. Lecture Notes in Computer Science. Warwick, UK: Springer Verlag, July 2012, pp. 363–374. ISBN: 978-3-642-31593-0. DOI: 10.1007/978-3-642-31594-7_31 (cit. on pp. 33, 50).

[GS12b] Serge Gaspers and Stefan Szeider. Backdoors to Satisfaction. In: *The Multivariate Algorithmic Revolution and Beyond*. Ed. by Hans Bodlaender et al. Vol. 7370. Lecture Notes in Computer Science. Heidelberg, Germany: Springer Verlag, 2012, pp. 287–317. ISBN: 978-3-642-30890-1. DOI: 10.1007/978-3-642-30891-8_15 (cit. on pp. 2, 13, 23, 24, 33, 49, 135).

[GS12c] Serge Gaspers and Stefan Szeider. Strong Backdoors to Nested Satisfiability. In: *Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing (SAT'12)*. Ed. by Alessandro Cimatti and Roberto Sebastiani. Vol. 7317. Lecture Notes in Computer Science. Trento, Italy: Springer Verlag, June 2012, pp. 72–85. ISBN: 978-3-642-31611-1. DOI: 10.1007/978-3-642-31612-8_7 (cit. on p. 33).

[GS13] Serge Gaspers and Stefan Szeider. Strong Backdoors to Bounded Treewidth SAT. In: *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS'13)*. Ed. by Omer Reingold. IEEE Computer Soc. Berkeley, CA, USA, Oct. 2013, pp. 489–498. ISBN: 978-0-7695-5135-7. DOI: 10.1109/FOCS.2013.59 (cit. on pp. 33, 50).

[GS13] Martin Gebser and Torsten Schaub. Tableau Calculi for Logic Programs under Answer Set Semantics. In: *ACM Trans. Comput. Log.* 14.2 (2013), p. 15. DOI: 10.1145/2480759.2480767 (cit. on p. 11).

[GS14] Serge Gaspers and Stefan Szeider. Guarantees and limits of preprocessing in constraint satisfaction and reasoning. In: *Artificial Intelligence* 216 (2014), pp. 1–19. ISSN: 0004-3702. DOI: 10.1016/j.artint.2014.06.006 (cit. on pp. 21, 62, 64).

[GS88] Giorgio Gallo and Maria Grazia Scutellà. Polynomially solvable satisfiability problems. In: *Information Processing Letters* 29.5 (1988), pp. 221–227. DOI: 10.1016/0020-0190(88)90113-5 (cit. on pp. 71, 81).

[GSS02] Georg Gottlob, Francesco Scarcello, and Martha Sideri. Fixed-parameter complexity in AI and nonmonotonic reasoning. In: *Artificial Intelligence* 138.1-2 (2002), pp. 55–86. ISSN: 0004-3702. DOI: 10.1016/S0004-3702(02)00182-0 (cit. on pp. 21, 25, 34, 40, 50, 51, 90, 93, 95).

[GST07] Martin Gebser, Torsten Schaub, and Sven Thiele. GrinGo: A New Grounder for Answer Set Programming. English. In: *Proceedings of the 9th International Conference Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*. Ed. by Chitta Baral, Gerhard Brewka, and John Schlipf. Vol. 4483. Lecture Notes in Computer Science. Tempe, AZ, USA: Springer Verlag, May 2007, pp. 266–271. ISBN: 978-3-540-72199-4. DOI: 10.1007/978-3-540-72200-7_24 (cit. on p. 11).

[Gur14] Gurobi Optimization, Inc. *Gurobi Optimizer Reference Manual*. http://www.gurobi.com. Version 5.0.2. 2014 (cit. on p. 126).

[Hoo12] Holger H. Hoos. Automated Algorithm Configuration and Parameter Tuning. English. In: *Autonomous Search*. Ed. by Youssef Hamadi, Eric Monfroy, and Frédéric Saubion. Springer Verlag, 2012, pp. 37–71. ISBN: 978-3-642-21433-2. DOI: 10.1007/978-3-642-21434-9_3 (cit. on p. 130).

[HS15] Ronald de Haan and Stefan Szeider. Machine Characterizations for Parameterized Complexity Classes Beyond Para-NP. English. In: *Proceedings of the 41st International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'15)*. Ed. by Giuseppe F. Italiano et al. Vol. 8939. Lecture Notes in Computer Science. Pec pod Sněžkou, Czech Republic: Springer Verlag, Jan. 2015, pp. 217–229. ISBN: 978-3-662-46077-1. DOI: 10.1007/978-3-662-46078-8_18 (cit. on p. 134).

[Hut+14] Frank Hutter et al. Algorithm runtime prediction: Methods & evaluation. In: *Artificial Intelligence* 206 (2014), pp. 79–111. ISSN: 0004-3702. DOI: 10.1016/j.artint.2013.10.003 (cit. on p. 13).

[ILO11] IBM ILOG. *CPLEX Optimization Studio CPLEX User's Manual*. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer. IBM, 2011 (cit. on p. 126).

[Jan+06] Tomi Janhunen et al. Unfolding Partiality and Disjunctions in Stable Model Semantics. In: *ACM Trans. Comput. Log.* 7.1 (2006), pp. 1–37. DOI: 10.1145/1119439.1119440 (cit. on pp. 11, 14, 122).

[Jan06] Tomi Janhunen. Some (in)translatability results for normal logic programs and propositional theories. In: *J. Applied Non-Classical Logics* 16.1-2 (2006), pp. 35–86. DOI: 10.3166/jancl.16.35-86 (cit. on pp. 2, 14, 107, 113, 118, 122).

[Jan+09] Tomi Janhunen et al. Modularity aspects of disjunctive stable models. In: *J. Artif. Intell. Res.* 35.2 (2009), pp. 813–857. DOI: 10.1613/jair.2810 (cit. on pp. 118, 124).

[Jan13a] Tomi Janhunen. *ASP(LC)*. http://research.ics.aalto.fi/software/asp/mingo. Mar. 2013 (cit. on p. 11).

[Jan13b] Tomi Janhunen. *GnT (Generate'n'Test): A Solver for Disjunctive Logic Programs*. http://www.tcs.hut.fi/Software/gnt. Feb. 2013 (cit. on pp. 11, 122).

[Jan13c] Tomi Janhunen. *LP2DIFF: Translating Normal/Smodels Programs for SMT (difference logic) solvers*. http://www.tcs.hut.fi/Software/lp2diff. May 2013 (cit. on p. 11).

[Jan13d] Tomi Janhunen. *LP2SAT: Translating Normal Programs for SAT solvers*. http://www.tcs.hut.fi/Software/lp2sat. Feb. 2013 (cit. on p. 11).

