Artificial Intelligence 220 (2015) 64–103

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

# Backdoors to tractable answer set programming ☆

Johannes Klaus Fichte [a,b], Stefan Szeider [a,*]

[a] Vienna University of Technology, Favoritenstrasse 9-11, 1040 Vienna, Austria
[b] University of Potsdam, August-Bebel-Strasse 89, 14482 Potsdam, Germany

ARTICLE INFO

*Article history:*
Received 7 March 2014
Received in revised form 1 December 2014
Accepted 6 December 2014
Available online 15 December 2014

*Keywords:*
Answer set programming
Backdoors
Computational complexity
Parameterized complexity
Kernelization

ABSTRACT

Answer Set Programming (ASP) is an increasingly popular framework for declarative programming that admits the description of problems by means of rules and constraints that form a disjunctive logic program. In particular, many AI problems such as reasoning in a nonmonotonic setting can be directly formulated in ASP. Although the main problems of ASP are of high computational complexity, complete for the second level of the Polynomial Hierarchy, several restrictions of ASP have been identified in the literature, under which ASP problems become tractable.

In this paper we use the concept of backdoors to identify new restrictions that make ASP problems tractable. Small backdoors are sets of atoms that represent "clever reasoning shortcuts" through the search space and represent a hidden structure in the problem input. The concept of backdoors is widely used in theoretical investigations in the areas of propositional satisfiability and constraint satisfaction. We show that it can be fruitfully adapted to ASP. We demonstrate how backdoors can serve as a unifying framework that accommodates several tractable restrictions of ASP known from the literature. Furthermore, we show how backdoors allow us to deploy recent algorithmic results from parameterized complexity theory to the domain of answer set programming.



## 1. Introduction

*Answer Set Programming* (ASP) is an increasingly popular framework for declarative programming [115,122]. ASP admits the description of problems by means of rules and constraints that form a disjunctive logic program. Solutions to the program are so-called stable models or answer sets. Many important problems of AI and reasoning can be succinctly represented and successfully solved within the ASP framework. It has been applied to several large industrial applications, e.g., social networks [97], match making [74], planning in a seaport [129], optimization of packaging of Linux distributions [69], and general game playing [145].

The main computational problems for 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 [41]; thus, 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 [78,2], for a detailed trichotomy (tractable, first level, second level of PH) see [148].

☆ Fichte and Szeider's research was supported by the European Research Council, grant reference 239962 (COMPLEX REASON). Fichte's research was partially supported by the Austrian Science Fund (FWF) project Y698.

* Corresponding author.
*E-mail addresses:* fichte@kr.tuwien.ac.at (J.K. Fichte), stefan@szeider.net (S. Szeider).

http://dx.doi.org/10.1016/j.artint.2014.12.001
0004-3702/

### 1.1. Contribution

In this paper we use the concept of *backdoors* to identify new restrictions that make propositional ASP problems tractable. Small backdoors are sets of atoms that represent "clever reasoning shortcuts" through the search space and represent a hidden structure in the problem input. Backdoors were originally introduced by Williams, Gomes, and Selman [152,153] as a tool to analyze the behavior of DPLL-based SAT solvers. Backdoors have been widely used in theoretical investigations in the area of propositional satisfiability [152,135,138,103] and constraint satisfaction [84], and also for abductive reasoning [125], argumentation [40], and quantified Boolean formulas [137]. A backdoor is defined with respect to some fixed *target class* for which the computational problem under consideration is polynomial-time tractable. The size of the backdoor can be seen as a distance measure that indicates how far the instance is from the target class.

In this paper we develop a rigorous theory of backdoors for answer set programming. 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.

For a worst-case complexity analysis of various problems involving backdoors such as finding a small backdoor or using it to solve the problem, it is crucial to investigate how the running time depends on the size of the backdoor, and how well running time scales with backdoor size. *Parameterized Complexity* [35,55,85] provides a most suitable theoretical framework for such an analysis. It provides the key notion of *fixed-parameter tractability* which, in our context, means polynomial-time tractability for fixed backdoor size, where the order of the polynomial does not depend on the backdoor size. We show how backdoors allow us to deploy recent algorithmic results from parameterized complexity theory to the domain of answer set programming.

Parameterized complexity provides tools for a rigorous analysis of *polynomial-time preprocessing* in terms of *kernelization* [7,58]. A kernelization is a polynomial-time self-reduction of a parameterized decision problem that outputs a decision equivalent problem instance whose size is bounded by a function $f$ of the parameter (the kernel size). It is known that every decidable fixed-parameter tractable problem admits a kernelization, but some problems admit small kernels (of size polynomial in the parameter) and others do not. We provide upper and lower bounds for the kernel size of the problems backdoor evaluation and backdoor detection for disjunctive answer set programs. These bounds provide worst case guarantees and limits for polynomial-time preprocessing for the considered problems.

Several algorithms in the literature are defined for disjunction-free (i.e., normal) programs only. We introduce a general method for *lifting* these parameters to disjunctive programs, preserving fixed-parameter tractability under certain conditions.

Although our main focus is on a theoretical evaluation, we present some experimental results where we consider the backdoor size of structured programs and random programs of varied density.

### 1.2. Background and related work

*Complexity of ASP problems* Answer set programming is based on the *stable-model semantics* for logic programs [78,79]. The computational complexity of various problems arising in answer set programming has been subject of extensive studies. Eiter and Gottlob [41] have established that the main decision problems of (disjunctive) ASP are complete for the second level of the Polynomial Hierarchy ($\Sigma_2^P$- or $\Pi_2^P$-complete, respectively). Moreover, Bidoít and Froidevaux [5] and Marek and Truszczyński [114] have shown that the problems are NP-complete (co-NP-complete respectively) for disjunction-free (so-called *normal*) programs. Several fragments of programs where the main reasoning problems are polynomial-time tractable have been identified, e.g., Horn programs [78], stratified programs [2] and programs without even cycles [156]. Dantsin et al. [27] survey the classical complexity of the main reasoning problems for various semantics of logic programming, including fragments of answer set programming.

*ASP solvers* Various ASP solvers have been developed in recent years. Many of them 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 [141], Assat [111], Cmodels2 [82], and Clasp2 [71]. There are also solvers that deal with the full set of disjunctive programs, e.g., Clasp3 [37], Cmodels3 [108], DLV [107], and GnT [92]. Compilations to other problem domains and respective solvers have been considered for normal programs, e.g., propositional satisfiability [91], mixed integer programming [112], satisfiability modulo theories [93,76]. We would like to point out that these solvers use heuristics without non-trivial worst-case performance guarantees. In contrast we provide for the main reasoning problems of answer set programming theoretical worst-case time bounds that take certain hidden structures in disjunctive programs into account.

Preprocessing techniques and unit propagation used in solvers might be considered in a wider sense as implicitly exploiting Horn fragments. Grounders like Gringo [64] already solve Horn programs simply by propagating atoms which trivially (do not) belong to the minimal model, e.g., atoms that occur in the head of rules with an empty body, atoms that occur in the head of rules where all atoms in the positive body already belong to the minimal model, atoms that cannot belong to the minimal model according to some constraint, and atoms that cannot belong to the minimal model as they occur in no head. Moreover, SAT-based solvers like Clasp [62] transform the program into a propositional formula using Clark's completion (see e.g., [73]) where the resulting formula characterizes the classical models and necessary conditions for atoms to belong to a model. If the program contains no cycles in its positive dependency graph, unit propagation (as part of a

DPLL-based algorithm) already solves Horn fragments. Otherwise, loop formulas are added which prevent the solver from assigning true to atoms that occur on a cycle unless some atom from outside is set to true. If there is an atom from outside that has been assigned to true, atoms on the cycle are set again to true by unit propagation and so forth.

*Parameterizations of ASP* 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 and provide fixed-parameter tractability. The considered parameters include the number of atoms of a normal program that appear in negative rule bodies [4], the number of non-Horn rules of a normal program [4], the size of a smallest feedback vertex set in the dependency digraph of a normal program [85], the number of cycles of even length in the dependency digraph of a normal program [110], the treewidth of the incidence graph of a normal program [90,119], 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 programs [3]. Very recently we established [48] an fpt-reduction that reduces disjunctive ASP to normal ASP; in other words, a reduction from the second level of the Polynomial Hierarchy to the first level. However, these results do not provide fixed-parameter tractability of the ASP reasoning problems, and hence are not directly comparable to the results presented in this paper. A general theoretical framework to classify parameterized problems on whether they admit an fpt-reduction to SAT or not has lately been introduced in [28]. Similar to our result in [48] this does not provide fixed-parameter tractability of the ASP reasoning problems, but in certain cases fixed-parameter tractability of the reduction to SAT.

*Backdoors* The concept of a backdoor originates in SAT/CSP and was introduced by Williams et al. [152,153]. Since then, backdoors have been used frequently in the literature for theoretical investigations. The study of the parameterized complexity of backdoor detection was initiated by Nishimura et al. [123] who considered satisfiability backdoors for the base classes Horn and 2CNF. Since then, the study has been extended to various other base classes, including clustering formulas [124], renamable Horn formulas [128], QHorn formulas [59], Nested formulas [56], acyclic formulas [54], and formulas of bounded incidence treewidth [57]; for a survey, see [55]. Several results extend the concept of backdoors to other problems, e.g., backdoor sets for constraint satisfaction problems [152], quantified Boolean formulas [137], abstract argumentation [40], and abductive reasoning [125]. Samer and Szeider [136] 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 et al. [31,32] 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, they have also established that backdoor detection for the base classes Horn and 2CNF is already harder than NP when empty clause detection is added. Moreover, Szeider [144] has shown that (strong) backdoor detection is W[1]-hard for almost all base classes when empty clause detection is added and thus unlikely to be fixed-parameter tractable.

### 1.3. Prior work and paper organization

This paper is an extended and updated version of the papers that appeared in the proceedings of the 22nd International Joint Conference on Artificial Intelligence [47] and in the selected papers proceedings of the Student Session of the 23rd European Summer School in Logic, Language, and Information [49]. The present paper provides a higher level of detail, in particular full proofs and more examples. Furthermore, the paper extends its previous versions in the following way: additional attention is payed to the minimality check (Lemma 3.7). Theorem 5.13 is extended to entail some very recent results in parameterized complexity theory. A completely new section (Section 6) is devoted to a rigorous analysis of preprocessing methods for the problems of backdoor detection and backdoor evaluation. We present a general method to lift parameters from rules of normal programs to disjunctive programs (Section 7). We extend the section on the theoretical comparison of parameters (Section 8) by additional comparisons to other parameters, e.g., weak feedback width and interaction graph treewidth, and to other classes of programs, e.g., head-cycle-free and tight programs. Additionally, we provide some empirical data on backdoor detection and discuss the evaluation of backdoors in a practical setting in Section 4.1.

## 2. Preliminaries

### 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 following form

$$x_1 \vee \ldots \vee x_l \quad \leftarrow \quad y_1, \ldots, y_m, \neg z_1, \ldots, \neg z_n$$

where $x_1, \ldots, x_l, y_1, \ldots, y_m, z_1, \ldots, z_n$ are atoms and $l, m, n$ are non-negative integers. Let $r$ be a rule. We write $\{x_1, \ldots, x_l\} = H(r)$ (the *head* of $r$), $\{y_1, \ldots, y_m\} = B^+(r)$ (the positive body of $r$) and $\{z_1, \ldots, z_n\} = B^-(r)$ (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. A rule $r$ is *negation-free* if $B^-(r) = \emptyset$, *normal* if $|H(r)| \leq 1$, a *constraint* if $|H(r)| = 0$, *constraint-free* if $|H(r) > 0|$, *Horn* if it is normal and negation-free or a constraint, *positive* if it is Horn and constraint-free, *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 $H(r') \subseteq H(r)$, $B^+(r') \subseteq B^+(r)$, $B^-(r') \subseteq B^-(r)$. 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. Note that many natural classes of programs (and all classes considered in this paper) are hereditary.

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 second removing all $\neg z$ where $z \in B^-(r)$ from the remaining rules $r$ [79]. $M$ is an *answer set* (or *stable model*) of a program $P$ if $M$ is a minimal model of $P^M$. We denote by $\mathrm{AS}(P)$ the set of all answer sets of $P$.

**Example 2.1.** Consider the program $P$ consisting of the following rules:

$$d \leftarrow a, e; \quad a \leftarrow d, \neg b, \neg c; \quad e \vee c \leftarrow f;$$
$$f \leftarrow d, c; \quad c \leftarrow f, e, \neg b; \quad c \leftarrow d;$$
$$b \leftarrow c; \quad f.$$

The set $M = \{b, c, f\}$ is an answer set of $P$, since $P^M = \{d \leftarrow a, e;\ f \leftarrow d, c;\ b \leftarrow c;\ e \vee c \leftarrow f;\ c \leftarrow d;\ f\}$ and the minimal models of $P^M$ are $\{b, c, f\}$ and $\{e, f\}$.

In this paper we generally assume that programs contain no tautological rules since one can simply remove tautological rules from a program without effecting the answer sets, i.e., $\mathrm{AS}(P) = \mathrm{AS}(P')$ where $P'$ is a program obtained from $P'$ by removing all tautological rules [13, The. 5.5] or [42]. In one case we allow tautological rules and state that explicitly (Proposition 5.10). Moreover, we generally assume that programs contain no rules $r$ where $H(r) \cap B^-(r) \neq \emptyset$ since one can simply remove from those rules the head atoms in $H(r) \cap B^-(r)$ without effecting the answer sets, i.e., $\mathrm{AS}(P) = \mathrm{AS}(P')$ where $P'$ is a program obtained from $P'$ by setting $H(r) := H(r) \setminus B^-(r)$ for every rule $r$ where $H(r) \cap B^-(r) \neq \emptyset$ [42].

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 [149] have shown that every constraint-free Horn program has a unique minimal model. Dowling and Gallier [33] 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.*

### 2.2. 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 $\mathcal{AspReason}$ the family of the reasoning problems CHECKING, CONSISTENCY, BRAVE REASONING, and SKEPTICAL REASONING. We denote by $\mathcal{AspFull}$ the family of all the problems defined above. The family $\mathcal{AspReason}$ consists of decision problems, and $\mathcal{AspFull}$ adds to it a counting and an enumeration problem. In the sequel we will occasionally write $L_{\mathbf{Normal}}$ to denote a problem $L \in \mathcal{AspFull}$ restricted to input programs from **Normal**.

CHECKING is co-NP-complete in general [41], but $\text{CHECKING}_{\mathbf{Normal}}$ is polynomial [14]. CONSISTENCY and BRAVE REASONING are $\Sigma_2^P$-complete, SKEPTICAL REASONING is $\Pi_2^P$-complete [41]. Both reasoning problems are NP-complete (or co-NP-complete) for normal programs [113], but are polynomial-time solvable for Horn programs [78]. COUNTING is easily seen to be $\#P$-hard[1] as it entails the problem #SAT.

### 2.3. Parameterized complexity

Problem instances that originate from practical applications are often structured in a certain way that facilitates to obtain a solution relatively fast. Such instances seem to be harder in theory, where worst-case running times are given in terms of the input size in bits, than they are in practice. The framework of *parameterized complexity*, introduced by Downey and Fellows [34], takes structural properties of problem instances in form of a parameter into account. In consequence, it offers a framework for a more detailed theoretical analysis that is closer to the practical hardness of problems. In recent years parameterized complexity theory has become a very active research area. Since there are many ways of defining and capturing structure in a problem instance, there are 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. For more detailed information we refer to other sources [34,35,50,84,120]. An instance of a *parameterized problem* $L$ is a pair $(I,k) \in \Sigma^* \times \mathbb{N}$ for some finite alphabet $\Sigma$. 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 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 $\Sigma$ and $\Sigma'$. 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 $r(I,k) = (I',k') \in L'$ and $k' \leq g(k)$ 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 $\mathcal{O}(f(k)\|I\|^c)$. 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.

The *Weft Hierarchy* consists of parameterized complexity classes $\text{W}[1] \subseteq \text{W}[2] \subseteq \cdots$ 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 $\text{W}[i]$ are not fixed-parameter tractable. A prominent W[2]-complete problem is HITTING SET [34,35] defined as follows:

HITTING SET

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

The class XP of *non-uniform* 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 $f$. The parameterized complexity class paraNP 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 $c$. A parameterized decision problem is paraNP-complete if it is in NP and NP-complete when restricted to finitely many parameter values [50]. By co-paraNP we denote the class of all parameterized decision problems whose complement (yes and no instances swapped) is in paraNP. Using the concepts and terminology of Flum and Grohe [50], co-paraNP = para-coNP.

### 2.4. Graphs

We recall some notations of graph theory. 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

[1] $\#P$ is the complexity class consisting of all the counting problems associated with the decision problems in NP.

edge $\{v, w\}$ 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 $v$.

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 $(u, v) \in E'$. 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. Please observe 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 $\mathrm{SCC}(G)$.

For further basic terminology on graphs and digraphs we refer to a standard text [30,11].

