Artificial Intelligence 174 (2010) 105–132

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

# Bounded treewidth as a key to tractability of knowledge representation and reasoning ☆

Georg Gottlob [a], Reinhard Pichler [b], Fang Wei [c,*]

[a] Computing Laboratory, Oxford University, Oxford OX1 3QD, UK
[b] Institut für Informationssysteme, Technische Universität Wien, A-1040 Vienna, Austria
[c] Institut für Informatik, Albert-Ludwigs-Universität Freiburg, D-79110 Freiburg i. Br., Germany

ARTICLE INFO

*Article history:*
Received 9 October 2008
Received in revised form 6 October 2009
Accepted 15 October 2009
Available online 20 October 2009

*Keywords:*
Fixed-parameter tractability
Treewidth
Monadic datalog
Abduction
Closed world reasoning
Disjunctive logic programming

ABSTRACT

Several forms of reasoning in AI – like abduction, closed world reasoning, circumscription, and disjunctive logic programming – are well known to be intractable. In fact, many of the relevant problems are on the second or third level of the polynomial hierarchy. In this paper, we show how the notion of treewidth can be fruitfully applied to this area. In particular, we show that all these problems become tractable (actually, even solvable in linear time), if the treewidth of the involved formulae or programs is bounded by some constant.

Clearly, these theoretical tractability results as such do not immediately yield feasible algorithms. However, we have recently established a new method based on monadic datalog which allowed us to design an efficient algorithm for a related problem in the database area. In this work, we exploit the monadic datalog approach to construct new algorithms for logic-based abduction.



## 1. Introduction

In the nineteen-nineties, several forms of reasoning in AI – like abduction, closed world reasoning, circumscription, and disjunctive logic programming – were shown to be highly intractable. In fact, many relevant problems in this area are on the second or even third level of the polynomial hierarchy [20–22].

In recent years, parameterized complexity has evolved as an interesting approach to dealing with intractability [19]. It has turned out that many hard problems become tractable if some problem parameter is fixed or bounded by a fixed constant. If a problem enjoys this property we speak of *fixed-parameter tractability* (FPT, for short). In the arena of graph problems, an important parameter thus investigated is the so-called *treewidth* of a graph $G$, which is a measure of the "tree-likeness" of $G$. If the treewidth of the graphs under consideration is bounded by a fixed constant, then many otherwise intractable problems become tractable, e.g., 3-colorability, Hamiltonicity, etc. More generally, treewidth can be considered for arbitrary finite structures. Treewidth has also been applied to some areas of AI, notably to constraint satisfaction [2].

A deep result and mathematical tool for deriving new FPT-results is Courcelle's Theorem [14], which states that if some property of finite structures is expressible by a Monadic Second Order (MSO, for short) sentence then this property is decidable in linear time for structures whose treewidth is bounded by a fixed constant. Courcelle's Theorem has been further

☆ This is an extended and enhanced version of results published in Gottlob, Pichler and Wei (2006, 2008) [29,31]. The work was supported by the Austrian Science Fund (FWF), project P20704-N18.

* Corresponding author.
*E-mail addresses:* georg.gottlob@comlab.ox.ac.uk (G. Gottlob), pichler@dbai.tuwien.ac.at (R. Pichler), fwei@informatik.uni-freiburg.de (F. Wei).

0004-3702/$ – see front matter 
doi:10.1016/j.artint.2009.10.003

developed in several works, e.g., in the recent paper [24], where also an efficient algorithm for counting the solutions of a SAT or Generalized SAT problem is presented.

In this paper, we revisit several intractable problems in AI. Our goal is to harness the powerful machinery of Courcelle's Theorem in the area of knowledge representation and reasoning. We show that virtually all decision problems arising in the context of abduction, closed world reasoning, circumscription, and disjunctive logic programming become tractable if the treewidth of the involved formulae or programs is bounded by some constant. The central idea for deriving these FPT-results is to encode the decision problems in terms of MSO sentences.

Clearly, an MSO description as such is not an algorithm and it is a highly non-trivial task to turn a theoretical tractability result based on Courcelle's Theorem into a feasible computation. Actually, recipes to devise concrete algorithms based on Courcelle's Theorem can be found in the literature, see e.g. [3,12,11,25]. The basic idea of these algorithms is to transform the MSO evaluation problem into an equivalent tree language recognition problem and to solve the latter via an appropriate finite tree automaton (FTA) [17,46,47]. In theory, this generic method of turning an MSO description into a concrete algorithm looks very appealing. However, in practice, it has turned out that even relatively simple MSO-formulae may lead to a "state explosion" of the FTA [26,41]. Hence, it was already stated in [32] that the algorithms derived via Courcelle's Theorem are "useless for practical applications". The main benefit of Courcelle's Theorem is that it provides "a simple way to recognize a property as being linear time computable". In other words, proving the FPT of some problem by showing that it is MSO expressible is the starting point (rather than the end point) of the search for an efficient algorithm.

In [30], we proposed monadic datalog (i.e., datalog where all intensional predicate symbols are unary) as a practical tool for devising efficient algorithms in situations where the FPT has been established via Courcelle's Theorem. Above all, we proved that if some property of finite structures is expressible in MSO then this property can also be expressed by means of a monadic datalog program over the *decomposed structure*: we mean by this that the original structure is augmented with new elements and new relations that encode one of its tree decompositions. Moreover, this approach was put to work by designing an efficient algorithm for the PRIMALITY problem (i.e., the problem of deciding if some attribute is part of a key in a given relational schema). In this paper, we show that this monadic datalog approach can also be applied to difficult knowledge representation and reasoning tasks. We thus present new algorithms for logic-based abduction and we report on experimental results with an implementation of these algorithms.

### 1.1. Summary of results

The main contribution of our paper is twofold: First, we prove the fixed-parameter tractability of many relevant decision problems arising in abduction, closed world reasoning, circumscription, and disjunctive logic programming. Second, we show how such theoretical tractability results can be actually turned into feasible computations. We only present new algorithms for logic-based abduction. However, the ideas underlying the construction of these algorithms are, of course, also applicable to the hard problems in the other areas mentioned above. Hence, the FPT-results shown here indeed open the grounds for the development of new parameterized algorithms for these problems.

### 1.2. Structure of the paper

The rest of the paper is organized as follows. After recalling some basic definitions and results in Section 2, we prove several new fixed-parameter tractability results via Courcelle's Theorem in Section 3. In Section 4, we present our new algorithms for logic-based abduction. Experimental results are discussed in Section 5. A conclusion is given in Section 6. Appendices A and B are provided in order to encapsulate some lengthy, technical details which are not required for the understanding of the main body of the text.

## 2. Preliminaries

### 2.1. Finite structures and treewidth

Let $\tau = \{R_1, \ldots, R_K\}$ be a set of predicate symbols. A *finite structure* $\mathcal{A}$ over $\tau$ (a *$\tau$-structure*, for short) is given by a finite domain $A = dom(\mathcal{A})$ and relations $R_i^{\mathcal{A}} \subseteq A^{\alpha_i}$, where $\alpha_i$ denotes the arity of $R_i \in \tau$. All structures and trees considered in this work are assumed to be *finite*. Hence, in the sequel, the finiteness will usually not be explicitly mentioned.

A *tree decomposition* $\mathcal{T}$ of a $\tau$-structure $\mathcal{A}$ is defined as a pair $\langle T, (A_t)_{t \in T} \rangle$ where $T$ is a *rooted* tree and each $A_t$ is a subset of $A$ with the following properties: (1) Every $a \in A$ is contained in some $A_t$. (2) For every $R_i \in \tau$ and every tuple $(a_1, \ldots, a_{\alpha_i}) \in R_i^{\mathcal{A}}$, there exists some node $t \in T$ with $\{a_1, \ldots, a_{\alpha_i}\} \subseteq A_t$. (3) For every $a \in A$, the set $\{t \mid a \in A_t\}$ induces a subtree of $T$.

Condition (3) above is usually referred to as the *connectedness condition*. The sets $A_t$ are called the *bags* (or *blocks*) of $\mathcal{T}$. The *width* of a tree decomposition $\langle T, (A_t)_{t \in T} \rangle$ is defined as $max\{|A_t| \mid t \in T\} - 1$. The *treewidth* of $\mathcal{A}$ is the minimal width of all tree decompositions of $\mathcal{A}$. It is denoted as $tw(\mathcal{A})$. Note that forests are the simple loop-free graphs of treewidth at most 1.

For given $w \geqslant 1$, it can be decided in linear time if some structure has treewidth at most $w$. Moreover, in case of a positive answer, a tree decomposition of width $w$ can be computed in linear time [8]. Strictly speaking, the result in [8]

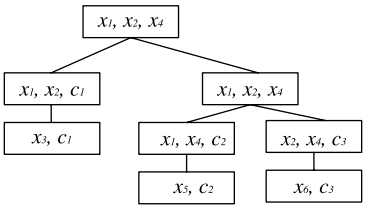


**Fig. 1.** Tree decomposition $\mathcal{T}$ of formula $F$.

refers to tree decompositions of *graphs* rather than arbitrary *structures*. However, we can associate a graph $\mathcal{G}$ (the so-called primal or Gaifman graph) with every structure $\mathcal{A}$ by taking the domain elements as the vertices of the graph. Moreover, two vertices are adjacent in $\mathcal{G}$ if and only if the corresponding domain elements jointly occur in some tuple in $\mathcal{A}$. It can be easily shown that $\mathcal{G}$ has precisely the same tree decompositions as $\mathcal{A}$.

Unfortunately, it has been shown that this linear time algorithm is mainly of theoretical interest and its practical usefulness is limited [39]. Recently, considerable progress has been made in developing heuristic-based tree decomposition algorithms which can handle graphs with moderate size of several hundreds of vertices [39,9,49,10]. Moreover, in some cases, a tree decomposition of low width may be obtained from a given problem in a "natural way". For instance, in [48], it was shown that the tree-width of the control-flow graph of any goto-free C program is at most six. A similar result was shown for Java programs in [33]. These results opened the ground for the efficient implementation of various compiler optimization tasks like the register allocation problem.

In this paper, we shall constantly have to deal with propositional formulae in CNF or, analogously, with clause sets. A propositional formula $F$ in CNF (respectively a clause set $F$) can be represented as a structure $\mathcal{A}$ over the alphabet $\tau = \{cl(.), var(.), pos(.,.), neg(.,.)\}$ where $cl(z)$ (respectively $var(z)$) means that $z$ is a clause (respectively a variable) in $F$ and $pos(x, c)$ (respectively $neg(x, c)$) means that $x$ occurs unnegated (respectively negated) in the clause $c$. We define the treewidth of $F$ as the treewidth of this structure $\mathcal{A}$, i.e., $tw(F) = tw(\mathcal{A})$.

**Example 2.1.** Consider the propositional formula $F$ with

$$F = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_4 \vee \neg x_5) \wedge (x_2 \vee \neg x_4 \vee x_6).$$

The corresponding structure $\mathcal{A}$ can be identified with the following set of ground atoms

$$\begin{aligned}\mathcal{A} = \big\{ & var(x_1), var(x_2), var(x_3), var(x_4), var(x_5), var(x_6),\\ & cl(c_1), cl(c_2), cl(c_3),\\ & pos(x_1, c_1), pos(x_3, c_1), pos(x_4, c_2), pos(x_2, c_3), pos(x_6, c_3),\\ & neg(x_2, c_1), neg(x_1, c_2), neg(x_5, c_2), neg(x_4, c_3)\big\}.\end{aligned}$$

A tree decomposition $\mathcal{T}$ of this structure is given in Fig. 1. Note that the maximal size of the bags in $\mathcal{T}$ is 3. Hence, the tree-width is at most 2. On the other hand, it is easy to check that the tree-width of $\mathcal{T}$ cannot be smaller than 2. In order to see this, we consider the ground atoms $pos(x_1, c_1)$, $neg(x_2, c_1)$ $pos(x_2, c_3)$, $neg(x_4, c_3)$ $pos(x_4, c_2)$, and $neg(x_1, c_2)$ in $\mathcal{A}$ as (undirected) edges of a graph. Clearly, these edges form a cycle. However, as we have recalled above, only forests are the simple loop-free graphs of treewidth at most 1. This tree decomposition is, therefore, optimal and we have $tw(F) = tw(\mathcal{A}) = 2$.

In [30], it was shown that any tree decomposition can be transformed into the following *normal form* in linear time:

**Definition 2.2.** Let $\mathcal{A}$ be a structure with tree decomposition $\mathcal{T} = \langle T, (A_t)_{t \in T} \rangle$ of width $w$. We call $\mathcal{T}$ *normalized* if $T$ is a rooted tree and conditions 1–4 are fulfilled: (1) All bags consist of pairwise distinct elements $a_0, \ldots, a_k$ with $0 \leqslant k \leqslant w$. (2) Every internal node $t \in T$ has either one or two child nodes. (3) If a node $t$ has one child node $t'$, then the bag $A_t$ is obtained from $A_{t'}$ either by removing one element or by introducing a new element. (4) If a node $t$ has two child nodes then these child nodes have identical bags as $t$.

**Example 2.3.** Recall the tree decomposition $\mathcal{T}$ from Fig. 1. Clearly, $\mathcal{T}$ is not normalized in the above sense. However, in can be easily transformed into a normalized tree decomposition $\mathcal{T}'$, see Fig. 2.