[Jan14] Tomi Janhunen. *LP2ACYC*. http://research.ics.aalto.fi/software/asp/lp2acyc. Oct. 2014 (cit. on p. 11).

[Jär+12] Matti Järvisalo et al. Relating Proof Complexity Measures and Practical Hardness of SAT. In: *Proceedings of the 18th International Conference Principles and Practice of Constraint Programming (CP'12)*. Ed. by Michela Milano. Lecture Notes in Computer Science. Québec City, QC, Canada: Springer Verlag, Oct. 2012, pp. 316–331. ISBN: 978-3-642-33557-0. DOI: 10.1007/978-3-642-33558-7_25 (cit. on p. 13).

[Ji+14] Jianmin Ji et al. Elementary Loops Revisited. In: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI'14)*. Ed. by Carla E. Brodley and Peter Stone. Québec City, QC, Canada.: The AAAI Press, July 2014, pp. 1063–1069. ISBN: 978-1-57735-661-5 (cit. on p. 14).

[JJ09] Matti Järvisalo and Tommi Junttila. Limitations of Restricted Branching in Clause Learning. In: *Constraints* 14.3 (2009), pp. 325–356. ISSN: 1383-7133. DOI: 10.1007/s10601-008-9062-z (cit. on p. 129).

[JLN11] Tomi Janhunen, Guohua Liu, and Ilkka Niemelä. Tight integration of non-ground answer set programming and satisfiability modulo theories. In: *Proceedings of the 1st Workshop on Grounding and Transformation for Theories with Variables (GTTV'11)*. Ed. by Pedro Cabalar et al. Vancouver, BC, Canada, May 2011, pp. 1–13 (cit. on p. 14).

[JM11] Mikoláš Janota and Joao Marques-Silva. A Tool for Circumscription-Based MUS Membership Testing. In: *Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11)*. Ed. by James P. Delgrande and Wolfgang Faber. Vol. 6645. Lecture Notes in Computer Science. Vancouver, Canada: Springer Verlag, May 2011, pp. 266–271. ISBN: 978-3-642-20894-2. DOI: 10.1007/978-3-642-20895-9_30 (cit. on p. 127).

[JN08] M. Järvisalo and Ilkka Niemelä. The Effect of Structural Branching on the Efficiency of Clause Learning SAT solving: An Experimental Study. In: *J. Algorithms* 63.1-3 (2008), pp. 90–113. ISSN: 0196-6774. DOI: 10.1016/j.jalgor.2008.02.005 (cit. on p. 129).

[JN11] Tomi Janhunen and Ilkka Niemelä. Compact Translations of Non-disjunctive Answer Set Programs to Propositional Clauses. In: *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning – Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*. Ed. by Marcello Balduccini and Tran Son. Vol. 6565. Lecture Notes in Artificial Intelligence. Springer Verlag, 2011, pp. 111–130. ISBN: 978-3-642-20831-7. DOI: 10.1007/978-3-642-20832-4_8 (cit. on p. 11).

[JNS09] Tomi Janhunen, Ilkka Niemelä, and Mark Sevalnev. Computing Stable Models via Reductions to Difference Logic. In: *Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*. Ed. by Esra Erdem, Fangzhen Lin, and Torsten Schaub. Vol. 5753. Lecture Notes in Computer Science. Potsdam, Germany: Springer Verlag, Sept. 2009, pp. 142–154. ISBN: 978-3-642-04237-9. DOI: 10.1007/978-3-642-04238-6_14 (cit. on pp. 11, 14, 118, 124).

[JPW09] Michael Jakl, Reinhard Pichler, and Stefan Woltran. Answer-Set Programming with Bounded Treewidth. In: *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*. Ed. by Craig Boutilier. Vol. 2. Pasadena, CA, USA: The AAAI Press, July 2009, pp. 816–822 (cit. on pp. 2, 21, 96, 97).

[JSS12] Holger Jost, Orkunt Sabuncu, and Torsten Schaub. Suggesting New Interactions Related to Events in a Social Network for Elderly. In: *Proceedings of the 2nd International Workshop on Design and Implementation of Independent and Assisted Living Technology (DIILT'12)*. Ed. by Matt-Mouley Bouamrane, Marilyn McGee-Lennon, and Frances S. Mair. Birmingham, UK, Sept. 2012 (cit. on pp. 1, 12).

[JWX15] Jianmin Ji, Hai Wan, and Peng Xiao. On Elementary Loops and Proper Loops for Disjunctive Logic Programs. In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI'15)*. Ed. by Blai Bonet and Sven Koenig. Austin, TX, USA: The AAAI Press, Jan. 2015, pp. 1518–1524. ISBN: 978-1-57735-698-1 (cit. on p. 14).

[Kau+15] Benjamin Kaufmann et al. *clasp – A conflict-driven nogood learning answer set solver*. http://www.cs.uni-potsdam.de/clasp/. 2015 (cit. on p. 11).

[KK12] Ken-ichi Kawarabayashi and Yusuke Kobayashi. Fixed-parameter tractability for the subset feedback set problem and the S-cycle packing problem. In: *J. Combin. Theory* 102.4 (2012), pp. 1020–1034. DOI: 10.1016/j.jctb.2011.12.001 (cit. on p. 47).

[KKK12] Naonori Kakimura, Ken-ichi Kawarabayashi, and Yusuke Kobayashi. Erdös-Pósa property and its algorithmic applications: parity constraints, subset feedback set, and subset packing. In: *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'12)*. Ed. by Dana Randall. Kyoto, Japan: Society for Industrial and Applied Mathematics (SIAM), Jan. 2012, pp. 1726–1736. ISBN: 978-1-61197-210-8. DOI: 10.1137/1.9781611973099.137 (cit. on p. 47).

[KKS08] Stephan Kottler, Michael Kaufmann, and Carsten Sinz. A New Bound for an NP-Hard Subclass of 3-SAT Using Backdoors. In: *Proceedings of the 11th International Conference on Theory and Applications of Satisfiability Testing (SAT'08)*. Ed. by Hans Kleine Büning and Xishun Zhao. Vol. 4996. Lecture Notes in Computer Science. Guangzhou, China: Springer Verlag, May 2008, pp. 161–167. ISBN: 978-3-540-79718-0. DOI: 10.1007/978-3-540-79719-7_16 (cit. on p. 33).

[KL80] Richard M. Karp and Richard J. Lipton. Some Connections between Nonuniform and Uniform Complexity Classes. In: *Proceedings of the 12th Annual ACM Symposium on Theory of Computing (STOC'80)*. Ed. by Raymond E. Miller et al. Los Angeles, CA, USA: Assoc. Comput. Mach., New York, Apr. 1980, pp. 302–309. ISBN: 0-89791-017-6. DOI: 10.1145/800141.804678 (cit. on p. 16).