### 2.5. Satisfiability backdoors

We also need some notions from *propositional satisfiability*. A *literal* is an atom or its negation and a *clause* is a finite set of literals A CNF formula is a finite set of clauses. A *truth assignment* is a mapping $\tau : X \to \{0, 1\}$ defined for a set $X \subseteq U$ of atoms. For $x \in X$ we put $\tau(\neg x) = 1 - \tau(x)$. By $2^X$ we denote the set of all truth assignments $\tau : X \to \{0, 1\}$. The *truth assignment reduct* of a CNF formula $F$ with respect to $\tau \in 2^X$ is the CNF formula $F_\tau$ obtained from $F$ by first removing all clauses $c$ that contain a literal set to 1 by $\tau$, and second removing from the remaining clauses all literals set to 0 by $\tau$. $\tau$ *satisfies* $F$ if $F_\tau = \emptyset$, and $F$ is *satisfiable* if it is satisfied by some $\tau$.

The following is obvious from the definitions:

**Observation 2.2.** *Let $F$ be a CNF formula and $X$ a set of atoms. $F$ is satisfiable if and only if $F_\tau$ 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 $\mathcal{O}(n^c)$. 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 $\mathcal{O}(2^k n^c)$ 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 [123]. The parameterized complexity of backdoor detection for many further target classes has been investigated [55].

The purpose of this paper 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) and various target classes for answer set programming require new algorithms for backdoor detection (see Section 5).

## 3. Answer set backdoors

### 3.1. Strong backdoors

In order to translate the notion of backdoors to the domain of ASP, 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 et al. [85].

**Definition 3.1.** Let $P$ be a program, $X$ a set of atoms, and $\tau \in 2^X$. The *truth assignment reduct* of $P$ under $\tau$ is the logic program $P_\tau$ 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 3.2.** 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.** We consider the program of Example 2.1. The set $\{b, c\}$ is a strong **Horn**-backdoor since all four truth assignment reducts $P_{\bar{b}\bar{c}} = P_{b=0,c=0} = \{d \leftarrow a, e;\ a \leftarrow d;\ e \leftarrow f;\ f\}$, $P_{\bar{b}c} = \{d \leftarrow a, e;\ f \leftarrow d;\ f\}$, $P_{b\bar{c}} = \{d \leftarrow a, e;\ e \leftarrow f;\ f\}$, and $P_{bc} = \{d \leftarrow a, e;\ f \leftarrow d;\ f\}$ are in the class **Horn**.

### 3.2. Deletion backdoors

Next we define a variant of answer set backdoors similar to satisfiability deletion backdoors. 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 3.3.** 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 base class to satisfy the very mild condition of being hereditary (see Section 2) which holds for all base classes considered in this paper.

**Lemma 3.4.** *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 3.1 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 2.2 does not hold for ASP, 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 2.2, where we can map each answer set of $P$ to an answer set of $P_\tau$ 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 3.5.** Let $P$ be a program and $X$ a set of atoms. We define

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

In other words, the sets in $\mathrm{AS}(P, X)$ are answer sets of $P_\tau$ for truth assignments $\tau$ to $X \cap \mathrm{at}(P)$ extended by those atoms which are set to true by $\tau$. 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 3.6.** $\mathrm{AS}(P) \subseteq \mathrm{AS}(P, X)$ *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 $P_\tau$) 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 $P_\tau$ 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 $P_\tau$, and so the lemma follows. □

In view of Lemma 3.6 we shall refer to the elements in $\mathrm{AS}(P, X)$ as "answer set candidates."

**Example 3.2.** We consider program $P$ of Example 2.1 and the strong **Horn**-backdoor $X = \{b, c\}$ of Example 3.1. The answer sets of $P_\tau$ are $\mathrm{AS}(P_{\bar{b}\bar{c}}) = \{\{e, f\}\}$, $\mathrm{AS}(P_{\bar{b}c}) = \{\{f\}\}$, $\mathrm{AS}(P_{b\bar{c}}) = \{\{e, f\}\}$, 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 3.6, 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 $P_\tau$ 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 [113], but polynomial for normal programs [14].

Fortunately, for our considerations it suffices to perform check (ii.b) for programs that are "close to **Normal**," 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 $|X|$ of the backdoor.
*Task:* Decide whether $M$ is an answer set of $P$.

**Lemma 3.7.** *Let* $\mathcal{C}$ *be a class of normal programs. The problem* Strong $\mathcal{C}$-Backdoor Asp Check *is fixed-parameter tractable.*

**Proof.** Let $\mathcal{C}$ be a class of normal programs, $P$ a program, and $X$ a strong $\mathcal{C}$-backdoor $X$ of $P$ with $|X| = k$. 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 $P_\tau$, 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 $1 \geq |H(r^*)| = |H(r) \setminus X| = |H(r')|$, and the claim follows.

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

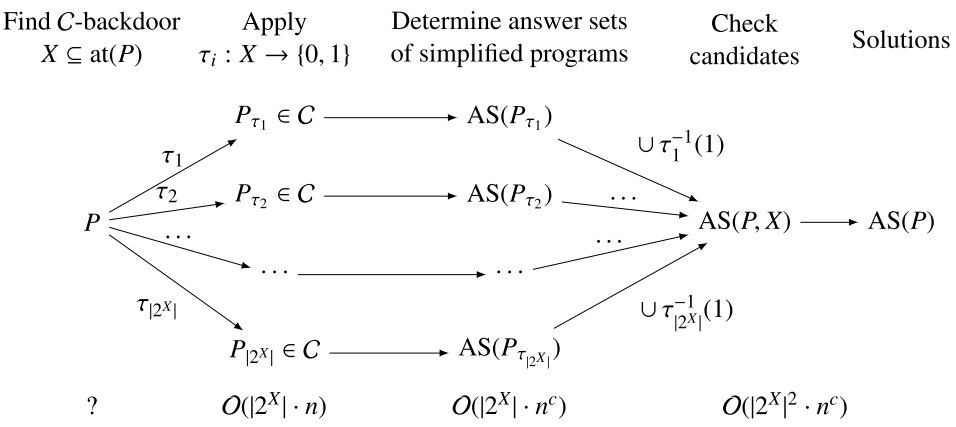


**Fig. 1.** Exploit pattern of ASP backdoors if the target class $\mathcal{C}$ is normal and enumerable where $n$ denotes the input size of $P$.

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: (ii.a) $L$ is not a subset of $M \setminus X$, or (ii.b) $L \cup X_1$ is not a proper subset of $M$, or (ii.c) $L \cup X_1$ 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 (ii.a): We have $L \subseteq M \setminus X$ since $L \subseteq M' \setminus X$ and $M' \setminus X \subseteq M \setminus X$. Case (ii.b): 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 (ii.c): And finally $L \cup X_1$ is a model of $P^M$, 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, and so the lemma follows. □

Fig. 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 $P_\tau$ 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 3.3.** We consider the set $\mathrm{AS}(P,X) = \{\{e,f\},\{c,f\},\{b,e,f\},\{b,c,f\}\}$ of answer set candidates of Example 3.2 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 3.7.

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 Lemma 3.7 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 3.6 and 3.7, 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 3.8.** 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*.

Please 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 8.4 for a certain parameter.

**Theorem 3.9.** *Let $\mathcal{C}$ be an enumerable class of normal programs. The problems in $\mathcal{A}sp\mathcal{F}ull$ 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 3.6, $\mathrm{AS}(P) \subseteq \mathrm{AS}(P,X)$. By means of Lemma 3.7 we can decide whether $M \in \mathrm{AS}(P)$ 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 $\mathcal{A}sp\mathcal{F}ull$ within polynomial time. □

Theorem 3.9 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 3.9 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 3.9 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 $k$.

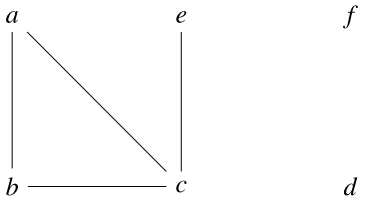


**Fig. 2.** Negation dependency graph $N_P$ of the program $P$ of Example 2.1.

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 [139,34,35], one can use a decision algorithm for Deletion $\mathcal{C}$-Backdoor Detection to actually find the backdoor. We only require the base class to be hereditary.

**Lemma 3.10.** *Let $\mathcal{C}$ be a hereditary class of programs. If* Deletion $\mathcal{C}$-Backdoor Detection *is fixed-parameter tractable, then also finding a 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 $x \in \mathrm{at}(P)$ whether $P - \{x\}$ has a deletion $\mathcal{C}$-backdoor of size at most $k - 1$. If the answer is NO for all $x$, then $P$ has no deletion $\mathcal{C}$-backdoor of size $k$. If the answer is YES for $x$, then by induction hypothesis we can compute a deletion $\mathcal{C}$-backdoor $X$ of size at most $k - 1$ of $P - x$, and $X \cup \{x\}$ 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" [31, 32] 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 [144]. These W[1]-hardness results carry over to the ASP setting if empty rule detection is added.

## 4. 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 3.4 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 $\tau$ is sound. It follows that $r' \in P_\tau$, contradicting the assumption that $X$ is a strong **Horn**-backdoor of $P$. □

**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.** Fig. 2 visualizes the negation dependency graph $N_P$ of the program $P$ of 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 the program $P$ of 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[2] 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 et al. [22]. 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 3.9 by dropping the assumption that the backdoor is given.

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

### 4.1. Horn-backdoors of benchmark instances

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 the size of backdoors for various benchmark programs, focusing on the target class **Horn**. As expected, structured programs, originating from application domains, have smaller backdoors than random instances. Since the direct implementation of our backdoor-based algorithms seems impracticable.

We have determined strong **Horn**-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 $1\{c_u, c_v\} \leftarrow e_{u,v}$ Moreover, we add a statement to minimize the number of atoms in $C$ that belong to an answer set (see [141,73] 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 [15, 17,16] on the graph $N_p$. For the SAT encoding we used an encoding similar to the encoding presented in [53]. We introduce for every edge $uv \in E$ a binary clause and add a sequential unary counter [142] to express that at most $k$ vertices belong to a vertex cover.

The answer set program that solves backdoor detection was generated by means of ASP meta programming [70] and solved using Clasp [71,72] version 3.0.5 with an unsatisfiable-core based optimization strategy (the command line parameter "–opt-strategy=5" yields the behavior) [1]. The integer linear program was generated using the open source mathematics framework Sage [44] with Python [150], solved using ILOG CPLEX 12 [89] and Gurobi [87].

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

**Table 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.

| Domain | Instance set | Disj. | #instances | #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 | − | 282 | 200.0 | 67.26 | 5.42 |
| | RandomNonTight | − | 220 | 200.0 | 89.85 | 0.24 |

ConformantPlanning: secure planning under incomplete initial states [147]; instances provided by Gebser and Kaminski [60]. Factoring: factorization of a number where an efficient algorithm would yield a cryptographic attack by Gebser [38]; for instances see [61]. HanoiTower: classic Towers of Hanoi puzzle by Truszczynski, Smith and Westlund; for instances see [18]. GraphColoring: classic graph coloring problem by Lierler and Balduccini; for instances see [18]. KnightTour: finding a tour for the knight piece traveling any square following the rules of chess by Zhou, Calimeri, and Santoro; for instances see [18]. Labyrinth: classical Ravensburger's Labyrinth puzzle by Gebser; for instances see [18]. MinimalDiagnosis: an application in systems biology [67]; for instances see [18]. MSS/MUS: problem whether a clause belongs to some minimal unsatisfiable subset [94]; instances provided by Gebser and Kaminski [60]. Solitaire: classical Peg Solitaire puzzle by Lierler and Balduccini; for instances see [18]. StrategicCompanies: encoding the $\Sigma_2^P$-complete problem of producing and owning companies and strategic sets between the companies [63]. RandomQBF: translations of randomly generated 2-QBF instances using the method by Chen and Interian [20]; for instances see [63]. RLP: Randomly generated normal programs, of various density (number of rules divided by the number of atoms) [155]; for instances see [63]. RandomNonTight: Randomly generated normal programs in [61] with $n = 40$, 50, and 60 variables, respectively with 40 instances per step instances.

The results obtained with ILP methods using modern solvers like CPLEX and Gurobi come along with a certain inaccuracy (see e.g., [25,24,140]). Therefore, we ran Clasp on the structured instances using the encoding of $k$-vertex cover problem described above to obtain optimality.

Table 1 illustrates our results on the size of small strong **Horn**-backdoors of the considered benchmark instances. We mainly used benchmark sets from the first three Answer Set Programming Competitions [18,29,63], because most of the instances contain only normal and/or disjunctive rules and no extended rules (cardinality/weight-constraints).[3] We reference in the caption of Table 1 from where the instances have been taken and indicate in the table the number of instances of each benchmark set.

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.

## 5. 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 et al. [2] 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 et al. [85].

**Definition 5.1.** 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 5.2.** 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 $uv$.

[3] 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 [66], those compilations blow up the instances significantly. Hence we omitted it for pragmatic reasons.

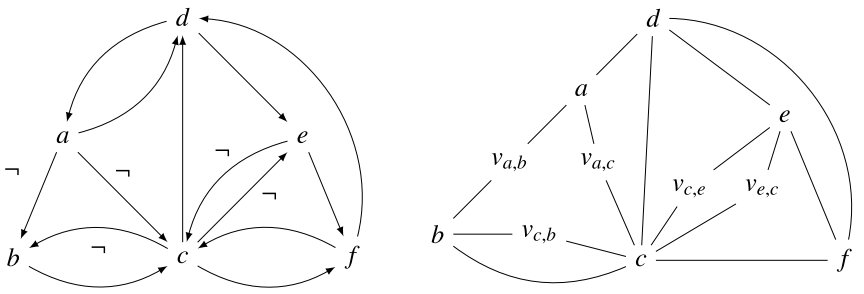


**Fig. 3.** Dependency digraph $D_P$ (left) and dependency graph $U_P$ (right) of the program $P$ of Example 2.1.

**Example 5.1.** Fig. 3 visualizes the dependency digraph $D_P$ and the dependency graph $U_p$ of the program $P$ of Example 2.1.

**Definition 5.3.** 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 5.4.** 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 5.2.** Consider the dependency graphs of the program $P$ of Example 2.1 as depicted in Fig. 3. 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 $P_{c=0} = P_0 = \{d \leftarrow a, e;\ a \leftarrow d, \neg b;\ e \leftarrow f;\ f\}$ and $P_1 = \{d \leftarrow a, e;\ f \leftarrow d;\ b;\ f\}$ are in the target class **no-DBEC**. $X$ is also a strong **no-BEC**-backdoor, since $P_0 \in$ **no-BEC** and $P_1 \in$ **no-BEC**. The answer sets of $P_\tau$ are $\mathrm{AS}(P_{\bar{c}}) = \{\{e, f\}\}$ 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 $\mathrm{AS}(P) = \{\{b, c, f\}\}$.

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

**Observation 5.5.** $\mathcal{C} \subseteq$ **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 5.6.** *All* $\mathcal{C} \in \mathcal{A}cyc$ *are hereditary.*

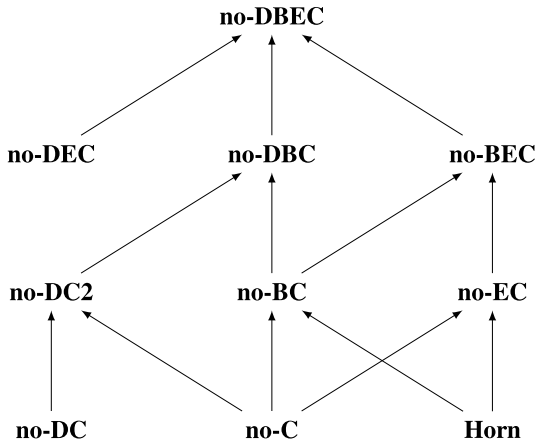


**Fig. 4.** Relationship between classes of programs with respect to their generality. A directed path from a class $\mathcal{C}$ to a class $\mathcal{C}'$ indicates that $\mathcal{C} \subseteq \mathcal{C}'$. If there is no path between two classes $\mathcal{C}$ and $\mathcal{C}'$, then neither $\mathcal{C} \subseteq \mathcal{C}'$ nor $\mathcal{C}' \subseteq \mathcal{C}$ and we say $\mathcal{C}$ and $\mathcal{C}'$ are *incomparable*.

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

**Observation 5.7.** *If $\mathcal{C}, \mathcal{C}' \in \mathcal{A}cyc \cup \{\textbf{Horn}\}$ such that the digraph in Fig. 4 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 [2,77,19]. 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 5.8.** *(See Apt et al. [2].)* **Strat** = **no-DBC**.

The class **no-DBEC**, the largest class in $\mathcal{A}cyc$, has already been studied by Zhao and Lin [156,110], 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 [121].

In our context this has the following important consequence.

**Proposition 5.9.** *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 5.5 and Proposition 5.9, all classes in $\mathcal{A}cyc$ satisfy the requirement of Theorem 3.9 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 $\mathcal{C} \in \mathcal{A}cyc$. 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.

### 5.1. Strong backdoor detection

**Proposition 5.10.** *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 2.3) to Strong $\mathcal{C}$-Backdoor Detection. Let $(\mathsf{S}, k)$ 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 $\mathsf{S}$ 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 $\mathsf{S}$ 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 $P_\tau$. Hence the only rules left in $P_\tau$ 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 $\mathsf{S}$. 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 $\mathsf{S}$. 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 $\mathsf{S}$. Consequently, the W[2]-hardness of Strong $\mathcal{C}$-Backdoor Detection follows. □

For the target classes in $\mathcal{D}$-$\mathcal{A}cyc$ we can avoid the use of tautological rules in the reduction and so strengthen Proposition 5.10 as follows (it would be interesting to know if this is also possible for the remaining classes mentioned in Proposition 5.10).