Let $\mathcal{A}$ be a $\tau$-structure with $\tau = \{R_1, \ldots, R_K\}$ and domain $A$ and let $w \geqslant 1$ denote the treewidth. Then we define the extended signature $\tau_{td}$ as

$$\tau_{td} = \tau \cup \{root, leaf, child_1, child_2, bag\}$$

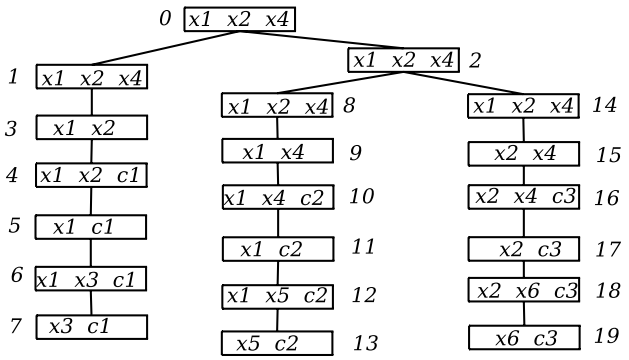


**Fig. 2.** Normalized tree decomposition $\mathcal{T}'$ of formula $F$.

where the unary predicates *root* and *leaf* as well as the binary predicates $child_1$ and $child_2$ are used to represent the tree $T$ of the (normalized) tree decomposition in the obvious way. For instance, we write $child_1(s_1, s)$ to denote that $s_1$ is either the first child or the only child of $s$. Finally, *bag* has arity $k+2$ with $k \leqslant w$, where $bag(t, a_0, \ldots, a_k)$ means that the bag at node $t$ is $(a_0, \ldots, a_k)$. By slight abuse of notation, we tacitly assume that *bag* is overloaded for various values of $k$. Note that the possible values of $k$ are bounded by a fixed constant $w$. For any $\tau$-structure $\mathcal{A}$ with tree decomposition $\mathcal{T} = \langle T, (A_t)_{t \in T} \rangle$ of width $w$, we write $\mathcal{A}_{td}$ to denote the *decomposed structure* or the *structure decomposing* $\mathcal{A}$. This structure is obtained in the following way: The domain of $\mathcal{A}_{td}$ is the union of $dom(\mathcal{A})$ and the nodes of $T$. In addition to the relations $R_i^{\mathcal{A}}$ with $R_i \in \tau$, the structure $\mathcal{A}_{td}$ also contains relations for each predicate *root*, *leaf*, $child_1$, $child_2$, and *bag* thus representing the tree decomposition $\mathcal{T}$. By combining results from [8] and [30], one can compute $\mathcal{A}_{td}$ from $\mathcal{A}$ in linear time w.r.t. the size of $\mathcal{A}$.

**Example 2.4.** Recall the propositional formula $F = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_4 \vee \neg x_5) \wedge (x_2 \vee \neg x_4 \vee x_6)$ from Example 2.1. We have already shown how $F$ can be represented as a structure $\mathcal{A}$ over the signature $\tau = \{cl, var, pos, neg\}$. In order to represent also the tree decomposition $\mathcal{T}'$ from Fig. 2 by the structure $\mathcal{A}_{td}$, we have to add the following ground atoms to $\mathcal{A}$: $\{root(t_1), child_1(t_2, t_1), child_2(t_3, t_1), child_1(t_4, t_2), \ldots, leaf(t_{16}), leaf(t_{19}), leaf(t_{20}), bag(t_1, x_1, x_2, x_4), bag(t_2, x_1, x_2, x_4), bag(t_3, x_1, x_2, x_4), bag(t_4, x_1, x_2), \ldots\}$. (The numbering of the nodes $t_1, t_2, t_3, \ldots$ corresponds to a breadth-first traversal of $\mathcal{T}'$.)

### 2.2. MSO and monadic datalog

*Monadic Second Order* logic (MSO) extends First Order logic (FO) by the use of *set variables* (usually denoted by upper case letters), which range over sets of domain elements. In contrast, the *individual variables* (which are usually denoted by lower case letters) range over single domain elements. Atomic formulae in an MSO-formula $\varphi$ over a $\tau$-structure have one of the following forms: (i) atoms with some predicate symbol from $\tau$, (ii) atoms whose predicate symbol is a monadic second order variable (i.e., a set variable), or (iii) equality atoms. It is convenient to use symbols like $\subseteq$, $\subset$, $\cap$, $\cup$, and $\rightarrow$ with the obvious meaning as abbreviations. An MSO-formula $\varphi(x)$ with exactly one free individual variable is called a *unary query*. The importance of MSO-formulae in the context of parameterized complexity comes from the following result:

**Theorem 2.5.** *(See [14].) Let $\varphi$ be an MSO-sentence over some signature $\tau$ and let $\mathcal{A}$ be a $\tau$-structure of treewidth $w$. Evaluating the sentence $\varphi$ over the structure $\mathcal{A}$ can be done in time $\mathcal{O}(f(|\varphi(x)|, w) * |\mathcal{A}|)$ for some function $f$.*

In the sequel, we shall simply say that structures "have bounded treewidth" without explicitly mentioning $w$. Note that the fixed-parameter linearity according to Theorem 2.5 only applies to the *data complexity*, i.e. the formula $\varphi$ is fixed. There is no such FPT-result, if we consider the *combined complexity* instead (i.e. also $\varphi$ is part of the input).

*Datalog* programs are function-free logic programs. Each of them has a unique *minimal model*. Evaluating a datalog program $\mathcal{P}$ over a structure $\mathcal{A}$ comes down to computing the minimal model of $\mathcal{P} \wedge \mathcal{A}$. This minimal model coincides with the set of (ground) facts obtained as the least fixpoint of the immediate consequence operator, for details see e.g. [1]. In the sequel, we shall refer to a program $\mathcal{P}$ together with a structure $\mathcal{A}$ as an *interpreted program*. A datalog program is called *ground* if it contains no variables. Evaluating a datalog program $\mathcal{P}$ over a structure $\mathcal{A}$ is equivalent to evaluating the following ground program $\mathcal{P}'$ over $\mathcal{A}$: For every clause $r$ in $\mathcal{P}$, $\mathcal{P}'$ contains all possible instances of $r$ that we get by instantiating every variable in $r$ by a constant occurring either in $\mathcal{P}$ or in $\mathcal{A}$ (the set of all these constants is referred to as the active domain). In general, $\mathcal{P}'$ is exponentially bigger than $\mathcal{P}$. However, in some cases, only a small fraction of the set of possible ground instances of every rule $r$ is necessary as we shall see, e.g., in the proof of Theorem 4.2.

Predicates occurring only in the body of rules are called extensional, while predicates occurring also in the head of some rule are called intensional.

Let $\mathcal{A}$ be a $\tau$-structure with domain $A$ and relations $R_1^{\mathcal{A}}, \ldots, R_K^{\mathcal{A}}$ with $R_i^{\mathcal{A}} \subseteq A^{\alpha_i}$, where $\alpha_i$ denotes the arity of $R_i \in \tau$. In the context of datalog, it is convenient to think of the relations $R_i^{\mathcal{A}}$ as sets of ground atoms. The set of all such ground atoms of a structure $\mathcal{A}$ is referred to as the extensional database (EDB) of $\mathcal{A}$, i.e.: a ground atom $R_i(\bar{a})$ is in the EDB if and only if $\bar{a} \in R_i^{\mathcal{A}}$.

In [30], the following connection between unary MSO queries over structures with bounded treewidth and monadic datalog was established:

**Theorem 2.6.** *Let $\tau$ be a signature and let $w \geqslant 1$. Every unary MSO-query $\varphi(x)$ over $\tau$-structures of treewidth $w$ is also definable by a monadic datalog program over $\tau_{td}$.*

*Moreover, for every $\tau$-structure $\mathcal{A}$, the resulting program can be evaluated in time $\mathcal{O}(f(|\varphi(x)|, w) * |\mathcal{A}|)$ for some function $f$, where $|\mathcal{A}|$ denotes the size of $\mathcal{A}$.*

Note that the connection between logic programming (with functions) and tree automata has already been studied much earlier, see e.g. [40,23]. As far as the upper bound on the complexity is concerned, the above theorem is a special case of Theorem 4.12 in [25].

## 3. Fixed-parameter tractability via Courcelle's Theorem

In this section, we show the fixed-parameter tractability w.r.t. the treewidth for many decision problems in the area of disjunctive logic programming, closed world reasoning, circumscription, and abduction. Fundamental to these problems is the evaluation of a propositional formula in an interpretation. We therefore start our exposition with the SAT problem, whose fixed-parameter tractability for various parameters (including cliquewidth and treewidth) was already shown in [15].

### 3.1. SAT problem

A *propositional formula* $F$ is built up from propositional variables denoted as $Var(F)$ and the logical connectives $\vee$, $\wedge$, and $\neg$. An *interpretation* of $F$ is a mapping that assigns one of the truth values true or false to each variable occurring in $F$. It is convenient to identify interpretations with subsets $X$ of $Var(F)$ with the intended meaning that the variables in $X$ evaluate to true, while all other variables evaluate to false. If $F$ evaluates to true in $X$, then $X$ is called a *model* of $F$. Moreover, $X$ is called a *minimal model*, if there exists no model $X'$ of $F$ with $X' \subset X$. We write $F_1 \models F_2$ if the formula $F_1 \rightarrow F_2$ is valid, i.e., $F_1 \rightarrow F_2$ is true in every interpretation $X \subseteq Var(F_1) \cup Var(F_2)$. Likewise, if $\mathcal{F} = \{F_1, \ldots, F_m\}$ is a set of formulae and $F$ a single formula, we write $\mathcal{F} \models F$ if the formula $F_1 \wedge \cdots \wedge F_m \rightarrow F$ is valid. The following result is folklore [5,44].

**Theorem 3.1.** *For every propositional formula $F$, there exists a formula $F'$ in CNF, with $Var(F) \subseteq Var(F')$, s.t. the following properties hold*:

(1) *For every interpretation $X$, s.t. $X$ is a model of $F$, there exists an interpretation $Y$ with $X \subseteq Y$ and $(Y \setminus X) \subseteq (Var(F') \setminus Var(F))$, s.t. $Y$ is a model of $F'$.*
(2) *For every interpretation $Y$, s.t. $Y$ is a model of $F'$, the interpretation $Y \cap Var(F)$ is a model of $F$.*

*Moreover, such an $F'$ can always be found in linear time and, thus, also the size of $F'$ is linearly bounded by the size of $F$.*

**Proof.** Given a propositional formula $F$ with variables in $V$, we first compute the parse tree $T$ of $F$. This is clearly feasible in linear time. This parse tree can be considered as a Boolean circuit where each node in the tree corresponds to a gate, the leaf nodes (labeled with variables) are the input gates, the root node is the output gate and each edge of the parse tree is oriented from the child to its parent. The construction of a CNF follows the well-known reduction from the CIRCUIT SAT problem to the SAT problem, see e.g. [44], Example 8.3:

In addition to the variables $V$, our CNF $F'$ contains one additional variable for each internal node $g$ of the parse tree $T$. For leaf nodes, we may simply use the corresponding propositional variable $x$ as the name of this node. The clauses of $F'$ are obtained for each internal node $g$ of $T$ by a case distinction over the possible operations represented by this node:

(1) Suppose that $g$ corresponds to a NOT-operation. Then $g$ has precisely one incoming arc, say from node $h$. Then we add the two clauses $(\neg g \vee \neg h)$ and $(g \vee h)$ to $F'$. That is, these clauses express the condition $g \leftrightarrow (\neg h)$.

(2) Suppose that $g$ corresponds to an AND-operation. Then $g$ has two incoming arcs, say from nodes $h_1$ and $h_2$. Then we add the three clauses $(\neg g \vee h_1)$, $(\neg g \vee h_2)$, and $(\neg h_1 \vee \neg h_2 \vee g)$ to $F'$. That is, these clauses express the condition $g \leftrightarrow (h_1 \wedge h_2)$.

(3) Suppose that $g$ corresponds to an OR-operation. Then $g$ has two incoming arcs, say from nodes $h_1$ and $h_2$. Then we add the three clauses $(g \vee \neg h_1)$, $(g \vee \neg h_2)$, and $(h_1 \vee h_2 \vee \neg g)$ to $F'$. That is, these clauses express the condition $g \leftrightarrow (h_1 \wedge h_2)$.

Finally, for the root node $r$ of $T$, we also add the one-literal clause $r$ to $F'$. Clearly, $F'$ is in CNF. The desired properties of $F'$ (in particular, the sat-equivalence with $F$) are easily verified. □

Note that the CNF $F'$ constructed in the proof of Theorem 3.1 is uniquely determined (up to variable renaming) by the parse tree of $F$ and hence by the form of $F$ itself. We shall refer to $F'$ as the *canonical CNF* of $F$. It can be represented by a $\tau$-structure $\mathcal{A}(F)$ with $\tau = \{var(.),\ cl'(.),\ var'(.),\ pos'(.,.), neg'(.,.)\}$, s.t. $var(.)$ is used to identify the variables occurring in $F$ and $cl'(.)$, $var'(.)$, $pos'(.,.), neg'(.,.)$ are used to represent the CNF $F'$ in the usual way. In Section 2.1, we have defined the treewidth of a propositional formula $F$ in CNF (or, analogously, of a clause set) as the treewidth of the structure representing $F$. Likewise, we can define the treewidth of an arbitrary propositional formula $F$ as the treewidth of the canonical CNF $F'$ of $F$ or, equivalently, as the treewidth of the $\tau$-structure $\mathcal{A}(F)$.