[KL99] Hans Kleine Büning and Theodor Lettman. *Propositional logic: deduction and algorithms*. New York, NY, USA: Cambridge University Press, Cambridge, 1999, p. 420. ISBN: 978-0521630177 (cit. on pp. 13, 17, 118).

[KS92] Kanchana Kanchanasut and Peter J. Stuckey. Transforming normal logic programs to constraint logic programs. In: *Theoretical Computer Science* 105.1 (1992), pp. 27–56. ISSN: 0304-3975. DOI: 10.1016/0304-3975(92)90286-O (cit. on p. 100).

[KSM11] Hadi Katebi, Karem A. Sakallah, and João P. Marques-Silva. Empirical Study of the Anatomy of Modern SAT Solvers. English. In: *Proceedings of the 14th International Conference on Theory and Applications of Satisfiability Testing (SAT'11)*. Ed. by Karem A. Sakallah and Laurent Simon. Vol. 6695. Lecture Notes in Computer Science. Ann Arbor, MI, USA: Springer Verlag, June 2011, pp. 343–356. ISBN: 978-3-642-21580-3. DOI: 10.1007/978-3-642-21581-0_27 (cit. on pp. 13, 130).

[Lam11] Michael Lampis. A kernel of order for vertex cover. In: *Information Processing Letters* 111.23–24 (2011), pp. 1089–1091. ISSN: 0020-0190. DOI: 10.1016/j.ipl.2011.09.003 (cit. on p. 63).

[LB10] Florian Lonsing and Armin Biere. DepQBF: A Dependency-Aware QBF Solver system description. In: *J. on Satisfiability, Boolean Modeling and Computation* 7 (2010), pp. 71–76 (cit. on p. 122).

[LB11] Zijie Li and Peter van Beek. Finding Small Backdoors in SAT Instances. English. In: *Proceedings of the 24th Canadian Conference on Artificial Intelligence (Canadian AI'11)*. Ed. by Cory Butz and Pawan Lingras. Vol. 6657. Lecture Notes in Computer Science. St. John's, NL, Canada: Springer Verlag, May 2011, pp. 269–280. ISBN: 978-3-642-21042-6. DOI: 10.1007/978-3-642-21043-3_33 (cit. on p. 33).

[Lee05] Joohyung Lee. A model-theoretic counterpart of loop formulas. In: *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*. Ed. by Leslie Pack Kaelbling and Alessandro Saffiotti. Vol. 19. Edinburgh, Scotland, UK: Professional Book Center, 2005, pp. 503–508. ISBN: 0938075934 (cit. on pp. 122, 123).

[Leo+06] Nicola Leone et al. The DLV System for Knowledge Representation and Reasoning. In: *ACM Trans. Comput. Log.* 7.3 (2006), pp. 499–562. ISSN: 1529-3785. DOI: 10.1145/1149114.1149117 (cit. on pp. 1, 11, 122).

[LEV13] Florian Lonsing, Uwe Egly, and Allen Van Gelder. Efficient Clause Learning for Quantified Boolean Formulas via QBF Pseudo Unit Propagation. English. In: *Proceedings of the 16th Conference on Theory and Applications of Satisfiability Testing (SAT'13)*. Ed. by Matti Järvisalo and Allen Van Gelder. Vol. 7962. Lecture Notes in Computer Science. Helsinki, Finland: Springer Verlag, July 2013, pp. 100–115. ISBN: 978-3-642-39070-8. DOI: 10.1007/978-3-642-39071-5_9 (cit. on pp. 13, 122).

[Lev73] Leonid Levin. Universal sequential search problems. In: *Problems of Information Transmission* 9.3 (1973), pp. 265–266 (cit. on p. 17).

[Lie05] Yuliya Lierler. CMODELS – SAT-Based Disjunctive Answer Set Solver. In: *Proceedings of the 8th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05)*. Ed. by Chitta Baral et al. Vol. 3662. Lecture Notes in Computer Science. Diamante, Italy: Springer Verlag, Sept. 2005, pp. 447–451. ISBN: 978-3-540-28538-0. DOI: 10.1007/11546207_44 (cit. on pp. 11, 104, 122).

[Lie11] Yulia Lierler. *Cmodels*. http://www.cs.utexas.edu/users/tag/cmodels. June 2011 (cit. on pp. 11, 122).

[Lie14] Yuliya Lierler. Relating constraint answer set programming languages and algorithms. In: *Artificial Intelligence* 207 (2014), pp. 1–22. ISSN: 0004-3702. DOI: 10.1016/j.artint.2013.10.004 (cit. on p. 11).

[Lif10] Vladimir Lifschitz. Thirteen Definitions of a Stable Model. In: *Fields of Logic and Computation – Essays Dedicated to Yuri Gurevich on the Occasion of His 70th Birthday*. Ed. by Andreas Blass, Nachum Dershowitz, and Wolfgang Reisig. Vol. 6300. Lecture Notes in Computer Science. Springer Verlag, 2010, pp. 488–503. ISBN: 978-3-642-15024-1. DOI: 10.1007/978-3-642-15025-8_24 (cit. on p. 10).

[Lif99] Vladimir Lifschitz. Answer Set Planning. English. In: *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'99)*. Ed. by Michael Gelfond, Nicola Leone, and Gerald Pfeifer. Vol. 1730. Lecture Notes in Computer Science. Paso, TX, USA: Springer Verlag, Dec. 1999, pp. 373–374. ISBN: 978-3-540-66749-0. DOI: 10.1007/3-540-46767-X_28 (cit. on p. 10).

[LJN12] Guohua Liu, Tomi Janhunen, and Ilkka Niemelä. Answer Set Programming via Mixed Integer Programming. In: *Proceedings of the 13th International Conference on the Principles of Knowledge Representation and Reasoning (KR'12)*. Ed. by Sheila McIlraith and Thomas Eiter. Rome, Italy: The AAAI Press, 2012, pp. 32–42. ISBN: 978-1-57735-560-1 (cit. on pp. 11, 14).

[LL03] Joohyung Lee and Vladimir Lifschitz. Loop Formulas for Disjunctive Logic Programs. In: *Proceedings of the 19th International Conference on Logic Programming (LP'03)*. Ed. by Catuscia Palamidessi. Vol. 2916. Lecture Notes in Computer Science. Mumbai, India: Springer Verlag, Dec. 2003, pp. 451–465. ISBN: 978-3-540-20642-2. DOI: 10.1007/978-3-540-24599-5_31 (cit. on pp. 13, 57, 104, 120, 122, 123).