**Theorem 5.11.** *For every target class $\mathcal{C} \in \mathcal{D}$-$\mathcal{A}cyc$, 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 $\mathcal{C} \in \mathcal{D}$-$\mathcal{A}cyc$ when we forbid tautological rules in the input, we modify the reduction used in the proof of Proposition 5.10 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 5.10 we can show that $\mathsf{S}$ 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-paraNP-hard (see Section 2.3).

**Theorem 5.12.** *The problem* Strong **no-DBEC**-Backdoor Detection *is* co-paraNP*-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 [52,104],

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 $P \notin \textbf{no-DBEC}$. 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 $P \notin \textbf{no-DBEC}$. Conversely, assume $P \notin \textbf{no-DBEC}$. 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 $m$. □

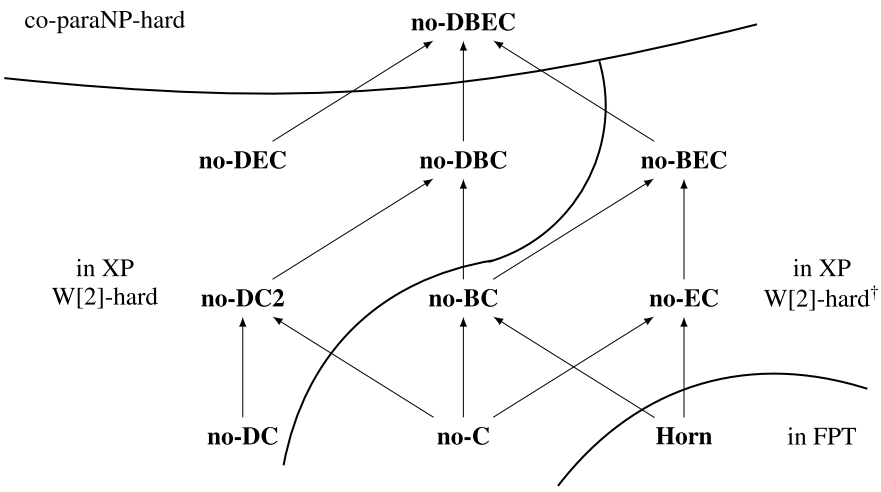


**Fig. 5.** Known complexity of the problem Strong $\mathcal{C}$-Backdoor Detection. † When we allow tautologies in the input program, see Theorem 5.11.

Fig. 5 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 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-paraNP-hardness also implies co-NP-hardness. Moreover, the reduction presented in the proof of Proposition 5.10 and Theorems 5.11, 5.12 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.

### 5.2. Deletion backdoor detection

The W[2]-hardness results of Theorems 5.11 and 5.12 suggest to relax the problem and to look for *deletion backdoors* instead of strong backdoors. In view of Lemma 3.4 and Observation 5.6, every deletion backdoor is also a strong backdoor for the considered acyclicity-based target classes, hence the backdoor approach of Theorem 3.9 works.

Fortunately, the results of this section show that the relaxation indeed gives us fixed-parameter tractability of backdoor detection for most considered classes. Fig. 6 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 5.13.** *The problem* Deletion $\mathcal{C}$-Backdoor Detection *is fixed-parameter tractable for all* $\mathcal{C} \in \mathcal{A}cyc \setminus \{$**no**-**DEC**, **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 [34,35] 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. [26] and Kawarabayashi and Kobayashi [102] showed that finding a $W$-feedback vertex set is fixed-parameter tractable, hence so is Deletion **no**-**BC**-Backdoor Detection.

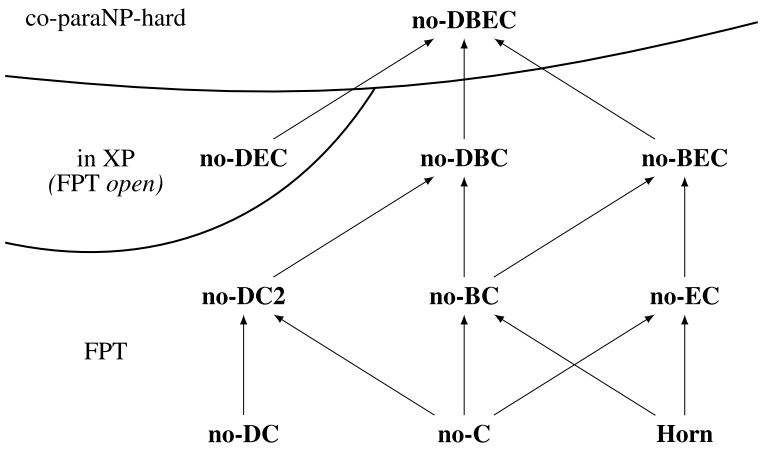


**Fig. 6.** Relationship between classes of programs and known complexity of the problem Deletion $\mathcal{C}$-Backdoor Detection. An arrow from $\mathcal{C}$ to $\mathcal{C}'$ indicates that deletion $\mathcal{C}$-backdoors are smaller than deletion $\mathcal{C}'$-backdoors. The FPT-results are established in Theorems 4.4 and 5.13. The XP-result is established in Theorem 5.15. The co-paraNP-hardness result is established in Theorem 5.16.

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. [116] for $W = V$, and by Kakimura et al. [98] for general $W$. In order to do this, we use the following trick of Aracena, Gajardo, and Montalva [117] 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 a 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. [21] 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 [12] 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. [26] 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. [23]. □

According to Observation 5.6, the classes mentioned in Theorem 5.13 are hereditary. Hence using Theorem 5.13 we can drop the assumption in Theorem 3.9 that the backdoor is given. We obtain directly the following results:

**Theorem 5.14.** *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 5.13. 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 5.15 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 5.16).

**Theorem 5.15.** *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 5.13 (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 [151] 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 [133] have shown that a Pfaffian orientation can be found in polynomial time, hence the test works in polynomial time. □

**Theorem 5.16.** *The problem* DELETION **no-DBEC**-BACKDOOR DETECTION *is* co-paraNP*-hard, and hence not fixed-parameter tractable unless* P = co-NP

**Proof.** The theorem follows from the reduction in the proof of Theorem 5.12. □

**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 5.13 and 5.15 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.

### 5.3. Backdoors of benchmark instances

For the acyclicity based target classes $\mathcal{C} \in \mathcal{A}cyc$ we have computed small deletion $\mathcal{C}$-backdoors only for very few selected instances with moderate size since the currently available algorithms can only deal with rather small instances within a reasonable computation time. The size of small deletion **no-C**-backdoors of selected instances of `Solitaire` was about half of the size of small strong **Horn**-backdoors.

## 6. Kernelization

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., [45,65,68]). 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. 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 [36]. 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 [134]. In the following 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.

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) 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 [22]. $L$ is called *compressible* if it admits a polynomial bi-kernel.

The following proposition states the connection between fixed-parameter tractable problems and kernels, as observed by Downey, Fellows, and Stege [36]:

**Proposition 6.1.** *(See Downey et al. [36], Flum and Grohe [50].) 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 3.9, 4.4, and 5.13 immediately provide that the mentioned problems admit a kernelization. In the following we investigate whether these problems admit polynomial kernels.

### 6.1. Backdoor detection

The first result of this section is quite positive.

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

**Proof.** First consider the case $\mathcal{C} =$ **Horn**. Let $(P,k)$ be an instance of Deletion **Horn**-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. We use the kernelization algorithm of Chen et al. [22] 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 $x \leftarrow \neg y$. Now $(P',k')$ is a polynomial kernel with a linear number of atoms.

Second consider the case $\mathcal{C} =$ **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 5.2). We use the kernelization algorithm of Thomassé [146] 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 5.13 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*.

### 6.2. 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 for 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 [51] 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.** *(See Fortnow and Santhanam [51].) If* Sat[Vars] *is compressible, then* NP $\subseteq$ co-NP/poly.

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 Turing machine with an additional polynomial-bounded input that depends only on the length of the input, but not on the input itself (advice function), for details see e.g., [100]. It is well-known from Yap's Theorem [154] that if NP $\subseteq$ 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 [58]. We need a different line of argument, as the technique used in [58] only applies to problems in NP or co-NP.

**Theorem 6.4.** *Let $\mathcal{C} \in \mathcal{A}cyc \cup \{\textbf{Horn}\}$. Then no problem in $\mathcal{A}sp\mathcal{R}eason$ admits a polynomial kernel when parameterized by the size of a smallest strong $\mathcal{C}$-backdoor or deletion $\mathcal{C}$-backdoor, unless* NP $\subseteq$ 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 [122] 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 $\mathcal{C} \in \mathcal{A}cyc$ ($\mathcal{C} = \textbf{Horn}$). 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. □

## 7. Lifting parameters

In this section we will 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 5.

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

**Definition 7.1.** 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 7.2.** 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 $P$.

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

**Definition 7.3.** 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 $\mathcal{C} \subseteq \textbf{Normal}$. 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 7.4** *(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 7.1.** Consider the program $P$ of Example 2.1 and let $\#\mathrm{neg}(P)$ denote the number of atoms that appear in negative rule bodies of a normal program (we will discuss this parameter in more detail in Section 8.2).

We determine $\#\mathrm{neg}^{\uparrow}(P) = 2$ 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 $\#\mathrm{neg}^{\uparrow}(P, X_1) = 2$ since $\#\mathrm{neg}(P - X_1) = 1$. We have $\#\mathrm{neg}^{\uparrow}(P, X_2) = 3$ since $\#\mathrm{neg}(P - X_2) = 2$. Thus, $\#\mathrm{neg}^{\uparrow}(P) = 2$.

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 $L[p]_{\mathrm{N}}$ to denote the restriction of $L[p]$ where instances are restricted to normal programs $P$. Similarly, BOUND$[p]_{\mathrm{N}}$ is the restriction of BOUND$[p]$ to normal programs. For all the problems $L[p]_{\mathrm{N}}$, $p$ only needs to be a normal ASP-parameter.

Next we state the main result of this section.

**Theorem 7.5** *(Lifting). Let $p$ be a normal ASP-parameter such that* BOUND$[p]_{\mathrm{N}}$ *and* ENUM$[p]_{\mathrm{N}}$ *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 7.6.** 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 appear together in the head of a rule of $P$.

**Lemma 7.7.** *Let $P$ be a disjunctive program. A set $X \subseteq \mathrm{at}(P)$ 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 7.8.** *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 7.7 and 7.8 we immediately obtain the next result.

**Proposition 7.9.** *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 7.5.** 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 7.9 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 $P_\tau$ is normal since $P - X$ is normal, and clearly $p(P_\tau) \leq p(P - X) \leq k$ by Definition 7.1. 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 $P_\tau$, we can indeed compute the set $\mathrm{AS}(P, X)$ in fpt-time.

By Lemma 3.6 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 3.4 that $X$ is a strong **Normal**-backdoor of $P$. Hence Lemma 3.7 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 7.2.** Consider the program $P$ of Example 2.1 with the deletion **Normal**-backdoor $X_1 = \{c\}$ from Example 7.1. We want to enumerate all answer sets of $P$. We obtain with Ben-Eliyahu's algorithm [4] 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 3.7) we observe that $\{b, c, f\}$ is the only answer set of $P$.

**Remark.** Please note that Definitions 4.2 and 5.1 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.

## 8. Theoretical comparison of ASP-parameters

In this section we compare several ASP-parameters in terms of their *generality*. Fig. 7 illustrates the results in terms of a lattice. 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 7.3 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

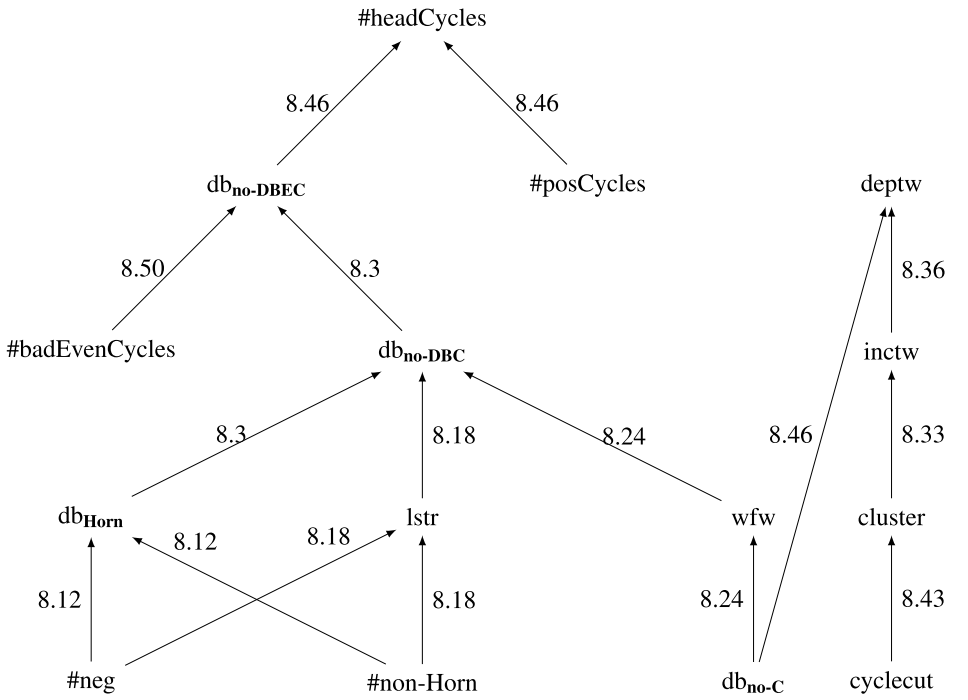


**Fig. 7.** Domination Lattice (relationship between ASP-parameters when restricted to normal programs). An arrow from $p$ to $p'$ indicates that $p'$ strictly dominates $p$. Please recall that deptw does not yield tractability (Proposition 8.36). When we do not restrict the parameters to normal programs and apply lifting (Observation 8.60) the parameter #headCycles is not strictly more general. A label $i$ of an edge indicates that Proposition $i$ establishes the result.

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 7.3. □

Let $\mathcal{C}$ be a class of programs. In the following, we omit $\cdot^{\downarrow}$ (see Definition 7.1) 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:

$$P_1^n := \{a \leftarrow \neg b_1, \ldots, \neg b_n\},$$
$$P_2^n := \{a_i \leftarrow \neg b : 1 \leq i \leq n\},$$
$$P_{31}^n := \{b_i \leftarrow \neg a;\ a \leftarrow \neg b_i : 1 \leq i \leq n\},$$
$$P_{32}^n := \{b_i \leftarrow a;\ a \leftarrow b_i : 1 \leq i \leq n\},$$
$$P_{33}^n := P_{31}^n \cup \{a \leftarrow d_1;\ d_i \leftarrow d_{i+1} : 1 \leq i < n\} \cup \{c_i \leftarrow b_i;\ d_i \leftarrow c_i;\ d_i \leftarrow b_i : 1 \leq i \leq n\},$$
$$P_{34}^n := P_{33}^n \cup \{d_i \leftarrow \neg b_i : 1 \leq i \leq n\},$$
$$P_{35}^n := P_{33}^n \setminus \{a \leftarrow \neg b_i;\ b_i \leftarrow \neg a : 1 \leq i \leq n\} \cup \{a_0 \leftarrow \neg a\} \cup \{b_i \leftarrow a_0 : 1 \leq i \leq n\},$$
$$P_4^n := \{c_i \leftarrow \neg a_i;\ c_i \leftarrow b_i;\ b_i \leftarrow \neg a_i;\ a_i \leftarrow e_i;\ e_i \leftarrow d_i;\ d_i \leftarrow a_i : 1 \leq i \leq n\},$$
$$P_{51}^n := \{b_i \leftarrow \neg a_i;\ a_i \leftarrow \neg b_i : 1 \leq i \leq n\},$$
$$P_{52}^n := \{b_i \leftarrow a_i;\ a_i \leftarrow \neg b_i : 1 \leq i \leq n\},$$
$$P_{53}^n := \{b_i \leftarrow a_i;\ a_i \leftarrow b_i : 1 \leq i \leq n\},$$
$$P_{54}^n := \{b_i \leftarrow a_i;\ c_i \leftarrow b_i;\ a_i \leftarrow c_i : 1 \leq i \leq n\},$$
$$P_6^n := \{a \leftarrow b_1, \ldots, b_n, c_i : 1 \leq i \leq n\},$$
$$P_7^n := \{a_j \leftarrow a_i : 1 \leq i < j \leq n\},$$

$$P_8^{m,n} := \{b \leftarrow a_1, \ldots, a_m\} \cup \{c_i \leftarrow c_{i+1} : 1 \leq i \leq n\} \cup \{c_{n+1} \leftarrow c_1\},$$
$$P_9^n := \{a_2 \leftarrow \neg a_1;\ a_3 \leftarrow \neg a_2\} \cup \{b_i \leftarrow a_3;\ a_1 \leftarrow b_i : 1 \leq i \leq n\}, \quad \text{and}$$
$$P_{11}^n := \{a_i \vee b \leftarrow c;\ c \leftarrow b;\ b \leftarrow a_i : 1 \leq i \leq n\}.$$

### 8.1. ASP-parameters based on backdoor size

Backdoor-based ASP-parameters can be related to each other in terms of their underlying base 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 $|\mathrm{at}(P)| \geq 1$. 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 5.7 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 3.4 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.** $\mathrm{sb}_{\mathbf{Horn}} \sim \mathrm{db}_{\mathbf{Horn}}$.