The notion of treewidth can be naturally extended to more than one formula in CNF, e.g., let $F_1$ and $F_2$ be two propositional formulae in CNF. Then $(F_1, F_2)$ can be represented by a $\tau$-structure $\mathcal{A}(F_1, F_2)$ with $\tau = \{cl_i(.), var_i(.), pos_i(.,.), neg_i(.,.) \mid 1 \leqslant i \leqslant 2\}$. We define the treewidth of $F_1$ and $F_2$ as the treewidth of the structure $\mathcal{A}$. We thus have $tw(F_1, F_2) = tw(\mathcal{A}(F_1, F_2)) = tw(F_1 \wedge F_2)$. If $F_1$ and $F_2$ are not in CNF, then we can represent these formulae by a structure $\mathcal{A}$ over the signature $\tau = \{var_i(.), cl'_i(.), var'_i(.), pos'_i(.,.), neg'_i(.,.) \mid 1 \leqslant i \leqslant 2\}$, where $var_i(.)$ is used to identify the variables occurring in $F_i$ and $cl'_i(.)$, $var'_i(.)$, $pos'_i(.,.), neg'_i(.,.)$ are used to represent the CNF $F'_i$ in the usual way. Again, the treewidth of $F_1$ and $F_2$ is defined as the treewidth of the structure $\mathcal{A}$.

From results in [15], the following relationship between CNF-formulae and MSO can be easily derived:

**Theorem 3.2.** *Let $F$ be a propositional formula in CNF which is given by a $\tau$-structure $\mathcal{A}$ with $\tau = \{cl, var, pos, neg\}$. Moreover, let $X$ denote a set of the variables with $X \subseteq Var(F)$. Then the property that $X$ is a model of $F$ can be expressed by an MSO-formula – referred to as $model(X, F)$ – over the signature $\tau$.*

**Proof.** We define the MSO-formula $model(X, F)$ as follows:

$$model(X, F) := (\forall c)\, cl(c) \rightarrow (\exists z)\big[\big(pos(z, c) \wedge z \in X\big) \vee \big(neg(z, c) \wedge z \notin X\big)\big].$$

This MSO-formula expresses the following equivalence: $F$ is true in $X$ if and only if every clause of $F$ is true in $X$. This in turn is the case if and only if every clause contains a positive occurrence of a variable $z \in X$ or a negative occurrence of a variable $z \notin X$. □

Clearly, Theorem 3.2 together with Courcelle's Theorem immediately yields the fixed-parameter tractability of the SAT problem w.r.t. the treewidth of the $\tau$-structure $\mathcal{A}$ (with $\tau = \{cl, var, pos, neg\}$) representing the propositional formula [15].

Actually, even if $F$ is not in CNF, the property that $X$ is a model of $F$ can be expressed in terms of MSO:

**Theorem 3.3.** *Let $F$ be a propositional formula with canonical CNF $F'$ and let $F$ be given by a $\tau$-structure $\mathcal{A}$ with $\tau = \{var, cl', var', pos', neg'\}$. Moreover, let $X$ denote a set of the variables with $X \subseteq Var(F)$. Then the property that $X$ is a model of $F$ can be expressed by an MSO-formula – referred to as $model'(X, F)$ – over the signature $\tau$.*

**Proof.** We define the MSO-formula $model'(X, F)$ as follows:

$$Ext_F(X, X') := X \subseteq X' \wedge (\forall z)\big[(z \in X' \wedge z \notin X) \rightarrow \big(var'(z) \wedge \neg var(z)\big)\big],$$
$$model'(X, F) := (\exists X')\big[Ext_F(X, X') \wedge \big(model(X', F')\big)\big].$$

The auxiliary formula $Ext_F(X, X')$ means that $X'$ is an extension of the interpretation $X$ to the variables in $F'$. Moreover, the subformula $model(X', F')$ is precisely the CNF-evaluation from Theorem 3.2. □

Finally, we also provide an MSO-formula expressing that $F_1 \models F_2$ holds for two propositional formulae $F_1$ and $F_2$.

**Theorem 3.4.** *Let $F_1$, $F_2$ be propositional formulae with canonical CNFs $F'_1$, $F'_2$ and let $F_1$, $F_2$ be given by a $\tau$-structure $\mathcal{A}$ with $\tau = \{var_i(.), cl'_i(.), var'_i(.), pos'_i(.,.), neg'_i(.,.) \mid 1 \leqslant i \leqslant 2\}$. Then the property that $F_1 \models F_2$ holds can be expressed by means of an MSO-formula – referred to as $implies(F_1, F_2)$ – over the signature $\tau$.*

**Proof.** We define the MSO-formula $implies(F_1, F_2)$ as follows:

$$(\forall X)\big[model'(X, F_1) \rightarrow \big(model'(X, F_2)\big)\big].$$

The subformulae $model'(X, F_i)$ with $i \in \{1, 2\}$ are the MSO-formulae expressing the evaluation of propositional formulae $F_i$ according to Theorem 3.3. □

### 3.2. Disjunctive logic programming

A *disjunctive logic program* (DLP, for short) $P$ is a set of DLP clauses $a_1 \vee \cdots \vee a_n \leftarrow b_1, \ldots, b_k, \neg b_{k+1}, \ldots, \neg b_m$. Let $I$ be an interpretation. Then the *Gelfond–Lifschitz reduct* $P^I$ of P w.r.t. $I$ contains precisely the clauses $a_1 \vee \cdots \vee a_n \leftarrow b_1, \ldots, b_k$,

s.t. for all $i \in \{k+1, \ldots, m\}$, $b_i \notin I$. An interpretation $I$ is called a *disjunctive stable model* (*DSM*, for short) if and only if $I$ is a minimal model of $P^I$ [27,45].

Without any restrictions, the following problems are all on the second level of the polynomial hierarchy [22]:

- Consistency: Does $P$ have a *DSM*?
- Brave Reasoning: Is a propositional formula $F$ true in at least one *DSM* of $P$ (written as $P \models_b F$)?
- Cautious Reasoning: Is a propositional formula $F$ true in all *DSM*s of $P$ (written as $P \models_c F$)?

The intuition of stable models is as follows: Consider the reduct of a program $P$ with respect to some interpretation $I$. We can think of $I$ as making an assumption about what is true and what is false. Consequently, we may delete those rules from the program whose body is definitely false under this assumption, thus arriving at the reduct $P^I$. Now we can compute the positive information derivable from $P^I$. If the result is $I$ itself, then $I$ is called a stable model.

Negation under the stable model semantics has received a lot of interest since it significantly increases the expressive power of logic programs. For instance, even without making use of disjunctions in the rule heads, we can express the classical NP-complete 3-colorability problem by the following program: Let $G = (V, E)$ be a graph with vertices $V = \{v_1, \ldots, v_n\}$ and edges $E$. Then $G$ can be represented by a logic program consisting of the following facts:

$$\{e_{i,j} \mid 1 \leqslant i, j \leqslant n \text{ and there exists an edge between } v_i \text{ and } v_j\}.$$

The 3-colorability of $G$ is expressed by adding the following clauses to the program (note that the commas on the right-hand side of the rules stand for logical "and"):

$$\{r_i \leftarrow \neg g_i, \neg b_i;\ g_i \leftarrow \neg r_i, \neg b_i;\ b_i \leftarrow \neg r_i, \neg g_i \mid 1 \leqslant i \leqslant n\},$$

$$\{\ \leftarrow e_{i,j}, r_i, r_j;\ \leftarrow e_{i,j}, g_i, g_j;\ \leftarrow e_{i,j}, b_i, b_j \mid 1 \leqslant i, j \leqslant n\}.$$

The stable models of this program correspond to the proper 3-colorings of the graph $G$. Intuitively, the first collection of clauses makes sure that, in every stable model, exactly one of the variables $r_i, g_i, b_i$ is true for each $i \in \{1, \ldots, n\}$. The second collection of clauses excludes those "colorings" which would assign identical colors to the endpoints of some edge. In particular, the question if a given graph has at least one proper 3-coloring is equivalent to the consistency problem of the above described program, i.e., does the program have at least one stable model?

Suppose that we want to ask if *there exists at least one proper* 3*-coloring* in which the vertices $v_1, v_2$, and $v_3$ have the same color. This question corresponds to the brave reasoning problem $P \models_b F$, where $P$ is the above program and $F = (r_1 \wedge r_2 \wedge r_3) \vee (g_1 \wedge g_2 \wedge g_3) \vee (b_1 \wedge b_2 \wedge b_3)$. Likewise, the question if $v_1, v_2$, and $v_3$ have the same color *in every proper* 3*-coloring* corresponds to the cautious reasoning problem $P \models_c F$ with $P$ and $F$ as before. Many more examples of (disjunctive) logic programs with negation for solving problems in diverse application areas can be found at http://www.kr.tuwien.ac.at/research/projects/WASP/.

Below, we show that, if the treewidth of the programs $P$ and formulae $F$ under consideration is bounded by a constant, then we get much more favorable complexity results than in the general case. Suppose that a DLP $P$ is given by a $\tau$-structure with $\tau = \{var_P(.), cl_P(.), H(.,.), B^+(.,.), B^-(.,.)\}$, s.t. $var_P(.)$ and $cl_P(.)$ encode the variables and clauses of $P$. Moreover, $H(x, c)$ means that $x$ occurs in the head of $c$ and $B^+(x, c)$ (respectively $B^-(x, c)$) means that $x$ occurs unnegated (respectively negated) in the body of $c$. Then we have:

**Theorem 3.5.** *Consider the signatures* $\tau_1 = \{var_P, cl_P, H, B^+, B^-\}$ *as above and* $\tau_2 = \{var, cl', var', pos', neg'\}$ *as in Theorem* 3.3. *The* Consistency *problem of DLPs can be expressed by means of an MSO-sentence over the signature* $\tau_1$. *Likewise, the* Brave Reasoning *and* Cautious Reasoning *problem of DLPs can be expressed by means of MSO-sentences over the signature* $\tau_1 \cup \tau_2$.

**Proof.** Recall from Theorem 3.3 that the property that $X$ is a model of $F$ can be expressed by the MSO-formula $model'(X, F)$ over the signature $\tau_2$. We thus have:

$$GL(X, Y) := (\forall c) cl_P(c) \rightarrow (\exists z)\big[\big(H(z, c) \wedge z \in X\big) \vee \big(B^+(z, c) \wedge z \notin X\big) \vee \big(B^-(z, c) \wedge z \in Y\big)\big],$$

$$DSM(X) := GL(X, X) \wedge (\forall Z)\big[Z \subset X \rightarrow \neg GL(Z, X)\big],$$

Consistency: $(\exists X) DSM(X)$,

Brave Reasoning: $(\exists X)\big[DSM(X) \wedge model'(X, F)\big]$,

Cautious Reasoning: $(\forall X)\big[DSM(X) \rightarrow model'(X, F)\big]$.

The predicates defined above have the following meaning:

$GL(X, Y)$ = "$X$ is a model of the Gelfond–Lifschitz reduct of the program $P$ w.r.t. the interpretation $Y$".

$DSM(X)$ = "$X$ is a disjunctive stable model of $P$". □

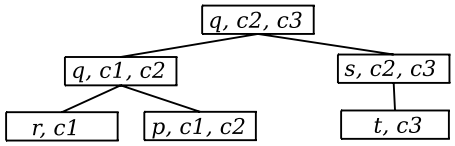


**Fig. 3.** Tree decomposition of structure $\mathcal{A}(P)$ in Example 3.6.

**Example 3.6.** Consider the following DLP:

$$P = \{c_1 : p \vee q \leftarrow \neg r,\ c_2 : q \leftarrow \neg p \wedge \neg s,\ c_3 : s \vee t \leftarrow q\}.$$

Clearly, $P$ is consistent since, for instance, $\{p\}$ is a *DSM*.

The DLP $P$ can be represented by the structure $\mathcal{A}(P)$, which consists of the following ground atoms:

$$\begin{aligned}\mathcal{A}(P) = \big\{ & var_P(p), var_P(q), var_P(r), var_P(s), var_P(t),\\ & cl_P(c_1), cl_P(c_2), cl_P(c_3),\\ & H(p, c_1), H(q, c_1), H(q, c_2), H(s, c_3), H(t, c_3),\\ & B^-(r, c_1), B^-(p, c_2), B^-(s, c_2), B^+(q, c_3)\big\}.\end{aligned}$$

A tree decomposition $\mathcal{T}$ of $\mathcal{A}(P)$ is given in Fig. 3. The MSO-formula $GL(X, Y)$ from the proof of Theorem 3.5 clearly evaluates to true over $\mathcal{A}(P)$ for $X = \{p\}$ and $Y = \{p\}$. Moreover, for $X = \{\}$ and $Y = \{p\}$, it evaluates to false. Hence, $DSM(X)$ evaluates to true for $X = \{p\}$ and, therefore, the consistency of $P$ is correctly established via the MSO-formula $(\exists X)DSM(X)$.