[Lon12] Florian Lonsing. Dependency Schemes and Search-Based QBF Solving: Theory and Practice. PhD thesis. Johannes Kepler Universität, Linz, Austria, 2012 (cit. on p. 122).

[LP84] Andrea S. Lapaugh and Christos H. Papadimitriou. The even-path problem for graphs and digraphs. In: *Networks* 14.4 (1984), pp. 507–513. ISSN: 1097-0037. DOI: 10.1002/net.3230140403 (cit. on p. 45).

[LR06] Vladimir Lifschitz and Alexander Razborov. Why are there so many loop formulas? In: *ACM Trans. Comput. Log.* 7.2 (2006), pp. 261–268. DOI: 10.1145/1131313.1131316 (cit. on pp. 14, 120, 122).

[LRS97] Nicola Leone, Pasquale Rullo, and Francesco Scarcello. Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics, and Computation. In: *Information and Computation* 135.2 (1997), pp. 69–112. DOI: 10.1006/inco.1997.2630 (cit. on p. 105).

[LT06] Zbigniew Lonc and Mirosław Truszczyński. Computing minimal models, stable models and answer sets. In: *Theory Pract. Log. Program.* 6 (04 July 2006), pp. 395–449. ISSN: 1475-3081. DOI: 10.1017/S1471068405002607 (cit. on p. 17).

[LT11] Yuliya Lierler and Mirosław Truszczyński. Transition systems for model generators – A unifying approach. In: *Theory Pract. Log. Program.* 11 (Special Issue 4-5 July 2011), pp. 629–646. ISSN: 1475-3081. DOI: 10.1017/S1471068411000214 (cit. on p. 11).

[LZ03] Fangzhen Lin and Jicheng Zhao. On tight logic programs and yet another translation from normal logic programs to propositional logic. In: *Proceedings of the 18th International Joint Conference on Artificial intelligence (IJCAI'03)*. Ed. by Georg Gottlob and Toby Walsh. Acapulco, Mexico: Morgan Kaufmann, Aug. 2003, pp. 853–858 (cit. on pp. 2, 14).

[LZ04a] Fangzhen Lin and Xishun Zhao. On Odd and Even Cycles in Normal Logic Programs. In: *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI'04)*. Ed. by Deborah L. McGuinness and George Ferguson. San Jose, CA, USA: The AAAI Press, July 2004, pp. 80–85. ISBN: 978-0-262-51183-4 (cit. on pp. 2, 14, 21, 43, 50, 51, 57, 102, 104).

[LZ04b] Fangzhen Lin and Yuting Zhao. ASSAT: Computing Answer Sets of a Logic Program by SAT Solvers. In: *Artificial Intelligence* 157.1-2 (2004), pp. 115–137. DOI: 10.1016/j.artint.2004.04.004 (cit. on pp. 11, 120, 122).

[MAG08] Marco Montalva, Julio Aracena, and Anahí Gajardo. On the complexity of feedback set problems in signed digraphs. In: *Electronic Notes in Discrete Mathematics* 30 (2008), pp. 249–254. ISSN: 1571-0653. DOI: 10.1016/j.endm.2008.01.043 (cit. on p. 47).

[Mal07] Sharad Malik. *zChaff*. http://www.princeton.edu/~chaff/zchaff.html. 2007 (cit. on p. 122).

[Mar+08] Maarten Mariën et al. SAT(ID): Satisfiability of Propositional Logic Extended with Inductive Definitions. In: *Proceedings of the 11th International Conference on Theory and Applications of Satisfiability Testing (SAT'08)*. Ed. by Hans Kleine Büning and Xishun Zhao. Vol. 4996. Lecture Notes in Computer Science. Guangzhou, China: Springer Verlag, May 2008, pp. 211–224. ISBN: 978-3-540-79718-0. DOI: 10.1007/978-3-540-79719-7_20 (cit. on p. 11).

[Mar99] João Marques-Silva. The Impact of Branching Heuristics in Propositional Satisfiability Algorithms. English. In: *Proceedings of the 9th Portuguese Conference on Artificial Intelligence (EPIA'99)*. Ed. by Pedro Barahona and José J. Alferes. Vol. 1695. Lecture Notes in Computer Science. Évora, Portugal: Springer Verlag, Sept. 1999, pp. 62–74. ISBN: 978-3-540-66548-9. DOI: 10.1007/3-540-48159-1_5 (cit. on p. 13).

[MFM05] Yogesh S. Mahajan, Zhaohui Fu, and Sharad Malik. Zchaff2004: An Efficient SAT Solver. In: *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*. Ed. by Holger H. Hoos and David G. Mitchell. Vol. 3542. Lecture Notes in Computer Science. Vancouver, BC, Canada: Springer Verlag, May 2005, pp. 360–375. ISBN:

3-540-27829-X, 978-3-540-27829-0. DOI: 10.1007/11527695_27 (cit. on p. 122).

[Min93] Jack Minker. An overview of nonmonotonic reasoning and logic programming. In: *J. Logic Programming* 17.2–4 (1993). Special Issue: Non-Monotonic Reasoning and Logic Programming, pp. 95–126. ISSN: 0743-1066. DOI: 10.1016/0743-1066(93)90028-F (cit. on p. 17).

[Mis+12] Pranabendu Misra et al. Parameterized Algorithms for Even Cycle Transversal. In: *Proceedings of the 38th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'12)*. Ed. by Martin Charles Golumbic et al. Vol. 7551. Lecture Notes in Computer Science. Jerusalem, Israel: Springer Verlag, June 2012, pp. 172–183. ISBN: 978-3-642-34610-1. DOI: 10.1007/978-3-642-34611-8_19 (cit. on p. 47).

[Mor+10] Michael Morak et al. A Dynamic-Programming Based ASP-Solver. In: *Proceedings of 12th European Conference on Logics in Artificial Intelligence (JELIA'10)*. Ed. by Tomi Janhunen and Ilkka Niemelä. Vol. 6341. Lecture Notes in Computer Science. Helsinki, Finland: Springer Verlag, Sept. 2010, pp. 369–372. ISBN: 978-3-642-15674-8. DOI: 10.1007/978-3-642-15675-5_34 (cit. on p. 21).

[MS99] J.P. Marques-Silva and K.A. Sakallah. GRASP: a search algorithm for propositional satisfiability. In: *IEEE Transactions on Computers* 48.5 (May 1999), pp. 506–521. ISSN: 0018-9340. DOI: 10.1109/12.769433 (cit. on p. 13).