**Observation 8.6.** *We make the following observations about programs from Example 8.1.*

1. *Consider program $P_{31}^n$ and $P_{32}^n$ and let $P \in \{P_{31}^n, P_{32}^n\}$. 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 $\mathrm{db}_{\mathcal{C}}(P_{31}^n) \leq 1$ and $\mathrm{db}_{\mathcal{C}}(P_{32}^n) \leq 1$ where $\mathcal{C} \in \{\mathbf{Horn}\} \cup \mathcal{A}cyc$.*
2. *Consider program $P_{33}^n$. Since $P_{33}^n - \{a\}$ is Horn and contains no directed cycle and no bad cycle, we obtain $\mathrm{db}_{\mathbf{Horn}}(P_{33}^n) = 0$, $\mathrm{db}_{\mathbf{no\text{-}DC}}(P_{33}^n) \leq 1$, and $\mathrm{db}_{\mathbf{no\text{-}BC}}(P_{33}^n) \leq 1$. According to Observation 8.3, we have $\mathrm{db}_{\mathcal{C}}(P_{33}^n) \leq 1$ where $\mathcal{C} \in \{\mathbf{Horn}, \mathbf{no\text{-}BC}, \mathbf{no\text{-}BEC}\} \cup \mathcal{D}\text{-}\mathcal{A}cyc$.*
3. *Consider program $P_{34}^n$. Since $P_{34}^n - \{a\}$ contains no even cycle, $\mathrm{db}_{\mathbf{no\text{-}EC}}(P_{34}^n) \leq 1$.*
4. *Consider program $P_4^n$. The negation dependency graph of $P_4^n$ contains $2n$ disjoint paths $a_i b_i$ and $a_i c_i$. Thus smallest deletion **Horn**-backdoor are of size at least $n$. $P_4^n$ 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 $\mathrm{db}_{\mathcal{C}}(P_4^n) \geq n$ where $\mathcal{C} \in \{\mathbf{Horn}, \mathbf{no\text{-}C}, \mathbf{no\text{-}BC}, \mathbf{no\text{-}DC}, \mathbf{no\text{-}DC2}, \mathbf{no\text{-}EC}, \mathbf{no\text{-}BEC}, \mathbf{no\text{-}DEC}\}$.*
5. *Consider program $P_{51}^n$. The negation dependency graph of $P_{51}^n$ contains $n$ disjoint paths and thus $\mathrm{db}_{\mathbf{Horn}}(P_{51}^n) = n$. $P_{51}^n$ contains $n$ disjoint directed bad even cycles and thus $\mathrm{db}_{\mathbf{no\text{-}DBEC}}(P_{51}^n) = n$. According to Observation 8.3, we obtain $\mathrm{db}_{\mathcal{C}}(P_{51}^n) \geq n$ where $\mathcal{C} \in \{\mathbf{Horn}\} \cup \mathcal{A}cyc$.*
6. *Consider program $P_{52}^n$. Since $P_{52}^n$ contains $n$ disjoint directed bad cycles, $\mathrm{db}_{\mathbf{no\text{-}DBC}}(P_{52}^n) = n$.*
7. *Consider program $P_{54}^n$. Since $P_{54}^n$ 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 $\mathrm{db}_{\mathcal{C}}(P_{54}^n) \geq n$ 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 $P_6^n$. Since $P_6^n$ is Horn and contains no cycle and no directed cycle, $\mathrm{db}_{\mathbf{Horn}}(P_6^n) = \mathrm{db}_{\mathbf{no\text{-}C}}(P_6^n) = \mathrm{db}_{\mathbf{no\text{-}DC}}(P_6^n) = 0$. According to Observation 8.3, we have $\mathrm{db}_{\mathcal{C}}(P_6^n) = 0$ where $\mathcal{C} \in \{\mathbf{Horn}\} \cup \mathcal{A}cyc$.*
9. *Consider program $P_7^n$. Since $P_7^n$ is Horn and contains no bad cycle and no directed cycle, $\mathrm{db}_{\mathbf{Horn}}(P_7^n) = \mathrm{db}_{\mathbf{no\text{-}BC}}(P_7^n) = \mathrm{db}_{\mathbf{no\text{-}DC}}(P_7^n) = 0$. According to Observation 8.3, we have $\mathrm{db}_{\mathcal{C}}(P_7^n) = 0$ where $\mathcal{C} \in \{\mathbf{Horn}, \mathbf{no\text{-}BC}, \mathbf{no\text{-}BEC}\} \cup \mathcal{D}\text{-}\mathcal{A}cyc$.*
10. *Consider program $P_8^{m,n}$. Since $P_8^{m,n}$ is Horn and $P_8^{m,n} - \{c_1\}$ contains no cycle and no directed cycle, we obtain $\mathrm{db}_{\mathbf{Horn}}(P_8^{m,n}) = 0$, $\mathrm{db}_{\mathbf{no\text{-}C}}(P_8^{m,n}) \leq 1$, $\mathrm{db}_{\mathbf{no\text{-}DC}}(P_8^{m,n}) \leq 1$. According to Observation 8.3, we have $\mathrm{db}_{\mathcal{C}}(P_8^{m,n}) \leq 1$ where $\mathcal{C} \in \{\mathbf{Horn}\} \cup \mathcal{A}cyc$.*
11. *Consider program $P_9^n$. Since $P_9^n - \{a_2\}$ is Horn and $P_9^n - \{a_1\}$ contains no cycle and no directed cycle, we have $\mathrm{db}_{\mathbf{Horn}}(P_9^n) \leq 1$, $\mathrm{db}_{\mathbf{no\text{-}C}}(P_9^n) \leq 1$, and $\mathrm{db}_{\mathbf{no\text{-}DC}}(P_9^n) \leq 1$. According to Observation 8.3, we have $\mathrm{db}_{\mathcal{C}}(P_9^n) \leq 1$ where $\mathcal{C} \in \{\mathbf{Horn}\} \cup \mathcal{A}cyc$.*

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* $\mathcal{C} \in \mathcal{A}cyc$. *Consequently,* $\mathrm{db}_{\mathcal{C}}(P^n_{11} - X) = 1$ *where* $\mathcal{C} \in \{\mathbf{Horn}, \mathbf{Normal}\} \cup \mathcal{A}cyc$.

### 8.2. 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.

**Definition 8.7.** (See Ben-Eliyahu [4].) Let $P$ be a normal program. Then

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

**Proposition 8.8.** *(See Ben-Eliyahu [4].) For each* $L \in \mathcal{A}sp\mathcal{F}ull$, $L[\#\mathrm{neg}]_{\mathrm{N}} \in \mathrm{FPT}$ *and* $L[\#\text{non-Horn}]_{\mathrm{N}} \in \mathrm{FPT}$.

Since $\textsc{Bound}[p]_{\mathrm{N}}$ for $p \in \{\#\mathrm{neg}, \#\text{non-Horn}\}$ is clearly solvable in polynomial time and thus fixed-parameter tractable, we can use the Lifting Theorem (Theorem 7.5) to obtain the following result.

**Corollary 8.9.** *For each* $L \in \mathcal{A}sp\mathcal{F}ull$, $L[\#\mathrm{neg}^{\uparrow}] \in \mathrm{FPT}$ *and* $L[\#\text{non-Horn}^{\uparrow}] \in \mathrm{FPT}$.

**Observation 8.10.** *We make the following observations about programs from Example 8.1.*

1. *Consider program* $P^n_1$ *which contains* $n$ *atoms that occur in* $B^-(r)$ *for some rule* $r \in P$ *and exactly one non-Horn rule. Thus,* $\#\mathrm{neg}(P^n_1) = n$ *and* $\#\text{non-Horn}(P^n_1) = 1$.
2. *Consider program* $P^n_2$ *which contains only the atom* $b$ *that occurs in* $B^-(r)$ *for some rule* $r \in P^n_2$ *and* $n$ *non-Horn rules. Thus,* $\#\mathrm{neg}(P^n_2) = 1$ *and* $\#\text{non-Horn}(P^n_2) = n$.
3. *Consider program* $P^n_{31}$ *which contains for* $1 \le i \le n$ *the atoms* $a$ *and* $b_i$ *that occur in* $B^-(r)$ *for some rule* $r \in P^n_{31}$ *and the non-Horn rules* $b_i \leftarrow \neg a$ *and* $a \leftarrow \neg b_i$. *Hence,* $\#\mathrm{neg}(P^n_{31}) = n + 1$ *and* $\#\text{non-Horn}(P^n_{31}) = 2n$.
4. *Consider program* $P^n_{32}$ *which is Horn. Thus,* $\#\mathrm{neg}(P^n_{32}) = \#\text{non-Horn}(P^n_{32}) = 0$.
5. *Consider program* $P^n_{35}$ *which contains only the atom* $a$ *that occurs in* $B^-(r)$ *for some rule* $r \in P^n_{35}$ *and exactly one non-Horn rule. So* $\#\mathrm{neg}(P^n_{35}) = \#\text{non-Horn}(P^n_{35}) = 1$.
6. *Consider program* $P^n_4$ *which contains for* $1 \le i \le n$ *the atoms* $a_i$ *that occur in* $B^-(r)$ *for some rule* $r \in P^n_4$ *and the non-Horn rules* $b_i \leftarrow \neg a_i$ *and* $c_i \leftarrow \neg a_i$. *Thus,* $\#\mathrm{neg}(P^n_4) = n$ *and* $\#\text{non-Horn}(P^n_4) = 2n$.
7. *Consider program* $P^n_{51}$ *which contains for* $1 \le i \le n$ *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,* $\#\mathrm{neg}(P^n_{51}) = \#\text{non-Horn}(P^n_{51}) = 2n$.
8. *Consider the program* $P^n_{52}$ *which contains the atoms* $b_i$ *that occur in* $B^-(r)$ *for some rule* $r \in P^n_{52}$ *and the non-Horn rules* $a_i \leftarrow \neg b_i$. *Hence,* $\#\mathrm{neg}(P^n_{52}) = \#\text{non-Horn}(P^n_{52}) = n$.
9. *Consider programs* $P^n_{54}$, $P^n_6$, $P^n_7$, *and* $P^{m,n}_8$ *which are Horn. Thus,* $\#\mathrm{neg}(P^n_{54}) = \#\text{non-Horn}(P^n_{54}) = \#\mathrm{neg}(P^n_6) = \#\text{non-Horn}(P^n_6) = \#\mathrm{neg}(P^n_7) = \#\text{non-Horn}(P^n_7) = \#\mathrm{neg}(P^{m,n}_8) = \#\text{non-Horn}(P^{m,n}_8) = 0$.
10. *Consider the program* $P^n_9$ *which contains only the atoms* $a_1$ *and* $a_2$ *that occur in* $B^-(r)$ *for some rule* $r \in P^n_9$ *and only the non-Horn rules* $a_2 \leftarrow \neg a_1$ *and* $a_3 \leftarrow \neg a_2$. *Hence,* $\#\mathrm{neg}(P^n_9) = \#\text{non-Horn}(P^n_9) = 2$.
11. *Consider the program* $P^n_{11}$. *The set* $X = \{b\}$ *is a deletion* **Normal**-*backdoor of* $P^n_{11}$. *Since* $P^n_{11} - X$ *is Horn, we have* $\#\mathrm{neg}(P^n_{11} - X) = \#\text{non-Horn}(P^n_{11} - X) = 0$. *Thus,* $\#\mathrm{neg}^{\uparrow}(P^n_{11}) = |X| + \#\mathrm{neg}(P^n_{11} - X) = 1$ *and* $\#\text{non-Horn}^{\uparrow}(P^n_{11}) = |X| + \#\text{non-Horn}(P^n_{11} - X) = 1$.

**Proposition 8.11.** #neg *and* #non-Horn *are incomparable.*

**Proof.** The proposition directly follows from considering $P^n_1$ and $P^n_2$ where $\#\mathrm{neg}(P^n_1) = n$ and $\#\text{non-Horn}(P^n_1) = 1$; and $\#\mathrm{neg}(P^n_2) = 1$ and $\#\text{non-Horn}(P^n_2) = n$ by Observation 8.10. □

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

**Proposition 8.12.** $\mathrm{db}_{\mathbf{Horn}}$ *strictly dominates* #neg *and* #non-Horn. $\mathrm{db}_{\mathcal{C}}$ *and* #neg*; and* $\mathrm{db}_{\mathcal{C}}$ *and* #non-Horn *are incomparable where* $\mathcal{C} \in \{\mathbf{no\text{-}C}, \mathbf{no\text{-}DC}, \mathbf{no\text{-}DC2}, \mathbf{no\text{-}EC}, \mathbf{no\text{-}DEC}\}$.

**Proof.** For a normal program $P$ define the sets $B^-(P) = \{a \in \mathrm{at}(P) : a \in B^-(r)$ for some rule $r \in P\}$ and $H(P) = \{a \in H(r) : r \in P, r$ is not Horn$\}$. We observe that $B^-(P)$ and $H(P)$ are deletion **Horn**-backdoors of $P$, hence $\mathrm{db}_{\mathbf{Horn}}(P) \leq \#\mathrm{neg}(P)$ and $\mathrm{db}_{\mathbf{Horn}}(P) \leq \#\text{non-Horn}(P)$. To show that $\mathrm{db}_{\mathbf{Horn}}$ strictly dominates the two parameters, consider $P^n_{31}$ where $\mathrm{db}_{\mathbf{Horn}}(P^n_{31}) \leq 1$, but $\#\mathrm{neg}(P^n_{31}) = n+1$ and $\#\text{non-Horn}(P^n_{31}) = 2n$ by Observations 8.6 and 8.10.

The second statement follows from considering the programs $P^n_{31}$ and $P^n_{54}$ where $\mathrm{db}_{\mathcal{C}}(P^n_{31}) \leq 1$ and $p(P^n_{31}) \geq n+1$; and $\mathrm{db}_{\mathcal{C}}(P^n_{54}) \geq n$ and $p(P^n_{54}) = 0$ for $\mathcal{C} \in \{$**no-C**, **no-DC**, **no-DC2**, **no-EC**, **no-DEC**$\}$ and $p \in \{\#\mathrm{neg}, \#\text{non-Horn}\}$ by Observations 8.6 and 8.10. Hence $\mathrm{db}_{\mathcal{C}} \bowtie \#\mathrm{neg}$ and $\mathrm{db}_{\mathcal{C}} \bowtie \#\text{non-Horn}$ for $\mathcal{C} \in \{$**no-C**, **no-DC**, **no-DC2**, **no-EC**, **no-DEC**$\}$. □

### 8.3. ASP-parameters based on the distance from being stratified

Ben-Eliyahu [4] and Gottlob et al. [85] 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 2.4 that $\mathrm{SCC}(G)$ denotes the partition of the vertex set of a digraph into strongly connected components.

**Definition 8.13.** (See Ben-Eliyahu [4].) Let $P$ be a normal program, $D_P$ its dependency digraph, and $A \subseteq \mathrm{at}(P)$. $P_{/A}$ 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

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

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

**Proposition 8.14.** *(See Ben-Eliyahu [4].) For each $L \in \mathcal{AspFull}$, $L[\mathrm{lstr}]_{\mathrm{N}} \in$ 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 7.5) to obtain the following result.

**Corollary 8.15.** *For each $L \in \mathcal{AspFull}$, $L[\mathrm{lstr}^{\uparrow}] \in$ FPT.*

**Observation 8.16.** *We make the following observations about programs from Example 8.1.*

1. *Consider program $P^n_{31}$ and let $P = P^n_{31}$. The partition $\mathrm{SCC}(D_P)$ contains only the set $C = \mathrm{at}(P)$ and thus $P_{/C} = P$. By Observation 8.10 $\#\mathrm{neg}(P) = n+1$ and $\#\text{non-Horn}(P) = 2n$ and hence $\mathrm{lstr}(P^n_{31}) = n+1$.*
2. *Consider program $P^n_{32}$ and let $P = P^n_{32}$. The partition $\mathrm{SCC}(D_P)$ contains only the set $C = \mathrm{at}(P)$ and $P_{/C} = P$. Since $\#\mathrm{neg}(P) = 0$ by Observation 8.10, we have $\mathrm{lstr}(P^n_{32}) = 0$.*
3. *Consider program $P^n_{35}$ and let $P = P^n_{35}$. The partition $\mathrm{SCC}(D_P)$ contains only the set $C = \mathrm{at}(P)$. Thus, $P = P_{/C}$. Since $\#\mathrm{neg}(P^n_{35}) = 1$ by Observation 8.10, we conclude $\mathrm{lstr}(P^n_{35}) \leq 1$.*
4. *Consider program $P^n_4$ and let $P = P^n_4$. 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 $\mathrm{lstr}(P^n_4) = 0$.*
5. *Consider program $P^n_{51}$ and $P^n_{52}$ and let $P \in \{P^n_{51}, P^n_{52}\}$. 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 $\mathrm{lstr}(P^n_{51}) = 2n$ and $\mathrm{lstr}(P^n_{52}) = n$.*
6. *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 $P_{/A} = \{a \leftarrow b_1, \ldots, b_n, c_i : 1 \leq i \leq n\}$ 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 $\mathrm{lstr}(P^n_6) = 0$.*
7. *Consider program $P^n_7$ and let $P = P^n_7$. 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 $\mathrm{lstr}(P^n_7) = 0$.*
8. *Consider program $P^{m,n}_8$ and let $P = P^{m,n}_8$. 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 $\mathrm{lstr}(P^{m,n}_8) = 0$.*
9. *Consider program $P^n_9$ and let $P = P^n_9$. 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 $\mathrm{lstr}(P^n_9) = 2$.*

10. *Consider program* $P^n_{11}$. *The set* $X = \{b\}$ *is a deletion* **Normal**-*backdoor of* $P^n_{11}$ *by Observation 8.6. We have* $P = P^n_{11} - X = \{a_i \leftarrow c;\ c;\ \leftarrow a_i : 1 \leq i \leq n\}$. *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,* $\mathrm{lstr}^{\uparrow}(P^n_{11}) = |X| + \mathrm{lstr}(P^n_{11} - X) = 1$.