The following complexity result is an immediate consequence of Courcelle's Theorem and Theorem 3.5.

**Theorem 3.7.** *Consider the signatures $\tau_1 = \{var_P, cl_P, H, B^+, B^-\}$ as above and $\tau_2 = \{var, cl', var', pos', neg'\}$ as in Theorem* 3.5. *Let an instance of the* Consistency *problem of DLPs be given by a $\tau_1$-structure $\mathcal{A}$ of width $w$. Then this problem can be solved in time $\mathcal{O}(f(w) * |\mathcal{A}|)$ for some function $f$.*

*Likewise, let an instance of* Brave Reasoning *problem or the* Cautious Reasoning *of DLPs be given by a $(\tau_1 \cup \tau_2)$-structure $\mathcal{A}$ of width $w$. Then this problem can be solved in time $\mathcal{O}(f'(w) * |\mathcal{A}|)$ for some function $f'$.*

### 3.3. Closed world reasoning and circumscription

Several forms of *closed world reasoning* (CWR, for short) are proposed in the literature, namely CWA (Closed World Assumption), GCWA (Generalized CWA), EGCWA (Extended GCWA), CCWA (Careful CWA), and ECWA (Extended CWA). They are defined in terms of the following terminology: Let $T$ (a "theory") and $F$ be propositional formulae and let $\langle P; Q; Z\rangle$ be a partition of $Var(T)$. Then we write $M(T)$ (respectively $MM(T)$) to denote the set of all models (respectively of all minimal models) of $T$. Moreover, we write $MM(T; P; Q; Z)$ to denote the set of $\langle P; Q; Z\rangle$-minimal models of $T$, i.e.: $X \in MM(T; P; Q; Z)$ if and only if $X$ is a model of $T$ and there exists no model $Y$ of $T$ with $(Y \cap P) \subset (X \cap P)$ and $(Y \cap Q) = (X \cap Q)$.

In [13], several equivalent characterizations of the closure of a theory $T$ under the various CWR rules are provided. Below, we recall those characterizations which are best suited for our purposes here:

- $\mathrm{CWA}(T) = T \cup \{\neg K \mid K$ is a positive literal s.t. $T \not\models K\}$.
- $\mathrm{GCWA}(T) = T \cup \{\neg K \mid K$ is a positive literal and for every $X \in MM(T)$, $K$ is false in $X\}$.
- $\mathrm{EGCWA}(T) \models F$ if and only if for every $X \in MM(T)$, $X$ is a model of $F$.
- $\mathrm{CCWA}(T; P; Q; Z) = T \cup \{\neg K \mid K$ is a positive literal from $P$ and for every $X \in MM(T; P; Q; Z)$, $K$ is false in $X\}$.
- $\mathrm{ECWA}(T; P; Q; Z) \models F$ if and only if, for every $X \in MM(T; P; Q; Z)$, $X$ is a model of $F$.

The Deduction problem of CWR-rule C with $\mathrm{C} \in \{\mathrm{CWA}, \mathrm{GCWA}, \mathrm{EGCWA}, \mathrm{CCWA}, \mathrm{ECWA}\}$ is as follows: Given $T$ and $F$ (and possibly $P, Q, Z$), does $\mathrm{C}(T) \models F$ (respectively $\mathrm{C}(T; P; Q; Z) \models F$) hold? In [20], this problem is shown to be $\Pi_2^p$-complete or even harder for all rules $\mathrm{C} \neq \mathrm{CWA}$. In [28] it was shown that, in the propositional case, Circumscription coincides with the ECWA-rule.

The above formalisms of closed world reasoning provide a formal basis of various kinds of inferences of facts which are not explicitly specified in a knowledge base. They give rise to non-monotonic reasoning in the sense that new information may invalidate previously derived conclusions. In [13], the classical Tweety example is used to highlight some differences between the above formalisms. Consider the formula, which represents the propositional version of the Tweety example:

$$\big((b \wedge \neg a) \rightarrow f\big) \wedge b.$$

The intended meaning is as follows: If Tweety is a bird (i.e., $b$) and Tweety is not abnormal (i.e., $\neg a$), then Tweety can fly (i.e., $f$). Moreover, we know that Tweety is a bird. We get the following closure under the various CWR-formalisms. For

the CCWA- and ECWA-rule, we assume $P = \{a\}$ (i.e., $P$ should be minimized), $Q = \{b\}$ (i.e., $Q$ is fixed), and $Z = \{f\}$ (i.e., $Z$ varies).

- $\mathrm{CWA}(T) = T \cup \{\neg a, \neg f\}$.
- $\mathrm{GCWA}(T) = T$.
- $\mathrm{EGCWA}(T) = T \cup \{\neg(a \wedge f)\}$.
- $\mathrm{CCWA}(T; P; Q; Z) = T \cup \{\neg a\}$.
- $\mathrm{ECWA}(T; P; Q; Z) = T \cup \{\neg a\}$.

The CWA-rule is the only rule which may lead to inconsistency. This is indeed the case in the above example. Only the CCWA- and ECWA-rule allow us to derive the atom $f$, i.e., Tweety can fly. By the EGCWA-rule, the formula $\neg(a \wedge f)$ may be inferred, i.e., Tweety is not abnormal or Tweety cannot fly.

Again, for bounded treewidth, we get much better complexity results than in the general case. Let $T$ and $F$ be propositional formulae with canonical CNFs $T'$ and $F'$, respectively. Moreover, suppose that the Deduction problem of CWR-rule C with $\mathsf{C} \in \{\mathrm{CWA}, \mathrm{GCWA}, \mathrm{EGCWA}, \mathrm{CCWA}, \mathrm{ECWA}\}$ is given by a $\tau$-structure with $\tau = \{var_T(.), var_F(.), var_{T'}(.), var_{F'}(.), cl_{T'}(.), cl_{F'}(.), \ldots\} \cup \{P, Q, Z\}$. The predicates $var_T(.)$ and $var_F(.)$ are used to identify the variables in the original formulae $T$ and $F$. The CNFs are encoded as usual by means of the predicates $var_{T'}(.)$, $var_{F'}(.)$, $cl_{T'}(.)$, $cl_{F'}(.)$, etc. Finally, the partition $\langle P; Q; Z\rangle$ of $Var(T)$ is encoded by the unary predicates $P, Q, Z$. Of course, for the CWR-rules CWA, GCWA, and EGCWA, the predicates $P, Q, Z$ may be omitted. Then we have:

**Theorem 3.8.** *Consider the signature $\tau = \{var_T, var_F, var_{T'}, var_{F'}, cl_{T'}, cl_{F'}, pos_{T'}, pos_{F'}, neg'_{T',F'}, P, Q, Z\}$ as described above. For all of the CWR-rules CWA, GCWA, EGCWA, CCWA, and ECWA, the* Deduction *problem (and, hence, also* Circumscription*) can be expressed by MSO sentences over the signature $\tau$ (of course, for CWA, GCWA, EGCWA, the predicates $P$, $Q$, and $Z$ are not needed).*

**Proof.** GCWA and EGCWA are special cases of CCWA and ECWA with $Q = Z = \emptyset$. Moreover, as mentioned above, Circumscription is equivalent to the ECWA-rule. The remaining cases are expressed by the following MSO-sentences, which make use of the formulae $model'(X, F)$ and $implies(F_1, F_2)$ according to Theorems 3.3 and 3.4, respectively:

$$MM(X) := model'(X, T) \wedge \neg(\exists Y)\big[(Y \cap P) \subset (X \cap P) \wedge (Y \cap Q) = (X \cap Q) \wedge model'(Y, T)\big],$$

$$var(z) := var_T(z) \vee var_F(z),$$

$$clo_1(\neg z) := var(z) \wedge \neg implies(T, z),$$

$$clo_2(\neg z) := var(z) \wedge z \in P \wedge (\forall Y)\big[MM(Y) \rightarrow \neg(z \in Y)\big].$$

Deduction with CWA-rule and CCWA-rule:

$$(\forall X)\big[model'(X, T) \wedge (\forall z)\big(clo_i(\neg z) \rightarrow implies(X, \neg z)\big) \rightarrow model'(X, F)\big].$$

Deduction with ECWA-rule:

$$(\forall X)\big[MM(X) \rightarrow model'(X, F)\big].$$

The above predicates have the following meaning:

$clo_i(\neg z)$ with $i \in \{1, 2\}$ means that $\neg z$ is in the closure of $T$ w.r.t. the CWA-rule (for $i = 1$) or CCWA-rule (for $i = 2$), respectively. $MM(X)$ means $X \in MM(T; P; Q; Z)$. □

Analogously to the previous section, we immediately get the following complexity result by Courcelle's Theorem together with Theorem 3.5.

**Theorem 3.9.** *Consider the signature $\tau = \{var_T, var_F, var_{T'}, var_{F'}, cl_{T'}, cl_{F'}, pos_{T'}, pos_{F'}, neg'_{T',F'}, P, Q, Z\}$ as in Theorem 3.8. Let an instance of* Circumscription *or of the* Deduction *problem $T \models_C F$ for any of the CWR-rules $\mathsf{C} \in \{$CWA, GCWA, EGCWA, CCWA, ECWA$\}$ be given by a $\tau$-structure $\mathcal{A}$. Then all these problems can be solved in time $\mathcal{O}(f(w) * |\mathcal{A}|)$ for some function $f$.*

### 3.4. Propositional abduction

A *propositional abduction problem* (PAP, for short) consists of a tuple $\langle V, H, M, \mathcal{C}\rangle$, where $V$ is a finite set of *propositional variables*, $H \subseteq V$ is the set of *hypotheses*, $M \subseteq V$ is the set of *manifestations*, and $\mathcal{C}$ is a consistent *theory* in the form of a clause set. A set $\mathcal{S} \subseteq H$ is a *solution* to $\mathcal{P}$ if $\mathcal{C} \cup \mathcal{S}$ is consistent and $\mathcal{C} \cup \mathcal{S} \models M$ holds. In an abductive diagnosis problem, the manifestations $M$ are the observed symptoms (e.g. describing some erroneous behavior of the system) and the clausal theory $\mathcal{C}$ constitutes the system description. The solutions $\mathcal{S} \subseteq H$ are the possible explanations for the observed symptoms.

Given a PAP $\mathcal{P}$, the basic problems of propositional abduction are the following:

- SOLVABILITY: Does there exist a solution of $\mathcal{P}$?
- RELEVANCE: Given $h \in H$, is $h$ contained in at least one solution of $\mathcal{P}$?
- NECESSITY: Given $h \in H$, is $h$ contained in every solution of $\mathcal{P}$?

In [21], the former two problems were shown to be $\Sigma_2^p$-complete while the latter is $\Pi_2^p$-complete.

The following example from [34] should help to illustrate these definitions. Consider the following football knowledge base:

$$\begin{aligned}\mathcal{C} = \{&\mathit{weak\_defense} \wedge \mathit{weak\_attack} \rightarrow \mathit{match\_lost},\\ &\mathit{match\_lost} \rightarrow \mathit{manager\_sad} \wedge \mathit{press\_angry}\\ &\mathit{star\_injured} \rightarrow \mathit{manager\_sad} \wedge \mathit{press\_sad}\}.\end{aligned}$$

Moreover, let the set of observed manifestations and the set of hypotheses be

$$M = \{\mathit{manager\_sad}\},$$

$$H = \{\mathit{star\_injured}, \mathit{weak\_defense}, \mathit{weak\_attack}\}.$$

This PAP has the following five abductive explanations (= "solutions"):

$$\mathcal{S}_1 = \{\mathit{star\_injured}\},$$

$$\mathcal{S}_2 = \{\mathit{weak\_defense}, \mathit{weak\_attack}\},$$

$$\mathcal{S}_3 = \{\mathit{weak\_attack}, \mathit{star\_injured}\},$$

$$\mathcal{S}_4 = \{\mathit{weak\_defense}, \mathit{star\_injured}\},$$

$$\mathcal{S}_5 = \{\mathit{weak\_defense}, \mathit{weak\_attack}, \mathit{star\_injured}\}.$$

For bounded treewidth of $\mathcal{C}$, we establish below the fixed-parameter tractability of the SOLVABILITY, RELEVANCE, and NECESSITY problem. A PAP $\mathcal{P}$ can be represented as a $\tau$-structure with $\tau = \{cl, var, pos, neg, H, M\}$, where the predicates $cl, var, pos, neg$ represent the clause set $\mathcal{C}$ and the unary predicates $H$ and $M$ identify the hypotheses and manifestations. In order to represent an instance of the RELEVANCE and NECESSITY problem, we extend the signature $\tau$ to $\tau' = \{cl, var, pos, neg, H, M, Dist\}$, where $Dist$ is a unary relation used for *distinguishing* some domain element, i.e., in addition to the $\tau$-predicates for expressing the PAP, the structure contains a fact $Dist(h)$ to express that we want to test hypothesis $h$ for Relevance respectively necessity. Then we have:

**Theorem 3.10.** *Consider the signature $\tau' = \{cl, var, pos, neg, H, M, Dist\}$. The problems* SOLVABILITY, RELEVANCE, *and* NECESSITY *of propositional abduction can be expressed by means of MSO sentences over the signature $\tau'$ (of course, for solvability, the predicate Dist is not used).*

**Proof.** Recall from Theorem 3.2 that the property that $X$ is a model of a clause set (or, equivalently, a formula in CNF) $\mathcal{C}$, can be expressed by the MSO-formula $model(X, F)$ using the predicates $cl, var, pos, neg$. We construct MSO-sentences for the above three decision problems as follows:

$$Sol(S) := S \subseteq H \wedge (\exists X)\big[model(X, \mathcal{C}) \wedge S \subseteq X\big] \wedge (\forall Y)\big[\big(model(Y, \mathcal{C}) \wedge S \subseteq Y\big) \rightarrow M \subseteq Y\big],$$

SOLVABILITY: $(\exists S)Sol(S)$,

RELEVANCE: $(\exists S)\big[Sol(S) \wedge Dist \subseteq S\big]$,

NECESSITY: $(\forall S)\big[Sol(S) \rightarrow Dist \subseteq S\big]$.