[MT91a] Wiktor Marek and Mirosław Truszczyński. Autoepistemic logic. In: *J. of the ACM* 38.3 (1991), pp. 588–619. ISSN: 0004-5411. DOI: 10.1145/116825.116836 (cit. on pp. 17, 28, 123).

[MT91b] Wiktor Marek and Mirosław Truszczyński. Computing intersection of autoepistemic expansions. In: *Proceedings of the 1st International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'91)*. Ed. by Anil Nerode, V. Wiktor Marek, and V. S. Subrahmanian. Washington, D.C., USA: MIT Press, July 1991, pp. 37–50 (cit. on pp. 109, 123).

[MT99] Victor W. Marek and Mirosław Truszczyński. Stable models and an alternative logic programming paradigm. In: *The Logic Programming Paradigm: a 25-Year Perspective*. Ed. by Krzysztof R. Apt et al. Artificial Intelligence. Springer Verlag, 1999, pp. 375–398. ISBN: 978-3642642494. DOI: 10.1007/978-3-642-60085-2 (cit. on pp. 1, 10).

[MW12] Michael Morak and Stefan Woltran. Preprocessing of Complex Non-Ground Rules in Answer Set Programming. In: *Technical Communications of the 28th International Conference on Logic Programming (ICLP'12)*. Ed. by Agostino Dovier and Vítor Santos Costa. Vol. 17. Leibniz International Proceedings in Informatics (LIPIcs). Budapest, Hungary: Dagstuhl Publishing, Sept. 2012, pp. 247–258. ISBN: 978-3-939897-43-9. DOI: 10.4230/LIPIcs.ICLP.2012.247 (cit. on pp. 64, 96).

[New+14] Zack Newsham et al. Impact of Community Structure on SAT Solver Performance. English. In: *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT'14)*. Ed. by Carsten Sinz and Uwe Egly. Vol. 8561. Lecture Notes in Computer Science. Held as Part of the Vienna Summer of Logic, VSL 2014. Vienna, Austria: Springer Verlag, July 2014, pp. 252–268. ISBN: 978-3-319-09283-6. DOI: 10.1007/978-3-319-09284-3_20 (cit. on p. 13).

[Nie06] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Vol. 31. Oxford Lecture Series in Mathematics and its Applications. New York, NY, USA: Oxford University Press, 2006. ISBN: 978-0-19-856607-6 (cit. on pp. 18, 20).

[Nie99] Ilkka Niemelä. Logic Programs With Stable Model Semantics as a Constraint Programming Paradigm. In: *Ann. Math. Artif. Intell.* 25.3 (1999), pp. 241–273. ISSN: 1012-2443. DOI: 10.1023/A:1018930122475 (cit. on pp. 1, 10, 14, 63).

[NL75] George L. Nemhauser and Jr. L. E. Trotter. Vertex packings: Structural properties and algorithms. English. In: *Mathematical Programming* 8.1 (1975), pp. 232–248. ISSN: 0025-5610. DOI: 10.1007/BF01580444 (cit. on p. 63).

[Nor06] Jakob Nordström. Narrow Proofs May Be Spacious: Separating Space and Width in Resolution. In: *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC'06)*. Ed. by Jon Kleinberg. Seattle, WA, USA: Assoc. Comput. Mach., New York, May 2006, pp. 507–516. ISBN: 1-59593-134-1. DOI: 10.1145/1132516.1132590 (cit. on p. 13).

[Nor14] Jakob Nordström. A (Biased) Proof Complexity Survey for SAT Practitioners. In: *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT'14)*. Ed. by Carsten Sinz and Uwe Egly. Lecture Notes in Computer Science. Held as Part of the Vienna Summer of Logic, VSL 2014. Vienna, Austria: Springer Verlag, July 2014, pp. 1–6. ISBN: 978-3-319-09283-6. DOI: 10.1007/978-3-319-09284-3_1 (cit. on p. 13).

[NR94] Ilkka Niemelä and Jussi Rintanen. On the impact of stratification on the complexity of nonmonotonic reasoning. In: *J. Applied Non-Classical Logics* 4.2 (1994), pp. 141–179. DOI: 10.1080/11663081.1994.10510830. eprint: http://www.tandfonline.com/doi/pdf/10.1080/11663081.1994.10510830 (cit. on p. 43).

[NRS04] Naomi Nishimura, Prabhakar Ragde, and Stefan Szeider. Detecting Backdoor Sets with Respect to Horn and Binary Clauses. In: *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*. Ed. by Holger H. Hoos and David G. Mitchell. Vol. 3542. Lecture Notes in Computer Science. Vancouver, BC, Canada: Springer Verlag, May 2004, pp. 96–103 (cit. on pp. 24, 33, 49, 50).

[NRS07] Naomi Nishimura, Prabhakar Ragde, and Stefan Szeider. Solving #SAT using Vertex Covers. In: *Acta Informatica* 44.7-8 (2007), pp. 509–523. ISSN: 0001-5903. DOI: 10.1007/s00236-007-0056-x (cit. on p. 33).

[NSS99] Ilkka Niemelä, P. Simons, and T. Soininen. Stable model semantics of weight constraint rules. In: *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'99)*. Ed. by Michael Gelfond, Nicola Leone, and Gerald Pfeifer. Vol. 1730. Lecture Notes in Computer Science. El Paso, TX, USA: Springer Verlag, Dec. 1999, pp. 317–331. ISBN: 3-540-66749-0. DOI: 10.1007/3-540-46767-X_23 (cit. on p. 11).

[OJ14] Emilia Oikarinen and Matti Järvisalo. Answer Set Solver Backdoors. English. In: *Proceedings of the 14th European Conference on Logics in Artificial Intelligence (JELIA'14)*. Ed. by Eduardo Fermé and João Leite. Vol. 8761. Lecture Notes in Computer Science. Madeira, Portugal: Springer Verlag, Sept. 2014, pp. 674–683. ISBN: 978-3-319-11557-3. DOI: 10.1007/978-3-319-11558-0_51 (cit. on p. 50).

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994, p. 523. ISBN: 0-470-86412-5 (cit. on pp. 14, 15, 17).

[PD07] Knot Pipatsrisawat and Adnan Darwiche. A lightweight component caching scheme for satisfiability solvers. In: *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing (SAT'07)*. Ed. by João P. Marques-Silva and Karem A. Sakallah. Vol. 4501. Lecture Notes in Computer Science. Lisbon, Portugal: Springer Verlag, May 2007, pp. 294–299. ISBN: 978-3-540-72787-3. DOI: 10.1007/978-3-540-72788-0_28 (cit. on p. 130).

[PD11] Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers as resolution engines. In: *Artificial Intelligence* 175.2 (2011), pp. 512–525. ISSN: 0004-3702. DOI: 10.1016/j.artint.2010.10.002 (cit. on pp. 2, 13).