**Observation 8.17.** 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 $B^-(P) = \{a \in \mathrm{at}(P) : a \in B^-(r)$ 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 $\sum_{C \in \mathrm{SCC}(D_P)} \#\text{non-Horn}(P_{/C}) \leq \#\mathrm{non-Horn}(P)$. 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 $P^n_4$ where $\mathrm{lstr}(P^n_4) = 0$, but $\#\mathrm{neg}(P^n_4) \geq n$ and $\#\mathrm{non-Horn}(P^n_4) \geq 2n$ by Observations 8.10 and 8.16. Hence the observation is true. □

**Proposition 8.18.** $\mathrm{db}_{\textbf{no-DBC}}$ *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 $B^-(P) = \{a \in \mathrm{at}(P) : a \in B^-(r)$ for some rule $r \in P\}$ and $H(P) = \{a \in H(r) : r \in P, r$ 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 8.6 and 8.16. 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 $\mathcal{C} \in \{\textbf{Horn}, \textbf{no-C}, \textbf{no-BC}, \textbf{no-DC}, \textbf{no-DC2}, \textbf{no-EC}, \textbf{no-BEC}, \textbf{no-DEC}\}$ by Observations 8.6 and 8.16. We conclude $\mathrm{db}_{\mathcal{C}} \bowtie \mathrm{lstr}$. □

**Definition 8.19.** (See Gottlob et al. [85].) 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

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

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

**Observation 8.20.** *Let* $P$ *be a normal program and* $D_P$ *its dependency digraph. Then* $\mathrm{fw(P)} = \mathrm{db}_{\textbf{no-C}}(P)$ *and hence*

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

**Proposition 8.21.** *(See Gottlob et al. [85].) For each* $L \in \mathcal{A}sp\mathcal{F}ull$, $L[\mathrm{fw}]_{\mathrm{N}} \in$ FPT *and* $L[\mathrm{wfw}]_{\mathrm{N}} \in$ FPT.

Since $\textsc{Bound}[\mathrm{fw}]_{\mathrm{N}}$ and $\textsc{Bound}[\mathrm{wfw}]_{\mathrm{N}}$ is fixed-parameter tractable, we can use the Lifting Theorem (Theorem 7.5) to obtain the following result.

**Corollary 8.22.** *For each* $L \in \mathcal{A}sp\mathcal{F}ull$, $L[\mathrm{fw}^{\uparrow}] \in$ FPT *and* $L[\mathrm{wfw}^{\uparrow}] \in$ FPT.

**Observation 8.23.** *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**, $\{r \in \hat{P}_{/C} - \mathrm{at}^+(\hat{P}_{/C}), C \in \mathrm{SCC}(D_P), \hat{P}_{/C} \notin \textbf{no-DBC}\} = P$. *We have* $\mathrm{db}_{\textbf{no-C}}(P) = 1$ *by Observation 8.6 and according to Observation 8.20, 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* $\{r \in \hat{P}_{/C} - \mathrm{at}^+(\hat{P}_{/C}) : C \in \mathrm{SCC}(D_P), \hat{P}_{/C} \notin \textbf{no-DBC}\} = \emptyset$. *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* $\{r \in \hat{P}_{/C} - \mathrm{at}^+(\hat{P}_{/C}), C \in \mathrm{SCC}(D_P), \hat{P}_{/C} \notin \textbf{no-DBC}\} = P$. *We have* $\mathrm{db}_{\textbf{no-C}}(P) = n$ *since* $P$ *contains* $n$ *disjoint* $\{b_i\}$*-cycles. According to Observation 8.20, 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,* $\{r \in \hat{P}_{/C} - \mathrm{at}^+(\hat{P}_{/C}), C \in \mathrm{SCC}(D_P), \hat{P}_{/C} \notin \textbf{no-DBC}\} = \emptyset$ *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**, $\{r \in \hat{P}_{/C} - \mathrm{at}^+(\hat{P}_{/C}) : C \in \mathrm{SCC}(D_P), \hat{P}_{/C} \notin \textbf{no-DBC}\} = P$. *We observe that* $\mathrm{db}_{\textbf{no-C}}(P) = n$ *and according to Observation 8.20, 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* $P^{m,n}_8$ *and let* $P = P^{m,n}_8$. *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* $\mathrm{wfw}(P^{m,n}_8) = 0$.
10. *Consider program* $P^n_9$ *and let* $P = P^n_9$. *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 8.6* $\mathrm{db}_{\textbf{no-C}}(P) \leq 1$ *and according to Observation 8.20, we obtain* $\mathrm{wfw}(P^n_9) \leq 1$.
11. *Consider program* $P^n_{11}$ *and let* $P = P^n_{11}$. *The set* $X = \{b\}$ *is a deletion* **Normal***-backdoor of* $P^n_{11}$ *by Observation 8.6 and* $P = P^n_{11} - X = \{a_i \leftarrow c;\ c;\ \leftarrow a_i : 1 \leq i \leq n\}$. *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 8.20, we obtain* $\mathrm{wfw}(P) = 0$. *Consequently,* $\mathrm{wfw}^{\uparrow}(P^n_{11}) = |X| + \mathrm{wfw}(P^n_{11} - X) = 1$.

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 et al. [85].

**Proposition 8.24.** 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 5.6), $\hat{P} - X \in$ **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 $P^n_4$ where $\mathrm{wfw}(P^n_4) = 0$ and $\mathrm{db}_{\textbf{no-C}}(P^n_4) = n$. Hence $\mathrm{wfw} \prec \mathrm{db}_{\textbf{no-C}}$ by Observations 8.3 and 8.23.

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 8.20, $\mathrm{wfw}(P) = \mathrm{db}_{\textbf{no-C}}(\hat{P})$ 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

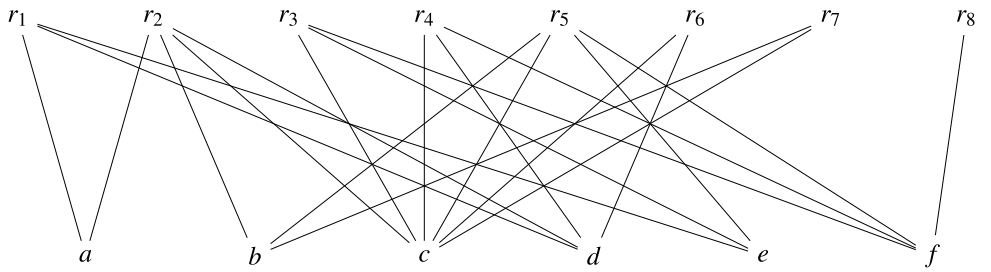


**Fig. 8.** Incidence graph $I_P$ of the program $P$ of Example 2.1.

**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 $P$. 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 8.6 and 8.23. 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 $\mathcal{C} \in \{\textbf{Horn}, \textbf{no-BC}, \textbf{no-DC}, \textbf{no-DC2}, \textbf{no-BEC}, \textbf{no-DEC}\}$ 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 8.6 and 8.23. Hence $\mathrm{db}_{\mathcal{C}} \bowtie \mathrm{wfw}$ for $\mathcal{C} \in \{\textbf{Horn}, \textbf{no-BC}, \textbf{no-DC}, \textbf{no-DC2}, \textbf{no-EC}, \textbf{no-BEC}, \textbf{no-DEC}\}$. □

**Observation 8.25.** *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 8.10, 8.16 and 8.23. □

### 8.4. Incidence treewidth

Treewidth is graph parameter introduced by Robertson and Seymour [130–132] that measures in a certain sense the tree-likeness of a graph. See [8–10,86] for further background and examples on treewidth. Treewidth has been widely applied in knowledge representation, reasoning, and artificial intelligence [39,86,90,118,126].

**Definition 8.26.** Let $G = (V, E)$ be a graph, $T = (N, E_T)$ a tree, and $\chi$ 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 $(T, \chi)$ 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 $(T, \chi)$ is $\max\{|\chi(t)| - 1 : t \in V(T)\}$. The *treewidth* of $G$, denoted by tw(G), is the minimum taken over the widths of all possible tree decompositions of $G$.

We will use the following basic properties of treewidth.

**Lemma 8.27.** *(Folklore, e.g., [132].) 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 8.28.** *(Folklore, e.g., [6].) 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 8.29.** (See Jakl et al. [90].) 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 $P$.

Fig. 8 illustrates the incidence graph $I_P$ of the program $P$ of Example 2.1.

**Proposition 8.30.** *(See Jakl et al. [90].) For each $L \in \mathcal{A}sp\mathcal{F}ull \setminus \{\textsc{Enum}\}$, $L[\mathrm{inctw}]_{\mathrm{N}} \in \mathrm{FPT}$ and for $\textsc{Enum}[\mathrm{inctw}]_{\mathrm{N}}$ the solutions can be enumerated with fixed-parameter linear delay between any two consecutive solutions.*

**Observation 8.31.** *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 8.28, a cycle has treewidth at most* 2 *and according to Lemma 8.27, 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* $P^{m,n}_8$. *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 8.28, a cycle has treewidth at most* 2, *and according to Lemma 8.27, we obtain* $\mathrm{inctw}(P^{m,n}_8) \leq 2$.

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

**Observation 8.32.** $\textsc{Enum}[\mathrm{inctw}]_{\mathrm{N}} \notin \mathrm{FPT}$.

**Proof.** Consider the program $P^n_{51}$ where $\mathrm{inctw}(P^n_{51}) \leq 2$. 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 8.33.** *If* $\mathcal{C} \in \{\textbf{Horn}\} \cup \mathcal{A}cyc$ *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 $P^n_{51}$ and $P^n_6$ where $p(P^n_{51}) \geq n$ and $\mathrm{inctw}(P^n_{51}) = 2$; and $p(P^n_6) \leq 1$ and $\mathrm{inctw}(P^n_6) \geq n-1$ by Observations 8.6, 8.10, 8.16, 8.23, and 8.31. □

### 8.5. 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 with respect to the target class **no-C**, but unfortunately parameterizing the main ASP problems by the dependency treewidth does not yield fixed-parameter tractability.

**Definition 8.34.** Let $P$ be a program, then $\mathrm{deptw}(P) = \mathrm{tw}(U_P)$. We call $\mathrm{deptw}(P)$ the *dependency treewidth* of $P$.

**Observation 8.35.** *We make the following observations about programs from Example 8.1.*

1. *Consider programs* $P^n_{32}$ *and* $P^n_6$ *where the dependency graph is a tree. Thus,* $\mathrm{deptw}(P^n_{32}) = \mathrm{deptw}(P^n_6) = 1$.
2. *Consider program* $P^n_{51}$. *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 8.28, a cycle has treewidth at most* 2 *and according to Lemma 8.27, we obtain* $\mathrm{deptw}(P^n_{51}) \leq 2$.
3. *Consider program* $P^n_7$. *Its dependency graph contains a clique on n vertices as a subgraph. Hence* $\mathrm{deptw}(P^n_7) \geq n-1$.

**Proposition 8.36.** deptw *strictly dominates* inctw *and* $\mathrm{db}_{\textbf{no-C}}$. *Let* $\mathcal{C} \in \{\textbf{Horn}\} \cup \mathcal{A}cyc \setminus \{\textbf{no-C}, \textbf{no-EC}\}$ *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 $(T, \chi)$ be an arbitrary tree decomposition of $I_P$. We create a tree decomposition $(T, \chi')$ 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 $(T, \chi')$ satisfies Conditions 1 and 2 of a tree decomposition of $U_P$. Since all edges of $I_P$ are covered in $(T, \chi)$ 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 $(T, \chi)$ and all corresponding elements $h$ are connected in $(T, \chi)$, Condition 3 holds for $(T, \chi')$. Thus, $(T, \chi')$ is a tree decomposition of the dependency graph $U_P$. Since the width of $(T, \chi')$ is less or equal to the width of $(T, \chi)$ 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 $P^n_6$ where $\mathrm{deptw}(P^n_6) \leq 1$ and $\mathrm{inctw}(P^n_6) \geq n$. 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 8.28, $\mathrm{tw}(U_P) \leq k+1$. Hence deptw $\preceq \mathrm{db}_{\textbf{no-C}}$. To show that deptw strictly dominates $\mathrm{db}_{\textbf{no-C}}$ consider the program $P^n_{51}$ where $\mathrm{deptw}(P^n_{51}) \leq 2$ and $\mathrm{db}_{\textbf{no-C}}(P^n_{51}) \geq n$. Consequently, deptw $\prec \mathrm{db}_{\textbf{no-C}}$ and the proposition sustains.

To show the last statement, consider again the programs $P^n_{51}$ and $P^n_7$ where $\mathrm{deptw}(P^n_{51}) \leq 2$ and $p(P^n_{51}) \geq n$; and $\mathrm{deptw}(P^n_7) \geq n-1$ and $p(P^n_7) = 0$ by Observations 8.6, 8.16, 8.23, and 8.35. □

**Proposition 8.37.** *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 $(T, \chi)$ 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 $\chi(t_{fx}) := \{a_x, a_{\bar{x}}, f\}$, $\chi(t_{x\bar{x}}) := \{a_x, a_{\bar{x}}, v_{a_x a_{\bar{x}}}\}$, and $\chi(t_{\bar{x}x}) := \{a_x, a_{\bar{x}}, v_{\bar{a}_x a_x}\}$. We observe that the pair $(T, \chi)$ 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 $(T, \chi)$. Hence $(T, \chi)$ is a tree decomposition of the dependency graph $U_P$. Since $\max\{|\chi(t)| - 1 : t \in V(T)\} = 2$, the tree decomposition $(T, \chi)$ is of width 2 and $\mathrm{deptw}(P) = 2$. Hence the problem Consistency[deptw]$_N$ 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. □

### 8.6. Interaction treewidth

In this section we consider two parameters investigated by Ben-Eliyahu and Dechter [3]: the interaction treewidth introduced under the term "clique width",[4] 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 8.38.** (See Ben-Eliyahu and Dechter [3].) 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 $H(r) \cap H(r') \neq \emptyset$.[5]

**Definition 8.39.** (See Kanchanasut and Stuckey [99], Ben-Eliyahu and Dechter [3].) 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 $y \in B^+(r)$.[6]

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.

**Definition 8.40.** (See Ben-Eliyahu and Dechter [3].) Let $G$ be a digraph and $G'$ a graph. Then

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

lc is the *length of the longest cycle*. cs is the *clique size*.[7]

Let $P$ be a normal program, $A_P$ its interaction graph, and $D_P^+$ its positive dependency digraph. Then

$$\mathrm{cluster}(P) := \mathrm{cs}(A_P) \cdot \log \mathrm{lc}\big(D_P^+\big)$$
$$\mathrm{cyclecut}(P) := \mathrm{fw}(A_P) \cdot \log \mathrm{lc}\big(D_P^+\big).$$

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

[4] Today the term "clique-width" is predominately used to refer to a different graph parameter, see, e.g., [83].

[5] This definition is equivalent to the original definition in [3] 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.

[6] Ben-Eliyahu and Dechter [3] used the term dependency graph while the term positive dependency graph was first used by Kanchanasut and Stuckey [99] and became popular by Erdem and Lifschitz [43].

[7] 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.

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

**Lemma 8.41.** *(See Robertson and Seymour [132].) Let $G$ be a graph. Then* $\mathrm{tw}(G) = \mathrm{cs}(G) + 1$.

**Corollary 8.42.** *Let $P$ be a normal program, $A_P$ its interaction graph, and $D_P^+$ its dependency digraph. Then*

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

**Proposition 8.43.** *(See Ben-Eliyahu and Dechter [3].) For each $L \in \mathcal{AspFull}$, $L[\mathrm{cluster}]_{\mathrm{N}} \in$ FPT and $L[\mathrm{cyclecut}]_{\mathrm{N}} \in$ FPT.*

**Observation 8.44.** *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 8.28, 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 8.41, we obtain $\mathrm{cs}(A_P) \geq m - 2$. According to Lemma 8.28, 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 $\mathrm{cyclecut}(P_8^{m,n}) \geq (m-2) \cdot \log n$.*

**Observation 8.45.** cluster *strictly dominates* cyclecut.

**Proof.** Let $P$ be a normal program and $A_P$ its interaction graph. According to Lemma 8.28, we obtain $\mathrm{tw}(A_P) \leq \mathrm{fw}(A_P) + 1$. Hence $\mathrm{cluster}(P) \prec \mathrm{cyclecut}(P)$. □

**Proposition 8.46.** inctw *strictly dominates* cluster. *If $\mathcal{C} \in \{\textbf{Horn}\} \cup \mathcal{A}cyc$ and $p \in \{\mathrm{db}_{\mathcal{C}}, \#\mathrm{neg}, \#\text{non-Horn}, \mathrm{lstr}, \mathrm{wfw}\}$, 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 $(T, \chi)$ be an arbitrary tree decomposition of $A_P$. We create a tree decomposition $(T, \chi')$ 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 $(T, \chi')$ 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 $(T, \chi')$. Thus, $(T, \chi')$ is a tree decomposition of the interaction graph $A_P$. Since the width of $(T, \chi')$ is less or equal to the width of $(T, \chi)$ 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 8.31 and 8.44. Hence inctw $\prec$ cluster.