The predicate $Sol(S)$ is a straightforward formulation of the property "$S$ is a solution of the PAP represented by $\mathcal{A}$", namely (i) $S$ is a subset of the propositional variables in $H$, (ii) $(\mathcal{C} \cup S)$ has at least one model $X$, and (iii) every model $Y$ of $(\mathcal{C} \cup S)$ is also a model of $M$. □

Usually, a refined version of the RELEVANCE (respectively NECESSITY) problem is considered. Rather than asking whether $h$ is contained in some (respectively every) solution, one is interested if $h$ is contained in some (respectively every) *acceptable* solution. In this context, "acceptable" means "minimal" w.r.t. some preorder $\preccurlyeq$ on the powerset $2^H$. Consequently, one speaks of $\preccurlyeq$-RELEVANCE (respectively $\preccurlyeq$-NECESSITY). The above treated basic abduction problems RELEVANCE and NECESSITY correspond to the special case where $\preccurlyeq$ is the equality. The other preorders studied are the following:

- subset-minimality "$\subseteq$",
- prioritization "$\subseteq_P$" for a fixed number $p$ of priorities: $H$ is partitioned into "priorities" $H_1, \ldots, H_p$. Then $A \subseteq_P B$ if and only if $A = B$ or there exists a $k$ s.t. $A \cap H_i = B \cap H_i$ for all $i < k$ and $A \cap H_k \subset B \cap H_k$,

- minimum cardinality "$\leqslant$": $A \leqslant B$ if and only if $|A| \leqslant |B|$,
- penalization "$\sqsubseteq_p$" (also referred to as "weighted abduction"): To each element $h \in H$, a weight $w(h)$ is attached. Then $A \sqsubseteq_p B$ if and only if $\sum_{h\in A} w(h) \leqslant \sum_{h\in B} w(h)$.

To illustrate the various notions of minimality in abduction, we revisit the football example from [34], which was recalled above. $\mathcal{S}_1$ and $\mathcal{S}_2$ are $\subseteq$-minimal, but only $\mathcal{S}_1$ is $\leqslant$-minimal. Priorities may be used to represent a qualitative version of probability. For instance, suppose that for some reason we know that (for a specific team) *star_injured* is much less likely to occur than *weak_defense* and *weak_attack*. This judgment can be formalized by assigning lower priority to the former. Then $\mathcal{S}_2$ is the only minimal solution with respect to the preorder $\subseteq_P$. If we have numeric values available for the repair cost or for the robustness of each component (e.g., based on data such as the empirically collected mean time to failure and component age), then the $\sqsubseteq_p$-minimal solutions correspond to the cheapest repair respectively the most likely explanation.

In all of the above cases, the resulting $\preccurlyeq$-Relevance (respectively $\preccurlyeq$-Necessity) problem is on the second or third level of the polynomial hierarchy [21]. Again, we show that the computational complexity decreases significantly if the clausal theories $\mathcal{C}$ under consideration have bounded treewidth. The last two cases turn out to be a bit tricky. Below, we establish the desired FPT-result for the first two ones:

**Theorem 3.11.** *Consider the signature $\tau' = \{cl, var, pos, neg, H, M, Dist\}$. For $\preccurlyeq \in \{\subseteq, \subseteq_P\}$, both the $\preccurlyeq$-Relevance problem and the $\preccurlyeq$-Necessity problem can be expressed by means of MSO sentences over the signature $\tau'$.*

**Proof.** It suffices to provide an MSO encoding of the predicates $Acc_{\subseteq}(S)$ and $Acc_{\subseteq_P}(S)$, which mean that $S$ is an acceptable solution for the preorders $\subseteq$ and $\subseteq_P$, respectively:

$$Acc_{\subseteq}(S) := Sol(S) \wedge (\forall X)\big[(X \subset S) \rightarrow \neg Sol(X)\big],$$
$$X \subset_P S := (X \cap H_1 \subset S) \vee (X \cap H_1 \subseteq S \wedge X \cap H_2 \subset S) \vee (X \cap H_1 \subseteq S \wedge X \cap H_2 \subseteq S \wedge X \cap H_3 \subset S),$$
$$Acc_{\subseteq_P}(S) := Sol(S) \wedge (\forall X)\big[(X \subset_P S) \rightarrow \neg Sol(X)\big].$$

By $X \subset_P S$ we mean that $X \subseteq_P S$ and $X \neq S$. Note that we are only considering 3 priority levels $H_1$, $H_2$, and $H_3$ above. The generalization to an arbitrary but fixed number $p$ of priority levels is clear. □

By Courcelle's Theorem and Theorems 3.10 and 3.11 above, we immediately get the following FPT-result:

**Theorem 3.12.** *Consider the signature $\tau' = \{cl, var, pos, neg, H, M, Dist\}$. Let $\preccurlyeq \in \{\subseteq, \subseteq_P\}$ and let an instance of the $\preccurlyeq$-Relevance or $\preccurlyeq$-Necessity problem be given by a $\tau'$-structure $\mathcal{A}$ of width $w$. Then these problems can be solved in time $\mathcal{O}(f'(w) * |\mathcal{A}|)$ for some function $f'$.*

It remains to consider the cases $\preccurlyeq \in \{\leqslant, \sqsubseteq_p\}$. Actually, $\leqslant$ is a special case of $\sqsubseteq_p$, where every $h \in H$ is assigned the same weight. Unfortunately, MSO is not powerful enough to express the cardinality-comparison $\leqslant$ (cf. the discussion in [3]). Nevertheless, also for $\preccurlyeq \in \{\leqslant, \sqsubseteq_p\}$, it is possible to establish the FPT-property in case of bounded treewidth via an extension of Courcelle's Theorem proved in [3,16]. To this end, we have to recall the definition of extremum problems via an extension of MSO (the definition in [3] is more general than our definition below; however, for our purposes, this restricted form is sufficient).

**Definition 3.13.** Let $\tau$ be a signature and $\varphi(X_1, \ldots, X_m)$ an MSO formula over signature $\tau$ with free set variables $X_1, \ldots, X_m$. Moreover, let $F: \mathbb{N}^m \rightarrow \mathbb{N}$ be a linear function. Then the *linear extended MSO extremum problem* of $\varphi$ and $F$ is defined as follows:

*Instance.* A $\tau$-structure $\mathcal{A}$ together with functions $g_1^{\mathcal{A}}, \ldots, g_m^{\mathcal{A}}: dom(\mathcal{A}) \rightarrow \mathbb{N}$.

*Question.* Find the extremum (either *min* or *max*) of

$$F\Big(\sum_{a\in A_1} g_1^{\mathcal{A}}(a), \ldots, \sum_{a\in A_m} g_m^{\mathcal{A}}(a)\Big)$$

over all tuples of sets $A_1, \ldots, A_m$ over the domain of $\mathcal{A}$ with $\mathcal{A} \models \varphi(A_1, \ldots, A_m)$.

**Theorem 3.14.** *(See [3].) Let a linear extended MSO extremum problem be given by a linear objective function $F$ and an MSO formula $\varphi(X_1, \ldots, X_m)$ over some signature $\tau$. Moreover, let an instance of this extremum problem be given by a $\tau$-structure $\mathcal{A}$ of treewidth $w$ together with functions $g_1^{\mathcal{A}}, \ldots, g_m^{\mathcal{A}}: dom(\mathcal{A}) \rightarrow \mathbb{N}$. Assuming unit cost for arithmetic operations, this extremum problem can be solved in time $\mathcal{O}(f(|\varphi(x)|, F, w) * |(\mathcal{A}, g_1^{\mathcal{A}}, \ldots, g_m^{\mathcal{A}})|)$ for some function $f$.*

This additional expressive power suffices for expressing the acceptability of PAP-solutions w.r.t. $\leqslant$ and $\sqsubseteq_p$:

**Theorem 3.15.** *Consider the signature $\tau' = \{cl, var, pos, neg, H, M, Dist\}$. Let $\preccurlyeq \in \{\leqslant, \sqsubseteq_p\}$ and let an instance of the $\preccurlyeq$-*Relevance *or $\preccurlyeq$-*Necessity *problem be given by a $\tau'$-structure $\mathcal{A}$ of width $w$. In case of $\sqsubseteq_p$-abduction, the problem instance additionally contains a weight function $g^{\mathcal{A}}$. Assuming unit cost for arithmetic operations, these problems can be solved in time $\mathcal{O}(f(w) * |\mathcal{A}|)$ (for the preorder $\leqslant$), respectively, in time $\mathcal{O}(f(w) * |(\mathcal{A}, g^{\mathcal{A}})|)$ (for the preorder $\sqsubseteq_p$) for some function $f$.*

**Proof.** It suffices to show that these problems can be decided by solving linear extended MSO extremum problems. We only consider the case of $\sqsubseteq_p$, since $\leqslant$ is a special case.

Let an instance of the $\sqsubseteq_p$-Relevance or $\sqsubseteq_p$-Necessity problem be given as a $\tau'$-structure $\mathcal{A}$ with $\tau' = \{cl, var, pos, neg, H, M, Dist\}$. Moreover, let $Sol(S)$ denote the MSO-formula from the proof of Theorem 3.10, expressing that $S$ is a solution of the PAP represented by $\mathcal{A}$. Clearly, the task of computing the minimal total weight over all solutions of this PAP corresponds to the linear extremum (i.e., minimum) problem defined by the MSO-formula $Sol(S)$ and taking the identity function as objective function $F$. The instances of this extremum problem are given as $\tau'$-structure $\mathcal{A}$ together with the function $g^{\mathcal{A}}$ being the weight function on the domain elements of $\mathcal{A}$.

Now let $Sol'(S)$ and $Sol''(S)$ be defined as $Sol'(S) := Sol(S) \wedge Dist \subseteq S$ and $Sol''(S) := Sol(S) \wedge Dist \not\subseteq S$, i.e., $Sol'(S)$ (respectively $Sol''(S)$) expresses the property that $S$ is a solution containing $h$ (respectively not containing $h$). Then $h$ is relevant if and only if the following properties hold: $\mathcal{A} \models (\exists S) Sol'(S)$ and the minimal weight over all sets $S$ fulfilling $\mathcal{A} \models (\exists S) Sol'(S)$ is the same as the minimal weight over all sets $S$ fulfilling $\mathcal{A} \models (\exists S) Sol(S)$. Both minimum-problems can be expressed as linear extended MSO extremum problems and are therefore solvable in the desired time bound. Likewise, the decision problem $\mathcal{A} \models (\exists S) Sol'(S)$ fits into this time bound.

Similarly, $h$ is necessary if and only if the following properties hold: Either $\mathcal{A} \not\models (\exists S) Sol''(S)$ holds or all of the following conditions are fulfilled: $\mathcal{A} \models (\exists S) Sol'(S)$, $\mathcal{A} \models (\exists S) Sol''(S)$, and the minimal weight over all sets $S$ with $\mathcal{A} \models (\exists S) Sol'(S)$ is strictly smaller than the minimal weight over all sets $S$ with $\mathcal{A} \models (\exists S) Sol''(S)$, i.e., all solutions of minimal weight actually do contain $h$. These decision problems as well as the minimum-problems can solved in the desired time bound. □

## 4. New algorithms via datalog

In this section, we show how the monadic datalog approach from [30] can be exploited in order to construct concrete, efficient algorithms for the problems whose fixed-parameter tractability was established in Section 3. We shall present new algorithms for the Solvability problem (i.e., does a given PAP have at least one solution) and for the Relevance enumeration problem (i.e., enumerate all relevant hypotheses of a given PAP) of propositional abduction. We believe that this approach is also applicable to other problems discussed in Section 3. Indeed, recently similar algorithms have been presented for circumscription and for disjunctive logic programming, see [36,37].

It is convenient to consider a PAP $\mathcal{P}$ to be given as a $\tau$-structure with $\tau = \{cl, var, pos, neg, hyp, man\}$. Note that, in contrast to Section 3.4, we denote the unary predicates for the hypotheses and manifestations as *hyp* and *man* rather than $H$ and $M$. This change of notation should help to increase readability of the datalog programs by conforming to the convention that predicate names are normally written in lower-case letters. As in Section 3, we again start our exposition with the SAT problem in order to illustrate the basic ideas before we tackle the harder problems in the area of abduction. Note that an efficient algorithm for exploiting bounded treewidth in the context of the SAT problem has been recently presented in [24]. The algorithm presented there is based on recursive splitting. In [24], the authors also outline how their method can be extended to other NP-complete problems (or their #P-complete counting variants). However, our goal is to tackle also problems on the second level of the polynomial hierarchy like abduction.