[Pic+14] Reinhard Pichler et al. Tractable answer-set programming with weight constraints: bounded treewidth is not enough. In: *Theory Pract. Log. Program.* 14 (02 Mar. 2014), pp. 141–164. ISSN: 1475-3081. DOI: 10.1017/S1471068412000099 (cit. on p. 2).

[Pon+12] Enrico Pontelli et al. Answer Set Programming and Planning with Knowledge and World-Altering Actions in Multiple Agent Domains. In: *Correct Reasoning – Essays on Logic-Based AI in Honour of Vladimir Lifschitz*. Ed. by Esra Erdem et al. Vol. 7265. Lecture Notes in Computer Science. Springer Verlag, 2012, pp. 509–526. ISBN: 978-3-642-30742-3. DOI: 10.1007/978-3-642-30743-0_35 (cit. on p. 12).

[PRS13] Andreas Pfandler, Stefan Rümmele, and Stefan Szeider. Backdoors to Abduction. In: *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'13)*. Ed. by Francesca Rossi. Beijing, China: The AAAI Press, Aug. 2013, pp. 1046–1052. ISBN: 978-1-57735-633-2 (cit. on pp. 2, 33, 122, 134).

[PRW09] Reinhard Pichler, Stefan Rümmele, and Stefan Woltran. Belief Revision with Bounded Treewidth. In: *Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*. Ed. by Esra Erdem, Fangzhen Lin, and Torsten Schaub. Vol. 5753. Lecture Notes in Computer Science. Potsdam, Germany: Springer Verlag, Sept. 2009, pp. 250–263. ISBN: 978-3-642-04237-9. DOI: 10.1007/978-3-642-04238-6_22 (cit. on p. 96).

[PY96] Christos H. Papadimitriou and Mihalis Yannakakis. On Limited Nondeterminism and the Complexity of the V-C Dimension. In: *J. of Computer and System Sciences* 53.2 (1996), pp. 161 –170. ISSN: 0022-0000. DOI: 10.1006/jcss.1996.0058 (cit. on p. 64).

[Ric+12] Francesco Ricca et al. Team-building with answer set programming in the Gioia-Tauro seaport. In: *Theory Pract. Log. Program.* 12 (03 Apr. 2012), pp. 361–381. ISSN: 1475-3081. DOI: 10.1017/S147106841100007X (cit. on pp. 1, 12).

[RKH04] Yongshao Ruan, Henry A. Kautz, and Eric Horvitz. The Backdoor Key: A Path to Understanding Problem Hardness. In: *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI'04)*. Ed. by Deborah L. McGuinness and George Ferguson. San Jose, CA, USA: The AAAI Press, July 2004, pp. 124–130. ISBN: 0-262-51183-5 (cit. on p. 33).

[RO09] Igor Razgon and Barry O'Sullivan. Almost 2-SAT is fixed parameter tractable. In: *J. of Computer and System Sciences* 75.8 (2009), pp. 435–450. ISSN: 0022-0000. DOI: 10.1016/j.jcss.2009.04.002 (cit. on p. 33).

[Ros10] Frances Rosamond. *Table of Races*. http://fpt.wikidot.com/. 2010 (cit. on pp. 60, 65).

[Ros95] Guido van Rossum. *Python Tutorial*. Tech. rep. CS-R9526. Amsterdam: Centrum voor Wiskunde en Informatica (CWI), May 1995 (cit. on p. 126).

[RS84] Neil Robertson and P.D. Seymour. Graph Minors. III. Planar Tree-Width. In: *J. Combin. Theory Ser. B* 36.1 (1984), pp. 49–64. ISSN: 0095-8956. DOI: 10.1016/0095-8956(84)90013-3 (cit. on pp. 21, 96).

[RS85] Neil Robertson and P.D. Seymour. Graph Minors – a Survey. In: *Surveys in Combinatorics 1985: Invited Papers for the 10th British Combinatorial Conference*. Ed. by Ian Anderson. London Mathematical Society Lecture Note Series. Cambridge University Press, Cambridge, 1985, pp. 153–171. ISBN: 0521315247 (cit. on p. 96).

[RS86] Neil Robertson and P.D. Seymour. Graph Minors. II. Algorithmic Aspects of Tree-Width. In: *J. Algorithms* 7.3 (1986), pp. 309–322. ISSN: 0196-6774. DOI: 10.1016/0196-6774(86)90023-4 (cit. on pp. 96, 97, 101).

[RST99] Neil Robertson, P.D. Seymour, and Robin Thomas. Permanents, Pfaffian Orientations, and Even Directed Circuits. In: *Annals of Mathematics* 150.3 (1999), pp. 929–975. ISSN: 0003-486X (cit. on p. 49).

[Sav70] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. In: *J. of Computer and System Sciences* 4.2 (1970), pp. 177–192. ISSN: 0022-0000. DOI: 10.1016/S0022-0000(70)80006-X (cit. on p. 13).

[Sch78] Thomas J. Schaefer. The complexity of satisfiability problems. In: *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC'78)*. Ed. by Richard J. Lipton et al. San Diego, CA, USA: Assoc. Comput. Mach., New York, 1978, pp. 216–226. DOI: 10.1145/800133.804350 (cit. on pp. 17, 50).

[Sch81] C. P. Schnorr. On Self-Transformable Combinatorial Problems. In: *Mathematical Programming at Oberwolfach*. Ed. by H. König, B. Korte, and K. Ritter. Vol. 14. Mathematical Programming Studies. Springer Verlag, 1981, pp. 225–243. ISBN: 978-3-642-00805-4. DOI: 10.1007/BFb0120931 (cit. on pp. 32, 77).

[Sch98] Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1998. ISBN: 0-471-98232-6 (cit. on p. 126).

[Sim08] Patrik Simons. *Smodels*. http://www.tcs.hut.fi/Software/smodels. July 2008 (cit. on p. 11).

[Sin05] Carsten Sinz. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In: *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP'05)*. Ed. by Peter van Beek. Vol. 3709. Lecture Notes in Computer Science. Sitges (Barcelona), Spain: Springer Verlag, Oct. 2005, pp. 827–831. ISBN: 978-3-540-29238-8. DOI: 10.1007/11564751_73 (cit. on p. 126).

[SM73] Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time. In: *Proceedings of the 5th Annual ACM Symposium on Theory of Computing (STOC'73)*. Ed. by Alfred V. Aho et al. Austin, TX, USA: Assoc. Comput. Mach., New York, Apr. 1973, pp. 1–9. DOI: 10.1145/800125.804029 (cit. on pp. 15, 17).