Let $p \in \{\mathrm{db}_{\mathcal{C}}, \#\mathrm{neg}, \#\text{non-Horn}, \mathrm{lstr}, \mathrm{wfw}\}$ and $\mathcal{C} \in \{\textbf{Horn}\} \cup \mathcal{A}cyc$. 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 8.6, 8.10, 8.16, 8.23, and 8.44. Consequently, the second statement holds. □

### 8.7. Number of bad even cycles

Lin and Zhao [110] 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 8.47.** (See Lin and Zhao [110].) Let $P$ be a normal program. Then

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

**Proposition 8.48.** *For each $L \in \mathcal{AspFull}$, $L[\#\mathrm{badEvenCycles}]_{\mathrm{N}} \in$ FPT.*

**Observation 8.49.** *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 $\#\mathrm{badEvenCycles}(P_4^n) = 0$.*

2. *Consider program* $P_{51}^n$ *which contains n disjoint directed bad even cycles. Thus,* #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* #badEvenCycles$(P_{52}^n) =$ #badEvenCycles$(P_7^n) =$ #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* #badEvenCycles$(P_9^n) = n$.

**Proposition 8.50.** $\mathrm{db}_{\textbf{no-DBEC}}$ *strictly dominates* #badEvenCycles. *Moreover,* $\mathrm{db}_{\mathcal{C}}$ *and* #badEvenCycles *are incomparable for the remaining target classes* $\mathcal{C} \in \mathcal{A}_{cyc} \setminus \{\textbf{no-DBEC}\} \cup \{\textbf{Horn}\}$. *If* $p \in \{$#neg, #non-Horn, lstr, wfw, inctw, deptw, cluster, cyclecut$\}$, *then p and* #badEvenCycles *are incomparable.*

**Proof.** To see that $\mathrm{db}_{\textbf{no-DBEC}}$ 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, $\mathrm{db}_{\textbf{no-DBEC}}(P) \leq$ #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 $\mathrm{db}_{\textbf{no-DBEC}}(P_9^n) \leq 1$ and #badEvenCycles$(P_9^n) = n$ by Observations 8.6 and 8.49. It follows that $\mathrm{db}_{\textbf{no-DBEC}} \prec$ #badEvenCycles and the proposition holds.

To show the second statement, consider the programs $P_4^n$, $P_{52}^n$, and $P_9^n$ where $\mathrm{db}_{\mathcal{C}}(P_9^n) = 1$ for $\mathcal{C} \in \mathcal{A}_{cyc} \cup \{\textbf{Horn}\}$ and #badEvenCycles$(P_9^n) = n$; conversely $\mathrm{db}_{\mathcal{C}}(P_4^n) \geq n$ for $\mathcal{C} \in \{\textbf{Horn}, \textbf{no-C}, \textbf{no-BC}, \textbf{no-DC}, \textbf{no-DC2}, \textbf{no-EC}, \textbf{no-DEC}, \textbf{no-BEC}\}$, $\mathrm{db}_{\textbf{no-DBEC}}(P_{52}^n) \geq n$, and #badEvenCycles$(P_4^n) =$ #badEvenCycles$(P_{52}^n) = 0$. Hence $\mathrm{db}_{\mathcal{C}} \bowtie$ #badEvenCycles for $\mathcal{C} \in \mathcal{A}_{cyc} \setminus \{\textbf{no-DBEC}\} \cup \{\textbf{Horn}\}$ by Observations 8.6 and 8.49.

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 inctw$(P_7^n) \geq n - 1$ and deptw$(P_7^n) \geq n - 1$, $p(P_{52}^n) \geq n$ for $p \in \{$#neg, #non-Horn, lstr, wfw$\}$, cyclecut$(P_8^{m,n}) \geq (m-2)\log n$, cluster$(P_8^{m,n}) \geq (m-2)\log n$, and #badEvenCycles$(P_7^n) =$ #badEvenCycles$(P_8^{m,n}) =$ #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 #badEvenCycles$(P_{51}^n) =$ #badEvenCycles$(P_9^n) = n$ by Observations 8.6, 8.10, 8.16, 8.23, 8.31, 8.35, 8.44, and 8.49. Hence $p \bowtie$ #badEvenCycles for $p \in \{$#neg, #non-Horn, lstr, wfw, inctw, deptw, cluster, cyclecut$\}$. □

### 8.8. Number of positive cycles (loop formulas)

Fages [46], Lin and Zhao [109], and Lee and Lifschitz [105] have introduced compilations of normal programs and disjunctive programs to SAT, respectively. Fages [46] 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 [109] 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., Clasp3 [37] and Cmodels3 [108]. 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 8.51.** (See Fages [46].) Let $P$ be a normal program and $D_P^+$ its positive dependency digraph. Then

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

The program $P$ is called *tight* if #posCycles $= 0$.[8]

The parameter has been generalized to disjunctive programs by Lee and Lifschitz [105].

**Proposition 8.52.** *(See Fages [46].) For* $L \in \mathcal{A}sp\mathcal{R}eason$, $L[\#\mathrm{posCycles}]_{\mathrm{N}}$ *is* NP*-hard or* co-NP*-hard, even for tight programs.*

**Observation 8.53.** *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* #posCycles$(P_{32}^n) =$ #posCycles$(P_{53}^n) = n$.
2. *Consider program* $P_{51}^n$ *and* $P_7^n$ *where the positive dependency digraphs contain no cycle. Hence* #posCycles$(P_{51}^n) =$ #posCycles$(P_7^n) = 0$.
3. *Consider program* $P_8^{m,n}$. *Its positive dependency digraph contains only the cycle* $c_1, c_2, \ldots, c_n, c_{n+1}$; *thus,* #posCycles$(P_8^n) = 1$.

[8] Fages [46] used the term positive-order consistent instead of tight.

**Proposition 8.54.** *If* $\mathcal{C} \in \{\textbf{Horn}\} \cup \mathcal{A}_{cyc}$ *and* $p \in \{\mathrm{db}_{\mathcal{C}}, \#\mathrm{neg}, \#\text{non-Horn}, \mathrm{lstr}, \mathrm{wfw}, \mathrm{inctw}, \mathrm{deptw}, \mathrm{cluster}, \mathrm{cyclecut}, \#\mathrm{badEvenCycles}\}$, *then* $p$ *and* #posCycles *are incomparable.*

**Proof.** We observe incomparability from the programs $P_{32}^n$, $P_{51}^n$, $P_{53}^n$, $P_7^n$, and $P_8^{n,m}$. We have $p(P_{51}^n) \geq n$ for $p \in \{\mathrm{db}_{\mathcal{C}}$ #neg, #non-Horn, lstr, wfw, #badEvenCycles$\}$, $\mathrm{inctw}(P_7^n) \geq n-1$, $\mathrm{deptw}(P_7^n) \geq n-1$, $\mathrm{cyclecut}(P_8^{n,m}) \geq (m-2)\cdot \log n$, $\mathrm{cluster}(P_8^{n,m}) \geq (m-2)\cdot \log n$, and $\#\mathrm{posCycles}(P_{51}^n) = \#\mathrm{posCycles}(P_7^n) = 0$ as well as $\#\mathrm{posCycles}(P_8^{m,n}) = 1$; conversely for $p \in \{\mathrm{db}_{\mathcal{C}}$, #neg, #non-Horn, lstr, wfw, inctw, deptw$\}$ we have $p(P_{32}^n) \leq 2$, for $p \in \{\mathrm{cluster}, \mathrm{cyclecut}\}$ we have $p(P_{53}^n) \leq 2$ and $\#\mathrm{posCycles}(P_{32}^n) = \#\mathrm{posCycles}(P_{53}^n) = n$ by Observations 8.6, 8.10, 8.16, 8.23, 8.31, 8.35, 8.44, 8.49, and 8.53. Consequently, the proposition holds. □

### 8.9. Head-cycles

Ben-Eliyahu and Dechter [3] 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 compiled 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., [106]). 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 8.55.** (See Ben-Eliyahu and Dechter [3].) Let $P$ be a program and $D_P^+$ its positive dependency digraph. A *head-cycle* of $D_P^+$ is an $\{x, y\}$-cycle[9] 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 8.56.** Let $P$ be a program and $D_P^+$ its positive dependency digraph. Then

$$\#\mathrm{headCycles} := \left|\left\{c : c \text{ is a head-cycle of } D_P^+\right\}\right|$$

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

**Proposition 8.57.** *(See Ben-Eliyahu and Dechter [3].) Each* $L \in \mathcal{A}sp\mathcal{R}eason$ *is* NP*-hard or* co-NP*-hard, even for head-cycle-free programs.*

**Observation 8.58.** *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,* $\#\mathrm{headCycles}(P_{51}^n) = 0$.
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,* $\#\mathrm{headCycles}(P_{11}^n) = n$.

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 8.59 and then consider disjunctive programs Observation 8.60.

**Observation 8.59.** *If* $\mathcal{C} \in \{\textbf{Horn}\} \cup \mathcal{A}_{cyc}$ *and* $p \in \{\mathrm{db}_{\mathcal{C}}, \#\mathrm{neg}, \#\text{non-Horn}, \mathrm{lstr}, \mathrm{wfw}, \mathrm{inctw}, \mathrm{deptw}, \mathrm{cluster}, \mathrm{cyclecut}, \#\mathrm{badEvenCycles}, \#\mathrm{posCycles}\}$, *then* #headCycles *strictly dominates* $p^{\downarrow}$.

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

**Observation 8.60.** *If* $\mathcal{C} \in \{\textbf{Horn}\} \cup \mathcal{A}_{cyc}$*, then* $\mathrm{db}_{\mathcal{C}}$ *and* #headCycles *are incomparable. Moreover, if* $p \in \{\#\mathrm{neg}, \#\text{non-Horn}, \mathrm{lstr}, \mathrm{wfw}\}$*, 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 $\mathrm{db}_{\mathcal{C}} \geq n$ as well as $p(P_{51}^n) \geq n$ and $\#\mathrm{headCycles}(P_{51}^n) = 0$; and $p(P_{11}^n) = 1$ and $\#\mathrm{headCycles}(P_{11}^n) = n$ by Observations 8.6, 8.10, 8.16, 8.23, and 8.58. □

[9] See Section 5.2 for the definition of a $W$-cycle.

## 9. Summary and future work

We have introduced the backdoor approach 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. Our approach makes recent progress in fixed-parameter algorithmics applicable to answer set programming and establishes a unifying approach that accommodates several parameters from the literature. This framework gives rise to a detailed comparison of the various parameters in terms of their generality. We introduce a general method of lifting parameters from normal to disjunctive programs and establish several basic properties of this method. We further studied the preprocessing limits of ASP rules in terms of kernelization taking backdoor size as the parameter.

The results and concepts of this paper give rise to several research questions. For instance, it would be interesting to consider backdoors for target classes that contain programs with an exponential number of answer sets, 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. It would be interesting to enhance our backdoor approach to extended rules in particular to weight constraints.

An interesting further research direction is to study whether and how backdoors can be used to control modern heuristics in ASP solvers to obtain a speed-up. 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 [101] and tuning of parameter values [88], modifying the branching heuristic by caching truth values of atoms and reusing them [127], modifying the atom score initially, or while learning conflict clauses, or at a certain depth of the search). So far various studies and theoretical considerations 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 and ASP, where both positive [80,81, 143,75] and negative effects [96,95] have been observed.

## References


[1] Benjamin Andres, Benjamin Kaufmann, Oliver Mattheis, Torsten Schaub, Unsatisfiability-based optimization in clasp, in: A. Dovier, V. Santos Costa (Eds.), Technical Communications of the 28th International Conference on Logic Programming, ICLP'12, in: Leibniz International Proceedings in Informatics (LIPIcs), vol. 17, 2012, pp. 212–221.

[2] Krzysztof R. Apt, Howard A. Blair, Adrian Walker, Towards a theory of declarative knowledge, in: Foundations of Deductive Databases and Logic Programming, 1988, pp. 89–148.

[3] Rachel Ben-Eliyahu, Rina Dechter, Propositional semantics for disjunctive logic programs, Ann. Math. Artif. Intell. 12 (1) (1994) 53–87.

[4] Rachel Ben-Eliyahu, A hierarchy of tractable subsets for computing stable models, J. Artif. Intell. Res. 5 (1996) 27–52.

[5] Nicole Bidoít, Christine Froidevaux, Negation by default and unstratifiable logic programs, Theor. Comput. Sci. 78 (1) (1991) 85–112.

[6] Hans L. Bodlaender, Arie M.C.A. Koster, Combinatorial optimization on graphs of bounded treewidth, Comput. J. 51 (3) (2008) 255–269.

[7] Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, Danny Hermelin, On problems without polynomial kernels, J. Comput. Syst. Sci. 75 (8) (2009) 423–434.

[8] Hans L. Bodlaender, A tourist guide through treewidth, Acta Cybern. 11 (1–2) (1993) 1–22.

[9] Hans L. Bodlaender, Treewidth: algorithmic techniques and results, in: Igor Prívara, Peter Ružička (Eds.), Proceedings of the 22nd International Symposium on Mathematical Foundations of Computer Science, MFCS'97, in: Lecture Notes in Computer Science, vol. 1295, Springer Verlag, 1997, pp. 19–36.

[10] Hans L. Bodlaender, Discovering treewidth, in: Peter Vojtáš, Mária Bieliková, Bernadette Charron-Bost, Ondrej Sýkora (Eds.), Proceedings of the 31st Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM'05, in: Lecture Notes in Computer Science, vol. 3381, Springer Verlag, 2005, pp. 1–16.

[11] John A. Bondy, U.S.R. Murty, Graph Theory, Graduate Texts in Mathematics, vol. 244, Springer Verlag, New York, 2008.

[12] Paul Bonsma, Daniel Lokshtanov, Feedback vertex set in mixed graphs, in: Frank Dehne, John Iacono, Jörg-Rüdiger Sack (Eds.), Proceedings of the 12th International Symposium on Algorithms and Data Structures, WADS'11, in: Lecture Notes in Computer Science, vol. 6844, Springer Verlag, 2011, pp. 122–133.

[13] Stefan Brass, Jürgen Dix, Characterizations of the disjunctive well-founded semantics: confluent calculi and iterated GCWA, J. Autom. Reason. 20 (1998) 143–165.

[14] Marco Cadoli, Maurizio Lenzerini, The complexity of propositional closed world reasoning and circumscription, J. Comput. Syst. Sci. 48 (2) (1994) 255–310.

[15] Shaowei Cai, Kaile Su, Qingliang Chen, Ewls: a new local search for minimum vertex cover, in: Nestor Rychtyckyj, Daniel G. Shapiro (Eds.), Proceedings of the 24th AAAI Conference on Artificial Intelligence, AAAI'10, Atlanta, GA, USA, July 2010, pp. 45–50.

[16] Shaowei Cai, Kaile Su, Chuan Luo, Abdul Sattar, Numvc: an efficient local search algorithm for minimum vertex cover, J. Artif. Intell. Res. 46 (2013) 687–716.

[17] Shaowei Cai, Kaile Su, Abdul Sattar, Local search with edge weighting and configuration checking heuristics for minimum vertex cover, Artif. Intell. 175 (2011) 1672–1696.

[18] Francesco Calimeri, Giovambattista Ianni, Francesco Ricca, Mario Alviano, Annamaria Bria, Gelsomina Catalano, Susanna Cozza, Wolfgang Faber, Onofrio Febbraro, Nicola Leone, Marco Manna, Alessandra Martello, Claudio Panetta, Simona Perri, Kristian Reale, Maria Santoro, Marco Sirianni, Giorgio Terracina, Pierfrancesco Veltri, The third answer set programming competition: preliminary report of the system competition track, in: James Delgrande, Wolfgang Faber (Eds.), Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR'11, Vancouver, Canada, in: Lecture Notes in Computer Science, vol. 6645, Springer Verlag, 2011, pp. 388–403, https://www.mat.unical.it/aspcomp2011/OfficialProblemSuite.

[19] Ashok K. Chandra, David Harel, Horn clause queries and generalizations, J. Log. Program. 2 (1) (1985) 1–15.

[20] Hubie Chen, Yannet Interian, A model for generating random quantified Boolean formulas, in: Leslie Pack Kaelbling, Alessandro Saffiotti (Eds.), Proceedings of the 19th International Joint Conference on Artificial Intelligence, IJCAI'05, Edinburgh, Scotland, UK, Professional Book Center, August 2005, pp. 66–71.

[21] Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, Igor Razgon, A fixed-parameter algorithm for the directed feedback vertex set problem, J. ACM 55 (5) (2008) 1–19.