### 4.1. SAT problem

Suppose that a clause set together with a tree decomposition $\mathcal{T}$ of width $w$ is given as a $\tau_{td}$-structure with $\tau_{td} = \{cl, var, pos, neg, root, leaf, child_1, child_2, bag\}$. Of course, we do not need the predicates *hyp* and *man* for the SAT problem. Without loss of generality, we may assume that $\mathcal{T}$ is in the normal form given in Definition 2.2. Note that now the domain elements are either variables or clauses. Consequently, a node $t$ in the tree decomposition $\mathcal{T}$ is called a *variable removal node*, a *clause removal node*, a *variable introduction node*, or a *clause introduction node*, respectively, depending on the kind of element that is removed or replaced – comparing the bag at node $t$ with the bag at the child node of $t$. Additionally, as in Definition 2.2, a node $t$ can be a *branch node* or a *leaf node* (if it has two respectively no child nodes).

In Fig. 4, we describe a datalog program which decides the SAT problem. Some words on the notation used in this program are in order: We are using lower case letters $v$, $c$, and $x$ (possibly with subscripts) as datalog variables for a single node in $\mathcal{T}$, for a single clause, or for a single propositional variable, respectively. In contrast, upper case letters are used as datalog variables denoting sets of variables (in the case of $X, P, N$) or sets of clauses (in the case of $C$). Note that the sets are not sets in the general sense, since their cardinality is restricted by the maximal size $w + 1$ of the bags, where $w$ is a fixed constant. Indeed, we have implemented these "fixed-size" sets by means of $k$-tuples with $k \leqslant (w + 1)$ over $\{0, 1\}$. The SAT-program in Fig. 4 could therefore be said "quasi-monadic", since all but one variable in the head atoms have bounded instantiations. For instance, in the atom $solve(v, P, N, C)$, the sets $P, N, C$ are subsets of the bag of $v$. Hence, each combination $P, N, C$ could be represented by 3 subsets $r_1, r_2, r_3$ over $\{0, \ldots, w\}$ referring to indices of elements in the bag

**Program** SAT

/* leaf node. */
$solve(v, P, N, C_1) \leftarrow leaf(v),\ bag(v, X, C),\ P \cup N = X,\ P \cap N = \emptyset,$
$true(P, N, C_1, C).$

/* variable removal node. */
$solve(v, P, N, C_1) \leftarrow bag(v, X, C),\ child_1(v_1, v),\ bag(v_1, X \uplus \{x\}, C),$
$solve(v_1, P \uplus \{x\}, N, C_1).$

$solve(v, P, N, C_1) \leftarrow bag(v, X, C),\ child_1(v_1, v),\ bag(v_1, X \uplus \{x\}, C),$
$solve(v_1, P, N \uplus \{x\}, C_1).$

/* clause removal node. */
$solve(v, P, N, C_1) \leftarrow bag(v, X, C),\ child_1(v_1, v),\ bag(v_1, X, C \uplus \{c\}),$
$solve(v_1, P, N, C_1 \uplus \{c\}).$

/* variable introduction node. */
$solve(v, P \uplus \{x\}, N, C_1 \cup C_2) \leftarrow bag(v, X \uplus \{x\}, C),\ child_1(v_1, v),$
$bag(v_1, X, C),\ solve(v_1, P, N, C_1),\ true(\{x\}, \emptyset, C_2, C).$

$solve(v, P, N \uplus \{x\}, C_1 \cup C_2) \leftarrow bag(v, X \uplus \{x\}, C),\ child_1(v_1, v),$
$bag(v_1, X, C),\ solve(v_1, P, N, C_1),\ true(\emptyset, \{x\}, C_2, C).$

/* clause introduction node. */
$solve(v, P, N, C_1 \cup C_2) \leftarrow bag(v, X, C \uplus \{c\}),\ child_1(v_1, v),\ bag(v_1, X, C),$
$solve(v_1, P, N, C_1),\ true(P, N, C_2, \{c\}).$

/* branch node. */
$solve(v, P, N, C_1 \cup C_2) \leftarrow bag(v, X, C),\ child_1(v_1, v),\ bag(v_1, X, C),$
$solve(v_1, P, N, C_1),\ child_2(v_2, v),\ bag(v_2, X, C),\ solve(v_2, P, N, C_2).$

/* result (at the root node). */
$success \leftarrow root(v), bag(v, X, C),\ solve(v, P, N, C).$

**Fig. 4.** SAT test.

of $v$. Since $w$ is considered as a fixed constant, $solve(v, P, N, C)$ is simply a succinct representation of constantly many monadic predicates of the form $solve_{\langle r_1, r_2, r_3\rangle}(v)$.

For the sake of readability, we are using non-datalog expressions involving the set operators $\uplus$ (disjoint union), $\cup$, and $\cap$. Of course, they could be easily replaced by "proper" datalog expressions. For instance, $P \cup N = X$ can be replaced by union $(P, N, X)$.

In order to facilitate the discussion, we introduce the following notation. Let $\mathcal{C}$ denote the input clause set with variables in $V$ and tree decomposition $\mathcal{T}$. For any node $v$ in $\mathcal{T}$, we write $\mathcal{T}_v$ to denote the subtree of $\mathcal{T}$ rooted at $v$. By $Cl(v)$ we denote the set of clauses in the bag of $v$ while $Cl(\mathcal{T}_v)$ denotes the set of clauses that occur in any bag in $\mathcal{T}_v$. Analogously, we write $Var(v)$ and $Var(\mathcal{T}_v)$ as a short-hand for the set of variables occurring in the bag of $v$ respectively in any bag in $\mathcal{T}_v$. Finally, the restriction of a clause $c$ to the variables in some set $U \subseteq V$ will be denoted by $c|_U$.

The SAT-program contains three intensional predicates *solve*, *true*, and *success*. The crucial predicate is $solve(v, P, N, C)$ with the following intended meaning: $v$ denotes a node in $\mathcal{T}$. $P$ and $N$ form a partition of $Var(v)$ representing a truth value assignment on $Var(v)$, s.t. all variables in $P$ take the value true and all variables in $N$ take the value false. $C$ denotes a subset of $Cl(v)$. For all values of $v, P, N, C$, the ground fact $solve(v, P, N, C)$ shall be true in the minimal model of the interpreted program if and only if the following condition holds:

**Property A.** *There exists an extension $J$ of the assignment $(P, N)$ to $Var(\mathcal{T}_v)$, s.t. $(Cl(\mathcal{T}_v) \setminus Cl(v)) \cup C$ is true in $J$ while for all clauses $c \in Cl(v) \setminus C$, the restriction $c|_{Var(\mathcal{T}_v)}$ is false in $J$.*

The main task of the program is the computation of all facts $solve(v, P, N, C)$ by means of a bottom–up traversal of the tree decomposition. The other predicates have the following meaning: $true(P, N, C_1, C)$ means that $C_1$ contains precisely those clauses from $C$ which are true in the (partial) assignment given by $(P, N)$. We do not specify the implementation of this predicate here. It can be easily achieved via the extensional predicates *pos* and *neg*. The 0-ary predicate *success* indicates if the input structure is the encoding of a satisfiable clause set.

The SAT-program has the following properties.

**Lemma 4.1.** *The solve-predicate has the intended meaning described above, i.e., for all values $v, P, N$, and $C$, the ground fact $solve(v, P, N, C)$ is true in the minimal model of the interpreted SAT-program if and only if Property A holds.*

**Proof idea.** The lemma can be shown by structural induction on $\mathcal{T}$. The induction step goes via a case distinction over all possible types of nodes. The proof is lengthy but straightforward. The details are worked out in Appendix A. □

**Table 1**
SAT example.

| Node | *solve* facts |
|---|---|
| 7 | $solve(7, \{x_3\}, \emptyset, \{c_1\})$, $solve(7, \emptyset, \{x_3\}, \emptyset)$ |
| 13 | $solve(13, \emptyset, \{x_5\}, \{c_2\})$, $solve(13, \{x_5\}, \emptyset, c_2)$ |
| 19 | $solve(19, \{x_6\}, \emptyset, \{c_3\})$, $solve(19, \emptyset, \{x_3\}, \emptyset)$ |
| 6 | $solve(6, \{x_1, x_3\}, \emptyset, \{c_1\})$, $solve(6, \emptyset, \{x_1, x_3\}, \emptyset)$, $solve(6, \{x_1\}, \{x_3\}, \{c_1\})$, $solve(6, \{x_3\}, \{x_1\}, \{c_1\})$ |
| 5 | $solve(5, \{x_1\}, \emptyset, \{c_1\})$, $solve(5, \emptyset, \{x_1\}, \emptyset)$, $solve(5, \emptyset, \{x_1\}, \{c_1\})$ |
| 4 | $solve(4, \{x_1, x_2\}, \emptyset, \{c_1\})$, $solve(4, \{x_1\}, \{x_2\}, \{c_1\})$, $solve(4, \{x_2\}, \{x_1\}, \emptyset)$, $solve(4, \emptyset, \{x_1, x_2\}, \{c_1\})$, $solve(4, \{x_2\}, \{x_1\}, \{c_1\})$ |
| 3 | $solve(3, \{x_1, x_2\}, \emptyset, \emptyset)$, $solve(3, \{x_1\}, \{x_2\}, \emptyset)$, $solve(3, \{x_2\}, \{x_1\}, \emptyset)$, $solve(3, \emptyset, \{x_1, x_2\}, \emptyset)$ |
| 1 | $solve(1, \{x_1, x_2, x_4\}, \emptyset, \emptyset)$, $solve(1, \{x_1, x_2\}, \{x_4\}, \emptyset)$, $solve(1, \{x_1, x_4\}, \{x_2\}, \emptyset)$, $solve(1, \{x_1\}, \{x_2, x_4\}, \emptyset)$, $solve(1, \{x_2, x_4\}, \{x_1\}, \emptyset)$, $solve(1, \{x_2\}, \{x_1, x_4\}, \emptyset)$, $solve(1, \{x_4\}, \{x_1, x_2\}, \emptyset)$, $solve(1, \emptyset, \{x_1, x_2, x_4\}, \emptyset)$ |
| 2 | $solve(2, \{x_1, x_2, x_4\}, \emptyset, \emptyset)$, $solve(2, \{x_1, x_2\}, \{x_4\}, \emptyset)$, $solve(2, \{x_1, x_4\}, \{x_2\}, \emptyset)$, $solve(2, \{x_1\}, \{x_2, x_4\}, \emptyset)$, $solve(2, \{x_2, x_4\}, \{x_1\}, \emptyset)$, $solve(2, \{x_2\}, \{x_1, x_4\}, \emptyset)$, $solve(2, \{x_4\}, \{x_1, x_2\}, \emptyset)$, $solve(2, \emptyset, \{x_1, x_2, x_4\}, \emptyset)$ |
| 0 | $solve(0, \{x_1, x_2, x_4\}, \emptyset, \emptyset)$, $solve(0, \{x_1, x_2\}, \{x_4\}, \emptyset)$, $solve(0, \{x_1, x_4\}, \{x_2\}, \emptyset)$, $solve(0, \{x_1\}, \{x_2, x_4\}, \emptyset)$, $solve(0, \{x_2, x_4\}, \{x_1\}, \emptyset)$, $solve(0, \{x_2\}, \{x_1, x_4\}, \emptyset)$, $solve(0, \{x_4\}, \{x_1, x_2\}, \emptyset)$, $solve(0, \emptyset, \{x_1, x_2, x_4\}, \emptyset)$ |

**Theorem 4.2.** *The datalog program in Fig.* 4 *decides the SAT problem, i.e., the fact "success" is true in the minimal model of the interpreted SAT-program if and only if the input structure $\mathcal{A}_{td}$ encodes a satisfiable clause set $\mathcal{C}$ together with a tree decomposition $\mathcal{T}$ of $\mathcal{C}$. Moreover, for any clause set $\mathcal{C}$ and tree decomposition $\mathcal{T}$ of width at most $w$, the program can be evaluated in time $\mathcal{O}(2^w * |\mathcal{C}|)$.*

**Proof.** By Lemma 4.1, the *solve*-predicate has the meaning according to Property A. Thus, the rule with head *success* reads as follows: *success* is true in the minimal model if and only if $v$ denotes the root of $\mathcal{T}$ and there exists an assignment $(P, N)$ on the variables in $Var(v)$, s.t. for some extension $J$ of $(P, N)$ to $Var(\mathcal{T}_v)$, all clauses in $Cl(\mathcal{T}_v) = (Cl(\mathcal{T}_v) \setminus Cl(v)) \cup C$ are true in $J$. But this simply means that $J$ is a satisfying assignment of $\mathcal{C} = Cl(\mathcal{T}_v)$.

For the upper bound on the time complexity, we observe that, in all facts $solve(v, P, N, C)$ derived by the program, the sets $P, N, C$ are disjoint subsets of the bag at $v$ (which contains at most $w + 1$ elements). Suppose that the bag at node $v$ consists of $k$ variables and $l$ clauses with $k + l \leqslant w + 1$. Then there are at most $2^k * 2^l = 2^{w+1}$ possible instantiations of these two arguments.