[SNS02] Patrik Simons, Ilkka Niemelä, and Timo Soininen. Extending and implementing the stable model semantics. In: *Artificial Intelligence* 138.1-2 (2002), pp. 181–234. ISSN: 0004-3702. DOI: 10.1016/S0004-3702(02)00187-X (cit. on pp. 11, 126).

[SS08] Marko Samer and Stefan Szeider. Backdoor Trees. In: *Proceedings of 23rd Conference on Artificial Intelligence (AAAI'08)*. Ed. by Robert C. Holte and Adele E. Howe. Chicago, IL, USA: The AAAI Press, July 2008, pp. 363–368. ISBN: 978-1-57735-368-3 (cit. on pp. 33, 67, 68, 79–81).

[SS09a] Marko Samer and Stefan Szeider. Backdoor sets of quantified Boolean formulas. In: *J. Automated Reasoning* 42.1 (2009), pp. 77–97. DOI: 10.1007/s10817-008-9114-5 (cit. on pp. 2, 33).

[SS09b] Marko Samer and Stefan Szeider. Fixed-Parameter Tractability. In: *Handbook of Satisfiability*. Ed. by Armin Biere et al. IOS Press, 2009. Chap. 13, pp. 425–454. ISBN: 978-1-58603-929-5 (cit. on p. 33).

[Sto76] Larry J. Stockmeyer. The polynomial-time hierarchy. In: *Theoretical Computer Science* 3.1 (1976), pp. 1–22. ISSN: 0304-3975. DOI: 10.1016/0304-3975(76)90061-X (cit. on p. 15).

[Str00] Ofer Strichman. Tuning SAT Checkers for Bounded Model Checking. In: *Proceedings of the 12th International Conference on Computer Aided Verification (CAV'00)*. Ed. by E. Allen Emerson and Aravinda Prasad Sistla. Vol. 1855. Lecture Notes in Computer Science. Chicago, IL, USA: Springer Verlag, July 2000, pp. 480–494. ISBN: 978-3-540-67770-3. DOI: 10.1007/10722167_36 (cit. on p. 129).

[Syr09] Tommi Syrjänen. Logic Programs and Cardinality Constraints: Theory and Practice. PhD thesis. Helsinki University of Technology, 2009 (cit. on p. 11).

[Sze08] Stefan Szeider. Matched Formulas and Backdoor Sets. In: *J. on Satisfiability, Boolean Modeling and Computation* 6 (2008), pp. 1–12. ISSN: 1574-0617 (cit. on pp. 33, 34).

[Tho09] Stéphan Thomassé. A Quadratic Kernel for Feedback Vertex Set. In: *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'09)*. Ed. by Claire Mathieu. New York, NY, USA: Society for Industrial and Applied Mathematics (SIAM), Jan. 2009, pp. 115–119. ISBN: 978-0-89871-680-1. DOI: 10.1137/1.9781611973068.13 (cit. on pp. 61, 65).

[TPS09] Son Thanh To, Enrico Pontelli, and Tran Cao Son. A Conformant Planner with Explicit Disjunctive Representation of Belief States. In: *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*. Ed. by Alfonso Gerevini et al. Thessaloniki, Greece: The AAAI Press, Sept. 2009, pp. 305–312. ISBN: 978-1-57735-406-2 (cit. on p. 126).

[Tru11] Mirosław Truszczyński. Trichotomy and dichotomy results on the complexity of reasoning with disjunctive logic programs. In: *Theory Pract. Log. Program.* 11 (06 Nov. 2011), pp. 881–904. ISSN: 1475-3081. DOI: 10.1017/S1471068410000463 (cit. on pp. 17, 135).

[Tse68] G. S. Tseitin. On the Complexity of Derivation in Propositional Calculus. In: *Zap. Nauchn. Sem. Leningrad Otd. Mat. Inst. Akad. Nauk SSSR* 8 (1968). Russian. English translation in J. Siekmann and G. Wrightson (eds.) *Automation of Reasoning. Classical Papers on Computer Science 1967–1970*, Springer Verlag, 466–483, 1983, pp. 23–41. DOI: 10.1007/978-3-642-81955-1_28 (cit. on p. 118).

[Val79] Leslie G. Valiant. The complexity of computing the permanent. In: *Theoretical Computer Science* 8.2 (1979), pp. 189–201. ISSN: 0304-3975. DOI: 10.1016/0304-3975(79)90044-6 (cit. on p. 16).

[Var14] Moshe Y. Vardi. Boolean Satisfiability: Theory and Engineering. In: *Communications of the ACM* 57.3 (Mar. 2014), pp. 5–5. ISSN: 0001-0782. DOI: 10.1145/2578043 (cit. on pp. 1, 13).

[VK76] M. H. Van Emden and Robert. A. Kowalski. The Semantics of Predicate Logic as a Programming Language. In: *J. of the ACM* 23 (4 Oct. 1976), pp. 733–742. ISSN: 0004-5411. DOI: 10.1145/321978.321991 (cit. on p. 9).

[VY88] Vijay Vazirani and Mihalis Yannakakis. Pfaffian Orientations, 0/1 Permanents, and Even Cycles in Directed Graphs. In: *Proceedings of the 15th International Colloquium on Automata, Languages and Programming (ICALP'88)*. Ed. by Timo Lepistö and Arto Salomaa. Vol. 317. Lecture Notes in Computer Science. Tampere, Finland: Springer Verlag, July 1988, pp. 667–681. ISBN: 978-3-540-19488-0. DOI: 10.1007/3-540-19488-6_149 (cit. on p. 49).

[WGS03a] Ryan Williams, Carla Gomes, and Bart Selman. Backdoors To Typical Case Complexity. In: *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*. Ed. by Georg Gottlob and Toby Walsh. Acapulco, Mexico: Morgan Kaufmann, Aug. 2003, pp. 1173–1178 (cit. on pp. 1, 2, 13, 21, 23, 33).

[WGS03b] Ryan Williams, Carla Gomes, and Bart Selman. On the Connections Between Backdoors, Restarts, and Heavy-tailedness in Combinatorial Search. In: *Informal Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*. Portofino, Italy, May 2003, pp. 222–230 (cit. on pp. 1, 2, 13, 21, 23, 33).

[Wil12] William A. Stein et.al. *Sage Mathematics Software (Version 5.1.rc0)*. http://www.sagemath.org. The Sage Development Team. 2012 (cit. on p. 126).

[Wra76] Celia Wrathall. Complete Sets and the Polynomial-Time Hierarchy. In: *Theoretical Computer Science* 3.1 (1976), pp. 23–33. ISSN: 0304-3975. DOI: 10.1016/0304-3975(76)90062-1 (cit. on pp. 15, 17).