[22] Jianer Chen, Iyad A. Kanj, Ge Xia, Improved upper bounds for vertex cover, Theor. Comput. Sci. 411 (40–42) (September 2010) 3736–3756.
[23] Rajesh Chitnis, Marek Cygan, Mohammadtaghi Hajiaghayi, Dániel Marx, Directed subset feedback vertex set is fixed-parameter tractable, in: Artur Czumaj, Kurt Mehlhorn, Andrew Pitts, Roger Wattenhofer (Eds.), Proceedings of the 39th International on Colloquium Automata, Languages, and Programming, ICALP'12, Warwick, UK, in: Lecture Notes in Computer Science, vol. 7391, Springer Verlag, July 2012, pp. 230–241.
[24] Vasek Chvatal, Linear Programming, Series of Books in the Mathematical Sciences, W.H. Freeman and Company, New York, 1983.
[25] William Cook, Thorsten Koch, Daniel E. Steffy, Kati Wolter, A hybrid branch-and-bound approach for exact rational mixed-integer programming, Math. Program. Comput. 5 (3) (2013) 305–344.
[26] Marek Cygan, Marcin Pilipczuk, Michał Pilipczuk, Jakub Onufry Wojtaszczyk, Subset feedback vertex set is fixed-parameter tractable, in: Luca Aceto, Monika Henzinger, Jiří Sgall (Eds.), Proceedings of the 38th International Colloquium on Automata, Languages and Programming, ICALP'11, in: Lecture Notes in Computer Science, vol. 6755, Springer Verlag, 2011, pp. 449–461.
[27] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, Andrei Voronkov, Complexity and expressive power of logic programming, ACM Comput. Surv. 33 (3) (2001) 374–425.
[28] Ronald DeHaan, Stefan Szeider, The parameterized complexity of reasoning problems beyond NP, in: Chitta Baral, Giuseppe De Giacomo, Thomas Eiter (Eds.), Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning, KR'14, Vienna, Austria, The AAAI Press, 2014, Full version available from arXiv:1312.1672.
[29] Marc Denecker, Joost Vennekens, Stephen Bond, Martin Gebser, Mirosław Truszczyński, The second answer set programming competition, in: Esra Erdem, Fangzhen Lin, Torsten Schaub (Eds.), Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR'09, Potsdam, Germany, in: Lecture Notes in Computer Science, vol. 5753, Springer Verlag, September 2009, pp. 637–654.
[30] Reinhard Diestel, Graph Theory, 2nd edition, Graduate Texts in Mathematics, vol. 173, Springer Verlag, New York, 2000.
[31] Bistra N. Dilkina, Carla P. Gomes, 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, in: Lecture Notes in Computer Science, vol. 4741, Springer Verlag, 2007, pp. 256–270.
[32] Bistra N. Dilkina, Carla P. Gomes, Ashish Sabharwal, Tradeoffs in backdoor detection for Sat and Unsat formulas, in: Proceedings of the 10th International Symposium on Artificial Intelligence and Mathematics, ISAIM'08, Fort Lauderdale, FL, USA, January 2008, pp. 256–270.
[33] William F. Dowling, Jean H. Gallier, Linear-time algorithms for testing the satisfiability of propositional Horn formulae, J. Log. Program. 1 (3) (1984) 267–284.
[34] Rodney G. Downey, Michael R. Fellows, Parameterized Complexity, Monographs in Computer Science, Springer Verlag, New York, 1999.
[35] Rodney G. Downey, Michael R. Fellows, Fundamentals of Parameterized Complexity, Texts in Computer Science, Springer Verlag, London, UK, 2013.
[36] Rodney G. Downey, Michael R. Fellows, Ulrike Stege, Parameterized complexity: a framework for systematically confronting computational intractability, in: Contemporary Trends in Discrete Mathematics: from DIMACS and DIMATIA to the Future, in: AMS–DIMACS, vol. 49, American Mathematical Society, 1999, pp. 49–99.
[37] Christian Drescher, Martin Gebser, Torsten Grote, Benjamin Kaufmann, Arne König, Max Ostrowski, Torsten Schaub, Conflict-driven disjunctive answer set solving, in: Gerhard Brewka, Jérôme Lang (Eds.), Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning, KR'08, Sydney, NSW, Australia, The AAAI Press, September 2008, pp. 422–432.
[38] Christian Drescher, Martin Gebser, Benjamin Kaufmann, Torsten Schaub, Heuristics in conflict resolution, in: Maurice Pagnucco, Michael Thielscher (Eds.), Proceedings of the 12th International Workshop on Nonmonotonic Reasoning, NMR'08, vol. UNSW-CSE-TR-0819, The University of New South Wales, 2008, pp. 141–149.
[39] Paul E. Dunne, Computational properties of argument systems satisfying graph-theoretic constraints, Artif. Intell. 171 (10–15) (2007) 701–729.
[40] Wolfgang Dvořák, Sebastian Ordyniak, Stefan Szeider, Augmenting tractable fragments of abstract argumentation, Artif. Intell. 186 (2012) 157–173.
[41] Thomas Eiter, Georg Gottlob, On the computational cost of disjunctive logic programming: propositional case, Ann. Math. Artif. Intell. 15 (3–4) (1995) 289–323.
[42] T. Eiter, M. Fink, H. Tompits, S. Woltran, Simplifying logic programs under uniform and strong equivalence, in: Ilkka Niemelä, Vladimir Lifschitz (Eds.), Proceedings 7th International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR'07, Tempe, AZ, USA, in: Lecture Notes in Computer Science, vol. 2923, Springer Verlag, May 2004, pp. 87–99.
[43] Esra Erdem, Vladimir Lifschitz, Tight logic programs, Theory Pract. Log. Program. 3 (July 2003) 499–518.
[44] William A. Stein, et al., Sage mathematics software (version 5.1.rc0), the Sage development team, http://www.sagemath.org, 2012.
[45] Wolfgang Faber, Nicola Leone, Cristinel Mateis, Gerald Pfeifer, Using database optimization techniques for nonmonotonic reasoning, in: I.O. Committee (Ed.), Proceedings of the 7th International Workshop on Deductive Databases and Logic Programming, DDLP'99, Prolog Association of Japan, 1999, pp. 135–139.
[46] Francois Fages, Consistency of Clark's completion and existence of stable models, Log. Methods Comput. Sci. 1 (1) (1994) 51–60.
[47] Johannes K. Fichte, Stefan Szeider, Backdoors to tractable answer-set programming, in: Toby Walsh (Ed.), Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI'11, Barcelona, Catalonia, Spain, AAAI Press/IJCAI, July 2011, pp. 863–868.
[48] Johannes K. Fichte, Stefan Szeider, Backdoors to normality for disjunctive logic programs, in: Marie des Jardins, Michael Littman (Eds.), Proceedings of the 27th AAAI Conference on Artificial Intelligence, AAAI'13, Bellevue, WA, USA, The AAAI Press, July 2013, pp. 320–327, A preliminary version of the paper was presented at the Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP'12).
[49] Johannes K. Fichte, The good, the bad, and the odd: cycles in answer-set programs, in: Daniel Lassiter, Marija Slavkovik (Eds.), Proceedings of the 23rd European Summer School in Logic, Language and Information (ESSLLI'11), New Directions in Logic, Language and Computation (ESSLLI'10 and ESSLLI'11 Student Sessions, Selected Papers Series), in: Lecture Notes in Computer Science, vol. 7415, Springer Verlag, 2012, pp. 78–90.
[50] Jörg Flum, Martin Grohe, Parameterized Complexity Theory, Theor. Comput. Sci., vol. XIV, Springer Verlag, Berlin, 2006.
[51] Lance Fortnow, Rahul Santhanam, Infeasibility of instance compression and succinct PCPs for NP, J. Comput. Syst. Sci. 77 (1) (2011) 91–106.
[52] Steven Fortune, John Hopcroft, James Wyllie, The directed subgraph homeomorphism problem, Theor. Comput. Sci. 10 (2) (1980) 111–121.
[53] Marco Gario, Horn backdoor detection via vertex cover: benchmark description, in: Adrian Balint, Anton Belov, Daniel Diepold, Simon Gerber, Matti Järvisalo, Carsten Sinz (Eds.), Proceedings of SAT Challenge 2012; Solver and Benchmark Descriptions, 2012.
[54] Serge Gaspers, Stefan Szeider, Backdoors to acyclic SAT, in: Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, Roger Wattenhofer (Eds.), Proceedings of the 39th International Colloquium on Automata, Languages, and Programming, ICALP'12, Warwick, UK, in: Lecture Notes in Computer Science, vol. 7391, Springer Verlag, July 2012, pp. 363–374.
[55] Serge Gaspers, Stefan Szeider, Backdoors to satisfaction, in: Hans Bodlaender, Rod Downey, Fedor Fomin, Dániel Marx (Eds.), The Multivariate Algorithmic Revolution and Beyond, in: Lecture Notes in Computer Science, vol. 7370, Springer Verlag, Heidelberg, Germany, 2012, pp. 287–317.
[56] Serge Gaspers, Stefan Szeider, Strong backdoors to nested satisfiability, in: Alessandro Cimatti, Roberto Sebastiani (Eds.), Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing, SAT'12, Trento, Italy, in: Lecture Notes in Computer Science, vol. 7317, Springer Verlag, June 2012, pp. 72–85.
[57] Serge Gaspers, Stefan Szeider, Strong backdoors to bounded treewidth SAT, in: Omer Reingold (Ed.), Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science, FOCS'13, Berkeley, California, USA, October 27–29, IEEE Computer Soc., 2013, pp. 489–498.
[58] Serge Gaspers, Stefan Szeider, Guarantees and limits of preprocessing in constraint satisfaction and reasoning, Artif. Intell. 216 (2014) 1–19.

[59] Serge Gaspers, Sebastian Ordyniak, M.S. Ramanujan, Saket Saurabh, Stefan Szeider, Backdoors to q-Horn, in: Natacha Portier, Thomas Wilke (Eds.), Proceedings of the 30th International Symposium on Theoretical Aspects of Computer Science, STACS'13, San Francisco, CA, in: Leibniz International Proceedings in Informatics (LIPIcs), vol. 20, Schloss Dagstuhl, 2013, pp. 67–79.
[60] Martin Gebser, Roland Kaminski, Personal communication, 2012.
[61] Martin Gebser, Torsten Schaub, Asparagus, http://asparagus.cs.uni-potsdam.de, 2009.
[62] M. Gebser, B. Kaufmann, A. Neumann, T. Schaub, Conflict-driven answer set solving, in: Manuela M. Veloso (Ed.), Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI'07, Hyderabad, India, IJCAI, January 2007, pp. 386–392.
[63] Martin Gebser, Lengning Liu, Gayathri Namasivayam, André Neumann, Torsten Schaub, Mirosław Truszczyński, The first answer set programming system competition, in: Chitta Baral, Gerhard Brewka, John Schlipf (Eds.), Proceedings of the 9th Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR'07, Tempe, AZ, USA, in: Lecture Notes in Computer Science, vol. 4483, Springer Verlag, May 2007, pp. 3–17.
[64] Martin Gebser, Torsten Schaub, Sven Thiele, Gringo: a new grounder for answer set programming, in: Chitta Baral, Gerhard Brewka, John Schlipf (Eds.), Proceedings of the 9th International Conference Logic Programming and Nonmonotonic Reasoning, LPNMR'07, Tempe, AZ, USA, in: Lecture Notes in Computer Science, vol. 4483, Springer Verlag, May 2007, pp. 266–271.
[65] Martin Gebser, Benjamin Kaufmann, André Neumann, Torsten Schaub, Advanced preprocessing for answer set solving, in: Malik Ghallab, Constantine D. Spyropoulos, Nikos Fakotakis, Nikolaos M. Avouris (Eds.), Proceedings of the 18th European Conference on Artificial Intelligence, ECAI'08, Patras, Greece, in: Front. Artif. Intell. Appl., vol. 178, IOS Press, July 2008, pp. 15–19.
[66] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, Sven Thiele, A user's guide to gringo, clasp, clingo, and iclingo, Technical report, University Potsdam, 2010.
[67] M. Gebser, T. Schaub, S. Thiele, P. Veber, Detecting inconsistencies in large biological networks with answer set programming, Theory Pract. Log. Program. 11 (2–3) (2011) 323–360.
[68] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Torsten Schaub, Challenges in answer set solving, in: Marcello Balduccini, TranCao Son (Eds.), Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning – Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday, in: Lecture Notes in Artificial Intelligence, vol. 6565, Springer Verlag, 2011, pp. 74–90.
[69] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Torsten Schaub, Multi-criteria optimization in answer set programming, in: John Gallagher, Michael Gelfond (Eds.), Technical Communications of the 27th International Conference on Logic Programming, ICLP'11, Dagstuhl, Germany, in: Leibniz International Proceedings in Informatics (LIPIcs), vol. 11, Dagstuhl Publishing, 2011, pp. 1–10.
[70] Martin Gebser, Roland Kaminski, Torsten Schaub, Complex optimization in answer set programming, Theory Pract. Log. Program. 11 (4–5) (2011) 821–839.
[71] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, Marius Schneider, Potassco: the Potsdam answer set solving collection, AI Commun. 24 (2) (2011) 107–124.
[72] M. Gebser, B. Kaufmann, T. Schaub, Conflict-driven answer set solving: from theory to practice, Artif. Intell. 187 (188) (2012) 52–89.
[73] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Torsten Schaub, Answer Set Solving in Practice, Morgan & Claypool, 2012.
[74] Martin Gebser, Thomas Glase, Orkunt Sabuncu, Torsten Schaub, Matchmaking with answer set programming, in: Pedro Cabalar, Tran Cao Son (Eds.), Proceedings of 12th International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR'13, Corunna, Spain, in: Lecture Notes in Computer Science, vol. 8148, Springer Verlag, 2013, pp. 342–347.
[75] Martin Gebser, Benjamin Kaufmann, Ramon P. Otero, Javier Romero, Torsten Schaub, Philipp Wanko, Domain-specific heuristics in answer set programming, in: Marie des Jardins, Michael Littman (Eds.), Proceedings of the 27th AAAI Conference on Artificial Intelligence, AAAI'13, Bellevue, WA, USA, The AAAI Press, July 2013, pp. 350–356.
[76] Martin Gebser, Tomi Janhunen, Jussi Rintanen, Answer set programming as SAT modulo acyclicity, in: Torsten Schaub, Gerhard Friedrich, Barry O'Sullivan (Eds.), Proceedings of the 21st European Conference on Artificial Intelligence, ECAI'14, Prague, Czech Republic, in: Front. Artif. Intell. Appl., vol. 263, IOS Press, August 2014, pp. 351–356.
[77] Allen Van Gelder, Negation as failure using tight derivations for general logic programs, J. Log. Program. 6 (1–2) (1989) 109–133.
[78] Michael Gelfond, Vladimir Lifschitz, The stable model semantics for logic programming, in: Robert A. Kowalski, Kenneth A. Bowen (Eds.), Proceedings of the 5th International Conference and Symposium, ICLP/SLP'88, Seattle, Washington, vol. 2, MIT Press, August 1988, pp. 1070–1080.
[79] Michael Gelfond, Vladimir Lifschitz, Classical negation in logic programs and disjunctive databases, New Gener. Comput. 9 (3/4) (1991) 365–386.
[80] E. Giunchiglia, A. Massarotto, R. Sebastiani, Act, and the rest will follow: exploiting determinism in planning as satisfiability, in: Jack Mostow, Charles Rich (Eds.), Proceedings of the 15th National Conference on Artificial Intelligence, AAAI'98, Madison, WI, USA, The AAAI Press, 1998, pp. 948–953.
[81] E. Giunchiglia, M. Maratea, A. Tacchella, Dependent and independent variables in propositional satisfiability, in: Logics in Artificial Intelligence, 2002, pp. 296–307.
[82] Enrico Giunchiglia, Yuliya Lierler, Marco Maratea, Answer set programming based on propositional satisfiability, J. Autom. Reason. 36 (4) (2006) 345–377.
[83] Georg Gottlob, Reinhard Pichler, Hypergraphs in model checking: acyclicity and hypertree-width versus clique-width, SIAM J. Comput. 33 (2) (2004) 351–378.
[84] Georg Gottlob, Stefan Szeider, Fixed-parameter algorithms for artificial intelligence, constraint satisfaction and database problems, Comput. J. 51 (3) (2008) 303–325.
[85] Georg Gottlob, Francesco Scarcello, Martha Sideri, Fixed-parameter complexity in AI and nonmonotonic reasoning, Artif. Intell. 138 (1–2) (2002) 55–86.
[86] Georg Gottlob, Reinhard Pichler, Fang Wei, Bounded treewidth as a key to tractability of knowledge representation and reasoning, Artif. Intell. 174 (1) (2010) 105–132.
[87] Inc. Gurobi Optimization, Gurobi optimizer reference manual, Version 5.0.2, 2014.
[88] Holger H. Hoos, Automated algorithm configuration and parameter tuning, in: Youssef Hamadi, Eric Monfroy, Frédéric Saubion (Eds.), Autonomous Search, Springer Verlag, 2012, pp. 37–71.
[89] IBM, IBM ILOG CPLEX optimization studio CPLEX user's manual, version 12 release 4 edition, 2011.
[90] Michael Jakl, Reinhard Pichler, Stefan Woltran, Answer-set programming with bounded treewidth, in: Craig Boutilier (Ed.), Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI'09, Pasadena, CA, USA, vol. 2, Elsevier Science Publishers, North-Holland, July 2009, pp. 816–822.
[91] Tomi Janhunen, Ilkka Niemelä, Compact translations of non-disjunctive answer set programs to propositional clauses, in: Marcello Balduccini, Tran Son (Eds.), Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning – Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday, in: Lecture Notes in Artificial Intelligence, vol. 6565, Springer Verlag, 2011, pp. 111–130.
[92] Tomi Janhunen, Ilkka Niemelä, Dietmar Seipel, Patrik Simons, Jia-Huai You, Unfolding partiality and disjunctions in stable model semantics, ACM Trans. Comput. Log. 7 (1) (2006) 1–37.
[93] Tomi Janhunen, Ilkka Niemelä, Mark Sevalnev, Computing stable models via reductions to difference logic, in: Esra Erdem, Fangzhen Lin, Torsten Schaub (Eds.), Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR'09, Potsdam, Germany, in: Lecture Notes in Computer Science, vol. 5753, Springer Verlag, September 2009, pp. 142–154.