Hence, the datalog program $\mathcal{P}$ in Fig. 4 is equivalent to a ground program $\mathcal{P}'$ where each rule of $\mathcal{P}$ is replaced by $\mathcal{O}(2^w * |\mathcal{C}|)$ ground rules, which can be computed in time $\mathcal{O}(2^w * |\mathcal{C}|)$. In total, we can evaluate the program $\mathcal{P}$ in Fig. 4 over an input formula $\mathcal{C}$ in CNF together with a tree decomposition $\mathcal{T}$ (given as a $\tau_{td}$-structure $\mathcal{A}_{td}$, whose size is linear w.r.t. the size of $\mathcal{C}$) as follows: we first compute the equivalent ground program $\mathcal{P}'$ with $\mathcal{O}(2^w * |\mathcal{C}|)$ rules and then evaluate $\mathcal{P}'$ in linear time [18,42] w.r.t. the size of $\mathcal{P}'$ and the size of $\mathcal{A}_{td}$, i.e., in time $\mathcal{O}(2^w * |\mathcal{C}| + |\mathcal{A}_{td}|) = \mathcal{O}(2^w * |\mathcal{C}|)$. □

**Example 4.3.** Let $F = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_4 \vee \neg x_5) \wedge (x_2 \vee \neg x_4 \vee x_6)$ be the propositional formula from Example 2.1 and consider the normalized tree decomposition $\mathcal{T}$ in Fig. 2. We shall run the SAT-program on the structure representing $F$ and $\mathcal{T}$. We process the tree nodes in a bottom–up manner. The *solve*-facts derived at each tree node are presented in Table 1.

We start with the leaf nodes 7, 13, and 19. At node 7, the variable $x_3$ can be assigned either the value true or false. If $x_3$ is assigned the value true ($P = \{x_3\}$ and $N = \emptyset$), then clause $c_1$ is true. Hence, we obtain the fact $solve(7, \{x_3\}, \emptyset, \{c_1\})$. If $x_3$ is assigned the value false ($P = \emptyset$ and $N = \{x_3\}$), then clause $c_1$ (restricted to the variable $x_3$) evaluates to false. Thus, we obtain the second fact $solve(7, \emptyset, \{x_3\}, \emptyset)$. By analogous arguments, we obtain the *solve*-facts for the leaf nodes 13 and 19.

Node 6 is a variable introduction node, which introduces the variable $x_1$ (which is not present in node 7). Again, $x_1$ can be either set to true or false. For each of the assignments, we test whether $c_1$ evaluates true. Therefore, at node 6, we obtain four *solve*-facts with all possible truth assignments for $x_1$ and $x_3$.

Node 5 is a variable removal node, which removes the variable $x_3$ (which was present in node 6). We *remember* the effect of the truth assignment to the removed variable (in this case $x_3$) by maintaining different sets of clauses in the *solve*-facts. Thus, at node 5, there exist two facts for the truth assignment $P = \emptyset$ and $N = \{x_1\}$, namely one with $C = \{c_1\}$ and one with $C = \emptyset$ – see the second and the third *solve*-fact in Table 1.

**Program** Solvability

/* leaf node. */
$solve(v, S, 0, P, N, C_1, d) \leftarrow leaf(v),\ bag(v, X, C),\ svar(v, S),\ S \subseteq P,$
$P \cup N = X,\ P \cap N = \emptyset,\ check(P, N, C_1, C, d).$

/* variable removal node. */
$aux(v, S, i, 0, P, N, C_1, d) \leftarrow bag(v, X, C),\ child_1(v_1, v),\ bag(v_1, X \uplus \{x\}, C),$
$solve(v_1, S, i, P \uplus \{x\}, N, C_1, d),\ x \notin S.$

$aux(v, S, i, 0, P, N, C_1, d) \leftarrow bag(v, X, C),\ child_1(v_1, v),\ bag(v_1, X \uplus \{x\}, C),$
$solve(v_1, S, i, P, N \uplus \{x\}, C_1, d).$

$aux(v, S, i, 1, P, N, C_1, d) \leftarrow bag(v, X, C),\ child_1(v_1, v),\ bag(v_1, X \uplus \{x\}, C),$
$solve(v_1, S \uplus \{x\}, i, P \uplus \{x\}, N, C_1, d).$

/* clause removal node. */
$solve(v, S, i, P, N, C_1, d) \leftarrow bag(v, X, C),\ child_1(v_1, v),\ bag(v_1, X, C \uplus \{c\}),$
$solve(v_1, S, i, P, N, C_1 \uplus \{c\}, d).$

/* variable introduction node. */
$solve(v, S, i, P \uplus \{x\}, N, C_1 \cup C_2, d_1) \leftarrow bag(v, X \uplus \{x\}, C),\ child_1(v_1, v),$
$bag(v_1, X, C),\ solve(v_1, S, i, P, N, C_1, d_1),\ check(\{x\}, \emptyset, C_2, C, d_2).$

$solve(v, S, i, P, N \uplus \{x\}, C_1 \cup C_2, d_1 \text{ or } d_2) \leftarrow bag(v, X \uplus \{x\}, C),\ child_1(v_1, v),$
$bag(v_1, X, C),\ solve(v_1, S, i, P, N, C_1, d_1),\ check(\emptyset, \{x\}, C_2, C, d_2).$

$solve(v, S \uplus \{x\}, i, P \uplus \{x\}, N, C_1 \cup C_2, d_1) \leftarrow bag(v, X \uplus \{x\}, C),\ child_1(v_1, v),$
$bag(v_1, X, C),\ solve(v_1, S, i, P, N, C_1, d_1),\ check(\{x\}, \emptyset, C_2, C, d_2),\ hyp(x).$

/* clause introduction node. */
$solve(v, S, i, P, N, C_1 \cup C_2, d_1) \leftarrow bag(v, X, C \uplus \{c\}),\ child_1(v_1, v),$
$bag(v_1, X, C),\ solve(v_1, S, i, P, N, C_1, d_1),\ check(P, N, C_2, \{c\}, d_2).$

/* branch node. */
$aux(v, S, i_1, i_2, P, N, C_1 \cup C_2, d_1 \text{ or } d_2) \leftarrow bag(v, X, C),$
$child_1(v_1, v),\ bag(v_1, X, C),\ child_2(v_2, v),\ bag(v_2, X, C),$
$solve(v_1, S, i_1, P, N, C_1, d_1),\ solve(v_2, S, i_2, P, N, C_2, d_2).$

/* variable removal and branch node: $aux \Rightarrow solve$ */
$solve(v, S, i, P, N, C, d) \leftarrow aux(v, S, i_1, i_2, P, N, C, d),\ reduce(v, S, i, i_1, i_2).$

/* result (at the root node). */
$success \leftarrow root(v),\ bag(v, X, C),\ solve(v, S, i, P, N, C,$ 'n'), not $solve'(v, S, i).$

$solve'(v, S, i) \leftarrow bag(v, X, C),\ solve(v, S, i, P, N, C,$ 'y').

**Fig. 5.** Solvability test.

The variable introduction node 4 follows the same principles as node 6. Note however that the number of *solve*-facts is not doubled by the introduction of variable $x_2$. Indeed, the facts $solve(5, \emptyset, \{x_1\}, \emptyset)$ and $solve(5, \emptyset, \{x_1\}, \{c_1\})$ both give rise to the fact $solve(4, \{x_2\}, \{x_1\}, \{c_1\})$. Hence, we have only 5 *solve*-facts at node 4.

Node 3 is a clause removal node where the clause $c_1$ is removed (comparing the bag of node 3 with node 4). Only those truth assignments in *solve*-facts from node 4 "survive", where the clause $c_1$ evaluates to true. Actually, in this example, only the fact $solve(4, \{x_2\}, \{x_1\}, \emptyset)$ violates this condition. However, there also exists a fact $solve(4, \{x_2\}, \{x_1\}, \{c_1\})$ at node 4. The latter gives rise to the fact $solve(3, \{x_2\}, \{x_1\}, \emptyset)$.

The traversal from node 3 to 1 introduces the new variable $x_4$, which is trivial, because there does not exist any clause in node 1. Thus for each of the eight truth assignments there is a *solve*-fact in node 1. The traversal from node 13 to 8, as well as from 19 to 14 are almost identical to those from node 7 to 1. Hence we do not illustrate the *solve*-facts explicitly.

Finally, at the root node 0, we check whether there is a truth assignment of the variables that makes all the clauses in the root true. This is trivial in our example. In fact, we obtain that *all* possible truth assignments to the variables $x_1, x_2$ and $x_4$ are solutions, since they can all be extended to models of $F$ by choosing appropriate truth values for $x_3$, $x_5$, and $x_6$.

### 4.2. Solvability problem

The SAT-program from the previous section can be extended to a Solvability program via the following idea: Recall that $\mathcal{S} \subseteq H$ is a solution of a PAP $\mathcal{P} = \langle V, H, M, \mathcal{C} \rangle$ if and only if $\mathcal{C} \cup \mathcal{S}$ is consistent and $\mathcal{C} \cup \mathcal{S} \models M$ holds. We can thus think of the abduction problem as a combination of SAT and UNSAT problems, namely $\mathcal{C} \cup \mathcal{S}$ has to be satisfiable and all formulae $\mathcal{C} \cup \mathcal{S} \cup \{\neg m\}$ for any $m \in M$ have to be unsatisfiable. Suppose that we construct such a set $\mathcal{S}$ along a bottom–up traversal of $\mathcal{T}$. Initially, $\mathcal{S}$ is empty. In this case, $\mathcal{C} \cup \mathcal{S} = \mathcal{C}$ is clearly satisfiable (otherwise the abduction problem makes no sense) and $\mathcal{C} \cup \mathcal{S} \cup \{\neg m\}$ is also satisfiable for at least one $M$ (otherwise the abduction problem is trivial). In other words, $\mathcal{C} \cup \mathcal{S}$ has many models – among them are also models where some $m \in M$ is false. The effect of adding a hypothesis $h$ to $\mathcal{S}$ is that we restrict the possible number of models of $\mathcal{C} \cup \mathcal{S}$ and of $\mathcal{C} \cup \mathcal{S} \cup \{\neg m\}$ in the sense that we eliminate all models where $h$

is false. Hence, the goal of our Solvability algorithm is to find (by a bottom–up traversal of the tree decomposition $\mathcal{T}$) a set $\mathcal{S}$ which is small enough so that at least one model of $\mathcal{C} \cup \mathcal{S}$ is left while all models of $\mathcal{C} \cup \mathcal{S} \cup \{\neg m\}$ for any $m \in M$ are eliminated.

The program in Fig. 5 realizes this SAT/UNSAT-intuition. For the discussion of this program, it is convenient to introduce the following additional notation. We shall write $H(v)$, $M(v)$, $H(\mathcal{T}_v)$ and $M(\mathcal{T}_v)$ to denote the restriction of $H$ and $M$ to the variables in the bag of $v$ or in any bag in $\mathcal{T}_v$, respectively. Of course, the unary predicates *hyp* and *man* are now contained in $\tau_{td}$.

The predicate $solve(v, S, i, P, N, C, d)$ has the following intended meaning: At every node $v$, we consider choices $S \subseteq H(v)$. $(P, N)$ again denotes an assignment on the variables in $Var(v)$ and $C \subseteq Cl(v)$ denotes a clause set, s.t. $(Cl(\mathcal{T}_v) \setminus Cl(v)) \cup C$ is true in some extension $J$ of $(P, N)$ to $Var(\mathcal{T}_v)$. But now we have to additionally consider the chosen hypotheses in $H(\mathcal{T}_v)$ and the manifestations in $M(\mathcal{T}_v)$ which decide whether $J$ is a candidate for the SAT and/or UNSAT problem. As far as $H$ is concerned, we have to be careful as to how $S \subseteq H(v)$ is extended to $\bar{S} \subseteq H(\mathcal{T}_v)$. For a different extension $\bar{S}$, different assignments $J$ on $Var(\mathcal{T}_v)$ are excluded from the SAT/UNSAT problems, since we only keep track of assignments $J$ where all hypotheses in $\bar{S}$ are true. Hence, we need a counter $i \in \{0, 1, 2, \ldots\}$ as part of the *solve*-predicate in order to distinguish between different extensions of $S \subseteq H(v)$ to $\bar{S} \subseteq H(\mathcal{T}_v)$. As far as $M$ is concerned, we have the argument $d$ with possible values 'y' and 'n' indicating whether some manifestation $m \in M(\mathcal{T}_v)$ is false in $J$. For the UNSAT problem, we take into account only those assignments $J$ where at least one $m \in M$ is false.