[Yap83] Chee-K. Yap. Some consequences of nonuniform conditions on uniform classes. In: *Theoretical Computer Science* 26.3 (1983), pp. 287–300. ISSN: 0304-3975. DOI: 10.1016/0304-3975(83)90020-8 (cit. on pp. 62, 64, 65).

[Zha02] Jicheng Zhao. A Study of Answer set Programming. MPhil Thesis. The Hong Kong University of Science and Technology, Dept. of Computer Science, 2002 (cit. on pp. 17, 43).

[Zha04] Yuting Zhao. *ASSAT*. http://assat.cs.ust.hk. Oct. 2004 (cit. on p. 11).

[ZL03] Yuting Zhao and Fangzhen Lin. Answer Set Programming Phase Transition: A Study on Randomly Generated Programs. In: *Proceedings of the 19th International Conference on Logic Programming (ICLP'03)*. Ed. by Catuscia Palamidessi. Vol. 2916. Lecture Notes in Computer Science. Mumbai, India: Springer Verlag, Dec. 2003, pp. 239–253. ISBN: 978-3-540-24599-5. DOI: 10.1007/978-3-540-24599-5_17 (cit. on p. 127).

[ZM02] Lintao Zhang and Sharad Malik. Conflict Driven Learning in a Quantified Boolean Satisfiability Solver. In: *Proceedings of the 2002 IEEE/ACM International Conference on Computeraided Design (ICCAD'02)*. Ed. by Lawrence T. Pileggi and Andreas Kuehlmann. San Jose, CA, USA: Assoc. Comput. Mach., New York, Nov. 2002, pp. 442–449. ISBN: 0-7803-7607-2. DOI: 10.1145/774572.774637 (cit. on p. 13).

Curriculum Vitae

Johannes K. Fichte

Curriculum Vitae

Favoritenstraße 9-11
1040 Wien, Austria
Tel: +43-1-58801-740061
Email: fichte@kr.tuwien.ac.at

Born: April 29, 1983
Neubrandenburg, Germany
Nationality: German

Research Interest

Parameterized Complexity Theory
Answer Set Programming
Sat and Sat-based Solving
Descriptive Complexity Theory

Education

Doctoral studies in Computer Science 2010–2015
Vienna University of Technology, Austria
Principal Advisor: Stefan Szeider
Co-Advisor: Torsten Schaub

Diploma in Computer Science 2005–2009
Humboldt-Universität zu Berlin, Germany
Thesis: Abstract State Machines – Programmieren mit Strukturen
Supervisor: Martin Grohe
Cumulative grade (German): 1.6 (good)

Prediploma in Computer Science 2003–2005
University of Rostock, Germany
Cumulative grade (German): 2 (good)

2002 *High School Diploma* (Abitur) at Neues Friedländer Gymnasium
Cummulative grade (German): 1.0 (valedictorian)

Extended Research Visits

2012–2015 Torsten Schaub, Knowledge Processing and Information Systems,
University of Potsdam, Germany

Publications

Journals

1. J.K. Fichte, S. Szeider
Backdoors to Normality for Disjunctive Logic Programs.
Under Review, 2015.

2. J.K. Fichte, S. Szeider
Backdoors to tractable answer set programming
Artificial Intelligence (AIJ), 2015.

3. A. Atserias, J.K. Fichte, M. Thurley:
Clause-Learning Algorithms with Many Restarts and Bounded-Width Resolution
Journal on Artificial Intelligence Research (JAIR), 2011

Conferences

4. J.K. Fichte, M. Truszczyński, S. Woltran:
Dual-normal Logic Programs – the Forgotten Class
Proceedings of the 31st International Conference on Logic Programming (ICLP'15), 2015, To appear

5. J.K. Fichte, S. Szeider:
Backdoors to Normality for Disjunctive Logic Programs
Proc. of the 27th AAAI Conference on Artificial Intelligence (AAAI), 2013.

6. J.K. Fichte:
The Good, the Bad, and the Odd: Cycles in Answer-Set Programs
New Directions in Logic, Language and Computation (LNCS 7415), 2012

7. J.K. Fichte, S. Szeider:
Backdoors to Tractable Answer-Set Programming
Proc. of the 22nd Intl. Conference on Artificial Intelligence (IJCAI), 2011

8. J.K. Fichte:
   Backdoors to Tractability of Answer-Set Programming
   Proc. of the 18th AAAI/SIGART Doctoral Consortium (AAAI DC), 2013.

9. J.K. Fichte:
   Backdoors to Tractability of Answer-Set Programming
   Proc. of 9th ICLP Doctoral Consortium (ICLP DC), 2013.

Invited Talks

Modern SAT Solving 2015
Department of Informatics, University of Bergen, Norway

Grants

Travel Grant by Potsdam Graduate School (PoGS)	500 EUR	2014
Travel Grant by data experts	500 EUR	
Travel Grant by Potsdam Graduate School (PoGS)	300 EUR	2013
Travel Grant by the Association for Computing Machinery (ACM)	1000 USD	
Fellowship by the German National Academic Foundation		2003-2009

Participation

European Summer School in Logic, Language and Information (ESSLLI)	2011	Schools
Answer Set Programming and Other Computing Paradigms (ASPOCP)	2012, 2013	Workshops
Algorithmic Model Theory (AlMoTh)	2009, 2010, 2011	
Parameterized Complexity of Computational Reasoning (PCCR)	2010, 2014	
Workshop on Kernels (Worker)	2009	
First Symposium on Structure in Hard Combinatorial Problems (Structure)	2013	
Theoretical Foundations of Applied SAT Solving at BIRS, AB, Canada	2014	
Vienna Summer of Logic (VSL) including CSL/LICS, SAT, LCC, and LC	2014	
Symposium on New Frontiers in Knowledge Compilation	2015	

Activities

PC Member — Answer Set Programming and Other Computing Paradigms (ASPOCP'14,'15)

Volunteering — Vienna Summer of Logic (VSL) Student Volunteer 2014.

Reviews — I have externally reviewed submissions for AAAI, ASPOCP, CP, ECAI, ICLP, IJCAI, JELIA, JLC, KR, LPAR, LPNMR, MFCS, and SAT.

Employment

2014– Project Assistant
Database and Artificial Intelligence Group
Vienna University of Technology, Austria

2011– Software Engineer
Algorithmic Software Optimization, Software Development, Process Optimization, Project Consulting
data experts, Berlin, Germany

2010–2011 Project Assistant
Discrete Reasoning Methods Group,
Vienna University of Technology, Austria

Last updated: 19th August, 2015 •