[94] Mikoláš Janota, Joao Marques-Silva, A tool for circumscription-based MUS membership testing, in: James P. Delgrande, Wolfgang Faber (Eds.), Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR'11, Vancouver, Canada, in: Lecture Notes in Computer Science, vol. 6645, Springer Verlag, May 2011, pp. 266–271.
[95] Matti Järvisalo, Tommi Junttila, Limitations of restricted branching in clause learning, Constraints 14 (3) (2009) 325–356.
[96] M. Järvisalo, Ilkka Niemelä, The effect of structural branching on the efficiency of clause learning SAT solving: an experimental study, J. Algorithms 63 (1–3) (2008) 90–113.
[97] Holger Jost, Orkunt Sabuncu, Torsten Schaub, Suggesting new interactions related to events in a social network for elderly, in: Proceedings of the 26th BCS Conference on Human Computer Interaction, HCI'12, Birmingham, UK, British Computer Society, Swinton, September 2012.
[98] Naonori Kakimura, Ken-ichi Kawarabayashi, Yusuke Kobayashi, Erdös-Pósa property and its algorithmic applications: parity constraints, subset feedback set, and subset packing, in: Dana Randall (Ed.), Proceedings of the 23rd Annual ACM–SIAM Symposium on Discrete Algorithms, SODA'12, San Francisco, CA, USA, Society for Industrial and Applied Mathematics (SIAM), 2012, pp. 1726–1736.
[99] Kanchana Kanchanasut, Peter J. Stuckey, Transforming normal logic programs to constraint logic programs, Theor. Comput. Sci. 105 (1) (1992) 27–56.
[100] Richard M. Karp, 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, Los Angeles, CA, USA, April 1980, pp. 302–309.
[101] Hadi Katebi, Karem A. Sakallah, João P. Marques-Silva, Empirical study of the anatomy of modern Sat solvers, in: Karem A. Sakallah, Laurent Simon (Eds.), Proceedings of the 14th International Conference on Theory and Applications of Satisfiability Testing, SAT'11, Ann Arbor, MI, USA, in: Lecture Notes in Computer Science, vol. 6695, Springer Verlag, June 2011, pp. 343–356.
[102] Kenichi Kawarabayashi, Yusuke Kobayashi, Fixed-parameter tractability for the subset feedback set problem and the s-cycle packing problem, Technical report, University of Tokyo, Japan, 2010.
[103] Stephan Kottler, Michael Kaufmann, Carsten Sinz, A new bound for an NP-hard subclass of 3-SAT using backdoors, in: Hans Kleine Büning, Xishun Zhao (Eds.), Proceedings of the 11th International Conference on Theory and Applications of Satisfiability Testing, SAT'08, Guangzhou, China, in: Lecture Notes in Computer Science, vol. 4996, Springer Verlag, May 2008, pp. 161–167.
[104] Andrea S. Lapaugh, Christos H. Papadimitriou, The even-path problem for graphs and digraphs, Networks 14 (4) (1984) 507–513.
[105] Joohyung Lee, Vladimir Lifschitz, Loop formulas for disjunctive logic programs, in: Catuscia Palamidessi (Ed.), Logic Programming, Mumbai, India, in: Lecture Notes in Computer Science, vol. 2916, Springer Verlag, 2003, pp. 451–465.
[106] Nicola Leone, Pasquale Rullo, Francesco Scarcello, Disjunctive stable models: unfounded sets, fixpoint semantics, and computation, Inf. Comput. 135 (2) (1997) 69–112.
[107] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, Francesco Scarcello, The DLV system for knowledge representation and reasoning, ACM Trans. Comput. Log. 7 (3) (2006) 499–562.
[108] Yuliya Lierler, CMODELS – SAT-based disjunctive answer set solver, in: Chitta Baral, Gianluigi Greco, Nicola Leone, Giorgio Terracina (Eds.), Proceedings of the 8th International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR'05, Diamante, Italy, in: Lecture Notes in Computer Science, vol. 3662, Springer Verlag, 2005, pp. 447–451.
[109] Fangzhen Lin, Jicheng Zhao, On tight logic programs and yet another translation from normal logic programs to propositional logic, in: Georg Gottlob, Toby Walsh (Eds.), Proceedings of the 18th International Joint Conference on Artificial Intelligence, IJCAI'03, Acapulco, Mexico, Morgan Kaufmann, August 2003, pp. 853–858.
[110] Fangzhen Lin, Xishun Zhao, On odd and even cycles in normal logic programs, in: Anthony G. Cohn (Ed.), Proceedings of the 19th National Conference on Artificial Intelligence, AAAI'04, San Jose, CA, USA, The AAAI Press, July 2004, pp. 80–85.
[111] Fangzhen Lin, Yuting Zhao, ASSAT: computing answer sets of a logic program by SAT solvers, Artif. Intell. 157 (1–2) (2004) 115–137.
[112] Guohua Liu, Tomi Janhunen, Ilkka Niemelä, Answer set programming via mixed integer programming, in: Sheila McIlraith, Thomas Eiter (Eds.), Proceedings of the 13th International Conference on the Principles of Knowledge Representation and Reasoning, KR'12, Rome, Italy, The AAAI Press, 2012, pp. 32–42.
[113] Wiktor Marek, Mirosław Truszczyński, Autoepistemic logic, J. ACM 38 (3) (1991) 588–619.
[114] Wiktor Marek, Mirosław Truszczyński, Computing intersection of autoepistemic expansions, in: V. Wiktor Marek, V.S. Subrahmanian, Anil Nerode (Eds.), Proceedings of the 1st International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR'91, Washington, D.C., USA, MIT Press, July 1991, pp. 37–50.
[115] Victor W. Marek, Mirosław Truszczyński, Stable models and an alternative logic programming paradigm, in: Krzysztof R. Apt, Victor W. Marek, Mirosław Truszczyński, David S. Warren (Eds.), The Logic Programming Paradigm: A 25-Year Perspective, in: Artificial Intelligence, Springer Verlag, Berlin, Germany, September 1999, pp. 375–398.
[116] Pranabendu Misra, Venkatesh Raman, M.S. Ramanujan, Saket Saurabh, Parameterized algorithms for even cycle transversal, in: Martin Charles Golumbic, Michal Stern, Avivit Levy, Gila Morgenstern (Eds.), Proceedings of the 38th International Workshop on Graph-Theoretic Concepts in Computer Science, WG'12, in: Lecture Notes in Computer Science, vol. 7551, Springer Verlag, 2012, pp. 172–183.
[117] Marco Montalva, Julio Aracena, Anahí Gajardo, On the complexity of feedback set problems in signed digraphs, Electron. Notes Discrete Math. 30 (2008) 249–254.
[118] Michael Morak, Stefan Woltran, Preprocessing of complex non-ground rules in answer set programming, in: Agostino Dovier, Vítor Santos Costa (Eds.), Technical Communications of the 28th International Conference on Logic Programming, ICLP'12, Dagstuhl, Germany, in: Leibniz International Proceedings in Informatics (LIPIcs), vol. 17, Dagstuhl Publishing, 2012, pp. 247–258.
[119] Michael Morak, Reinhard Pichler, Stefan Rümmele, Stefan Woltran, A dynamic-programming based ASP-solver, in: Tomi Janhunen, Ilkka Niemelä (Eds.), Proceedings of 12th European Conference on Logics in Artificial Intelligence, JELIA'10, Helsinki, Finland, in: Lecture Notes in Computer Science, vol. 6341, Springer Verlag, September 2010, pp. 369–372.
[120] Rolf Niedermeier, Invitation to Fixed-Parameter Algorithms, Oxford Lecture Series in Mathematics and Its Applications, vol. 31, Oxford University Press, New York, NY, USA, 2006.
[121] Ilkka Niemelä, Jussi Rintanen, On the impact of stratification on the complexity of nonmonotonic reasoning, J. Appl. Non-Class. Log. 4 (2) (1994) 141–179.
[122] Ilkka Niemelä, Logic programs with stable model semantics as a constraint programming paradigm, Ann. Math. Artif. Intell. 25 (3) (1999) 241–273.
[123] Naomi Nishimura, Prabhakar Ragde, Stefan Szeider, Detecting backdoor sets with respect to Horn and binary clauses, in: Holger H. Hoos, David G. Mitchell (Eds.), Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing, SAT'04, Vancouver, BC, Canada, in: Lecture Notes in Computer Science, vol. 3542, Springer Verlag, May 2004, pp. 96–103.
[124] Naomi Nishimura, Prabhakar Ragde, Stefan Szeider, Solving #SAT using vertex covers, Acta Inform. 44 (7–8) (2007) 509–523.
[125] Andreas Pfandler, Stefan Rümmele, Stefan Szeider, Backdoors to abduction, in: Francesca Rossi (Ed.), Proceedings of the 23rd International Joint Conference on Artificial Intelligence, IJCAI'13, Beijing, China, The AAAI Press, August 2013, pp. 1046–1052.
[126] Reinhard Pichler, Stefan Rümmele, Stefan Woltran, Belief revision with bounded treewidth, in: Esra Erdem, Fangzhen Lin, Torsten Schaub (Eds.), Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR'09, Potsdam, Germany, in: Lecture Notes in Computer Science, vol. 5753, Springer Verlag, 2009, pp. 250–263.

[127] Knot Pipatsrisawat, Adnan Darwiche, A lightweight component caching scheme for satisfiability solvers, in: João P. Marques-Silva, Karem A. Sakallah (Eds.), Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing, SAT'07, Lisbon, Portugal, in: Lecture Notes in Computer Science, vol. 4501, Springer Verlag, May 2007, pp. 294–299.
[128] Igor Razgon, Barry O'Sullivan, Almost 2-SAT is fixed parameter tractable, J. Comput. Syst. Sci. 75 (8) (2009) 435–450.
[129] Francesco Ricca, G. Grasso, Mario Alviano, Marco Manna, V. Lio, S. Iiritano, Nicola Leone, Team-building with answer set programming in the Gioia-Tauro seaport, Theory Pract. Log. Program. 12 (2012) 361–381.
[130] Neil Robertson, P.D. Seymour, Graph minors. III. Planar tree-width, J. Comb. Theory, Ser. B 36 (1) (1984) 49–64.
[131] Neil Robertson, P.D. Seymour, Graph minors – a survey, in: Surveys in Combinatorics 1985: Invited Papers for the 10th British Combinatorial Conference, in: London Mathematical Society Lecture Note Series, Cambridge University Press, Cambridge, 1985, pp. 153–171.
[132] Neil Robertson, P.D. Seymour, Graph minors. II. Algorithmic aspects of tree-width, J. Algorithms 7 (3) (1986) 309–322.
[133] Neil Robertson, P.D. Seymour, Robin Thomas, Permanents, Pfaffian orientations, and even directed circuits, Ann. Math. 150 (3) (1999) 929–975.
[134] Frances Rosamond, Table of races, in: Parameterized Complexity Newsletter, 2010, pp. 4–5, http://fpt.wikidot.com/.
[135] Yongshao Ruan, Henry A. Kautz, Eric Horvitz, The backdoor key: a path to understanding problem hardness, in: Deborah L. McGuinness, George Ferguson (Eds.), Proceedings of the 19th National Conference on Artificial Intelligence, AAAI'04, San Jose, CA, USA, The AAAI Press, July 2004, pp. 124–130.
[136] Marko Samer, Stefan Szeider, Backdoor trees, in: Robert C. Holte, Adele E. Howe (Eds.), Proceedings of 23rd Conference on Artificial Intelligence, AAAI'08, Vancouver, BC, Canada, The AAAI Press, July 2008, pp. 363–368.
[137] Marko Samer, Stefan Szeider, Backdoor sets of quantified Boolean formulas, J. Autom. Reason. 42 (1) (2009) 77–97.
[138] Marko Samer, Stefan Szeider, Fixed-parameter tractability, in: Armin Biere, Marijn Heule, Hans van Maaren, Toby Walsh (Eds.), Handbook of Satisfiability, IOS Press, 2009, pp. 425–454, chapter 13.
[139] C.P. Schnorr, On self-transformable combinatorial problems, in: H. König, B. Korte, K. Ritter (Eds.), Mathematical Programming at Oberwolfach, in: Mathematical Programming Studies, vol. 14, Springer Verlag, 1981, pp. 225–243.
[140] Alexander Schrijver, Theory of Linear and Integer Programming, John Wiley & Sons, 1998.
[141] Patrik Simons, Ilkka Niemelä, Timo Soininen, Extending and implementing the stable model semantics, in: Knowledge Representation and Logic Programming, Artif. Intell. 138 (1–2) (2002) 181–234.
[142] Sinz Carsten, Towards an optimal CNF encoding of boolean cardinality constraints, in: Peter van Beek (Ed.), Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming, CP'05, Sitges (Barcelona), Spain, in: Lecture Notes in Computer Science, vol. 3709, Springer Verlag, 2005, pp. 827–831.
[143] Ofer Strichman, Tuning SAT checkers for bounded model checking, in: E. Allen Emerson, Aravinda Prasad Sistla (Eds.), Proceedings of the 12th International Conference on Computer Aided Verification, CAV'00, Chicago, IL, USA, in: Lecture Notes in Computer Science, vol. 1855, Springer Verlag, July 2000, pp. 480–494.
[144] Stefan Szeider, Matched formulas and backdoor sets, J. Satisf. Boolean Model. Comput. 6 (2008) 1–12.
[145] Michael Thielscher, Answer set programming for single-player games in general game playing, in: Patricia M. Hill, David S. Warren (Eds.), Proceedings of the 25th International Conference on Logic Programming, ICLP'09, Pasadena, CA, USA, July 14–17, in: Lecture Notes in Computer Science, vol. 5649, Springer Verlag, 2009, pp. 327–341.
[146] Stéphan Thomassé, A quadratic kernel for feedback vertex set, in: Claire Mathieu (Ed.), Proceedings of the 29th Annual ACM–SIAM Symposium on Discrete Algorithms, SODA'09, New York, NY, USA, Society for Industrial and Applied Mathematics (SIAM), January 2009, pp. 115–119.
[147] Son Thanh To, Enrico Pontelli, Tran Cao Son, A conformant planner with explicit disjunctive representation of belief states, in: Alfonso Gerevini, Adele E. Howe, Amedeo Cesta, Ioannis Refanidis (Eds.), Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS'09, Thessaloniki, Greece, The AAAI Press, September 2009, pp. 305–312.
[148] Mirosław Truszczyński, Trichotomy and dichotomy results on the complexity of reasoning with disjunctive logic programs, Theory Pract. Log. Program. 11 (2011) 881–904.
[149] M.H. Van Emden, Robert A. Kowalski, The semantics of predicate logic as a programming language, J. ACM 23 (October 1976) 733–742.
[150] Guido van Rossum, Python tutorial, Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, May 1995.
[151] Vijay Vazirani, Mihalis Yannakakis, Pfaffian orientations, 0/1 permanents, and even cycles in directed graphs, in: Timo Lepistö, Arto Salomaa (Eds.), Proceedings of the 15th International Colloquium on Automata, Languages and Programming, ICALP'88, Tampere, Finland, in: Lecture Notes in Computer Science, vol. 317, Springer Verlag, 1988, pp. 667–681.
[152] Ryan Williams, Carla Gomes, Bart Selman, Backdoors to typical case complexity, in: Georg Gottlob, Toby Walsh (Eds.), Proceedings of the 18th International Joint Conference on Artificial Intelligence, IJCAI'03, Acapulco, Mexico, Morgan Kaufmann, August 2003, pp. 1173–1178.
[153] Ryan Williams, Carla Gomes, 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.
[154] Chee-K. Yap, Some consequences of nonuniform conditions on uniform classes, Theor. Comput. Sci. 26 (3) (1983) 287–300.
[155] Yuting Zhao, Fangzhen Lin, Answer set programming phase transition: a study on randomly generated programs, in: Catuscia Palamidessi (Ed.), Proceedings of the 19th International Conference on Logic Programming, ICLP'03, Mumbai, India, December 9–13, 2003, in: Lecture Notes in Computer Science, vol. 2916, Springer Verlag, 2003, pp. 239–253.
[156] Jicheng Zhao, A study of answer set programming, Mphil thesis, The Hong Kong University of Science and Technology, Dept. of Computer Science, 2002.