Then the program has the following meaning: For all values of $v, S, i$, and for any extension $\bar{S}$ of $S$ with $\bar{S} \subseteq (H(\mathcal{T}_v) \setminus H(v)) \cup S$, we define:

**Property B.** *Property B For all values of $P, N, C$, and $u$, the fact $solve(v, S, i, P, N, C, u)$ is true in the minimal model of the interpreted program if and only if there exists an extension $J$ of the assignment $(P, N)$ to $Var(\mathcal{T}_v)$, s.t. $(Cl(\mathcal{T}_v) \setminus Cl(v)) \cup \bar{S} \cup C$ is true in $J$ while for all clauses $c \in Cl(v) \setminus C$, the restriction $c|_{Var(\mathcal{T}_v)}$ is false in $J$. Moreover, $u =$ 'y' if and only if some $m \in M(\mathcal{T}_v)$ is false in $J$.*

The predicate *svar* in Fig. 5 is used to select sets of hypotheses, i.e., $svar(v, S)$ is true for every subset $S \subseteq H(v)$. The predicate *check* extends the predicate *true* from the SAT-program by additionally setting the $d$-bit, i.e., $check(P, N, C_1, C, d)$ if and only if $true(P, N, C_1, C)$. Moreover, $d =$ 'y' if and only if $N$ contains some manifestation.

Finally, the predicates *aux* and *reduce* have the following purpose: As was mentioned above, the index $i$ in $solve(v, S, i, P, N, C, d)$ is used to keep different extensions $\bar{S} \subseteq H(\mathcal{T}_v)$ of $S$ apart. Without further measures, we would thus loose the fixed-parameter tractability since the variable elimination nodes and branch nodes lead to an exponential increase (w.r.t. the number of hypotheses in $H(\mathcal{T}_v)$) of the number of extensions $\bar{S}$. The predicates *aux* and *reduce* remedy this problem as follows: In the first place, we compute facts $aux(v, S, i_1, i_2, P, N, C, d)$, where different extensions $\bar{S}$ of $S$ are identified by pairs of indices $(i_1, i_2)$. Now let $v$ and $S$ be fixed and consider for each pair $(i_1, i_2)$ the set $\mathcal{F}(i_1, i_2) = \{(P, N, C, d) \mid aux(v, S, i_1, i_2, P, N, C, d)$ is true in the minimal model$\}$. The predicate $reduce(v, S, i, i_1, i_2)$ maps pairs of indices $(i_1, i_2)$ to a unique index $i$. However, if there exists a lexicographically smaller pair $(j_1, j_2)$ with $\mathcal{F}(i_1, i_2) = \mathcal{F}(j_1, j_2)$, then $(i_1, i_2)$ is skipped. In other words, if two extensions $\bar{S}$ with index $(i_1, i_2)$ and $(j_1, j_2)$ are not distinguishable at $v$ (i.e., they give rise to facts $aux(v, S, i_1, i_2, P, N, C, d)$ and $aux(v, S, j_1, j_2, P, N, C, d)$ with exactly the same sets of values $(P, N, C, d)$), then it is clearly sufficient to keep track of exactly one representative. The predicate *reduce* could be easily implemented in datalog (with negation), see Appendix B. However, we prefer to consider it as a built-in predicate which can be implemented very efficiently via appropriate hash codes.

Analogously to Lemma 4.1 and Theorem 4.2, the Solvability program has the following properties.

**Lemma 4.4.** *The solve-predicate has the intended meaning described above, i.e., for all values $v, S, i$, and for any extension $\bar{S}$ of $S$ with $\bar{S} \subseteq (H(\mathcal{T}_v) \setminus H(v)) \cup S$, Property B holds.*

**Proof sketch.** The lemma can be shown by structural induction on $\mathcal{T}$. We restrict ourselves here to outlining the ideas underlying the various rules of the Solvability program. The induction itself is then obvious and therefore omitted.

(1) *Leaf nodes.* The rule for a leaf node $v$ realizes two "guesses" so to speak: (i) a subset $S$ of the hypotheses in $H(v)$ and (ii) a truth assignment $(P, N)$ on the variables in $Var(v)$. The values of $S, P, N$ have to fulfill several conditions, namely $S \subseteq P$, $P \cup N = X$, and $P \cap N = \emptyset$. The remaining variables in this rule are fully determined. In particular, the atom $check(P, N, C_1, C, d)$ is used to compute the set of clauses $C_1 \subseteq C$ which are true in the assignment $(P, N)$ and the $d$-bit is set depending on whether some manifestation is set to false by this assignment or not.

(2) *Variable removal node.* The three rules distinguish, in total, 3 cases, which may arise when a variable $x$ is removed from the bag at the child node $v_1$ of $v$, namely: was $x$ set to true or false and was $x$ part of the solution or not. Of course, if $x$ was part of the solution then its truth value must be true. The three rules for a variable removal node treat these cases in the order (true, not solution), (false, not solution), (true, solution).

The removal of $x$ may have the effect that the *solve*-facts which are true in the minimal model for some values $(v, S, i)$ and $(v, S, j)$ become identical even if they were different for $(v_1, S, i)$ and $(v_2, S, i)$. Hence, the rule for variable removal nodes first generates *aux*-facts. The corresponding *solve*-facts are then generated only after the duplicate elimination via the *reduce*-predicate.

Recall that the index $i$ in the *solve*-facts $solve(v, S, i, \ldots)$ is used to keep different extensions $\bar{S} \subseteq H(\mathcal{T}_v)$ of $S$ apart. Analogously, in the *aux*-facts $aux(v, S, i_1, i_2, \ldots)$, the pair $(i_1, i_2)$ does this job. Therefore, the *aux*-facts for the first two cases (where $x \notin \bar{S}$) and in the third case (where $x \in \bar{S}$) have different pairs of indices $(i, 0)$ and $(i, 1)$, respectively.

(3) *Clause removal node.* The rule for a clause removal node $v$ checks if the clause $c$ that is removed at $v$ evaluates to true in an extension $J$ of $(P, N)$. This check is carried out here because (by the connectedness condition) only at the removal of $c$ we can be sure that $c|_{Var(\mathcal{T}_v)}$ is identical to $c$. Hence, if $c$ evaluates to false in an extension $J$ of $(P, N)$ to $Var(\mathcal{T}_v)$, then there is no way to turn the truth value of $c$ into true by an appropriate assignment to the variables elsewhere in the tree decomposition.

(4) *Variable introduction node.* Analogously to the variable removal node, the three rules distinguish, in total, 3 cases, which may arise when a variable $x$ is introduced at the node $v$, namely: is $x$ set to true or false and is $x$ part of the solution or not. Of course, if $x$ is part of the solution then its truth value must be true. The three rules for a variable introduction node treat these cases in the order (true, not solution), (false, not solution), (true, solution).

The third rule has the additional atom $hyp(x)$ in the body to make sure that $x$ indeed is a hypothesis when it is added to the solution. In all three cases, the *check*-predicate in the rule body checks (i) if additional clauses $C_2$ evaluate to true because of the chosen truth value of $x$, and (ii) if some manifestation is thus set to false. Of course, the check (ii) is only relevant if $x$ is set to false.

(5) *Clause introduction node.* When a new clause $c$ is introduced, we have to check if $c$ evaluates to true in the assignment $(P, N)$ or not. This is done by the atom $check(P, N, C_2, \{c\}, d_2)$ in the rule body. The variable $C_2$ thus has one of the values $\{c\}$ or $\emptyset$. Clearly, by the connectedness condition, no variables occurring in $c$ can occur in $Var(\mathcal{T}_v) \setminus Var(v)$. Hence, it plays no role which extension $J$ of $(P, N)$ to $Var(\mathcal{T}_v)$ is considered.

(6) *Branch node.* The rule for a branch node $v$ combines the *solve*-facts for the child nodes $v_1$ and $v_2$ provided that they have identical values for $S, P, N$. By the connectedness condition, we can be sure that the variables and clauses occurring both in $\mathcal{T}_{v_1}$ and $\mathcal{T}_{v_2}$ also occur in the bags of $v_1, v_2$, and $v$. Hence the extensions $\bar{S}_1$ and $\bar{S}_2$ (respectively, $J_1$ and $J_2$) are consistent if and only if $\bar{S}_1 \cap H(v) = S = \bar{S}_2 \cap H(v)$ holds (respectively, if and only if $J_1$ and $J_2$ restricted to $Var(v)$ coincide with $(P, N)$).

Note that it may happen that combining the *solve*-facts for some values $(v_1, S, i_1)$ and $(v_2, S, i_2)$ yields the same values of $(P, N, C, d)$ as if we combine the *solve*-facts for some values $(v_1, S, j_1)$ and $(v_2, S, j_2)$. Hence, the rule for branch nodes first generates *aux*-facts. The corresponding *solve*-facts are then generated only after the duplicate elimination via the *reduce*-predicate. □

**Theorem 4.5.** *The datalog program in Fig.* 5 *decides the Solvability problem of PAPs, i.e., the fact "success" is true in the minimal model of the interpreted program if and only if $\mathcal{A}_{td}$ encodes a solvable PAP $\mathcal{P} = \langle V, H, M, \mathcal{C}\rangle$ together with a tree decomposition $\mathcal{T}$ of the clause set $\mathcal{C}$. Moreover, for any PAP $\mathcal{P} = \langle V, H, M, \mathcal{C}\rangle$ and tree decomposition $\mathcal{T}$ of width at most $w$, the program can be evaluated in time $\mathcal{O}(2^{5*3^{w+2}} * |\mathcal{P}|)$.*

**Proof sketch.** By Lemma 4.4, the *solve*-predicate indeed has the meaning according to Property B. Thus, the rule with head *success* reads as follows: The PAP $\mathcal{P}$ has a solution $\mathcal{S}$ if and only if there exists a set $S \subseteq H(v)$ (i.e., $S = \mathcal{S} \cap H(v)$) and there exists an extension $\bar{S}$ of $S$ to $H(\mathcal{T}_v)$ (i.e., $\bar{S} = \mathcal{S}$; this extension is identified by the index $i$), s.t. the following conditions hold:

(1) $Cl(\mathcal{T}_v) \cup \bar{S}$ has a model, i.e., there exists an assignment $(P, N)$ on $Var(v)$ which can be extended to $J$ on $Var(\mathcal{T}_v)$, s.t. $Cl(\mathcal{T}_v) \cup \bar{S}$ is true in $J$, and

(2) $Cl(\mathcal{T}_v) \cup \bar{S}$ does not have a model in which some $m \in M$ is false, i.e., there does not exists an assignment $(P, N)$ on $Var(v)$ which can be extended to $J$ on $Var(\mathcal{T}_v)$, s.t. $Cl(\mathcal{T}_v) \cup \bar{S}$ is true in $J$ and at least one $m \in M$ is false in $J$.

As in the proof of Theorem 4.2, the upper bound on the time complexity is obtained via an upper bound on the possible ground instantiations of each rule in the Solvability program. First consider the $solve(v, S, i, P, N, C, d)$-predicates: Suppose that the bag at some node $v$ consists of $k$ variables and $l$ clauses with $k + l \leqslant w + 1$. Then we get the following upper bound on the possible value combinations:

For $(S, P, N)$, we have at most $3^k$ possibilities, i.e., a variable can either occur in $N$ (i.e., it is set to false) or both in $S$ and $P$ (i.e., it is set to true and added to the solution $S$) or in $P$ but not in $S$ (i.e., it is set to true and but not added to the solution $S$). For $C$, we have the upper bound $2^l$, and for $d$ there are at most 2 possible values. In total, the number of combinations of $(S, P, N, C, d)$ is bounded by $3^k * 2^l * 2 \leqslant 3^k * 3^l * 3 = 3^{w+2}$. For the possible values of $i$, recall from the above considerations that, for every pair $i, j$ of distinct "indices", the set of facts $solve(v, S, i, \ldots)$ and the set of facts $solve(v, S, j, \ldots)$ must be different. Hence, for every value of $v$ and $S$, there are at most $2^{3^{w+2}}$ possible values of $i$. In total, we thus get the upper bound $\mathcal{O}(3^{w+2} * 2^{3^{w+2}} * |\mathcal{P}|)$ on the possible ground instantiations of the $solve(v, S, i, P, N, C, d)$-predicate.

For the remaining predicates used in the program in Fig. 5 (in particular, the *aux*-predicate but also the remaining auxiliary predicates like the *reduce*-predicate detailed in Appendix B), the worst situation that can happen is that a predicate contains two indices $i, j$. Hence, the program in Fig. 5 is equivalent to a ground program with $\mathcal{O}(3^{w+2} * 2^{3^{w+2}} * 2^{3^{w+2}} * |\mathcal{P}|) = \mathcal{O}(2^{5*3^{w+2}} * |\mathcal{P}|)$ rules. The Solvability program can thus be evaluated by first computing the equivalent ground program

**Table 2**
Solvability Example (1).

| Node | *solve* facts |
|---|---|
| 7 | $solve(7, 0, \emptyset, \{x_3\}, \emptyset, \{c_1\}, \text{'n'})$, $solve(7, 0, \emptyset, \emptyset, \{x_3\}, \emptyset, \text{'n'})$ |
| 6 | $solve(6, 0, \emptyset, \{x_1, x_3\}, \emptyset, \{c_1\}, \text{'n'})$, $solve(6, 0, \emptyset, \{x_1\}, \{x_3\}, \{c_1\}, \text{'n'})$, $solve(6, 0, \emptyset, \{x_3\}, \{x_1\}, \{c_1\}, \text{'n'})$, $solve(6, 0, \emptyset, \emptyset, \{x_1, x_3\}, \emptyset, \text{'n'})$, $solve(6, 0, \{x_1\}, \{x_1, x_3\}, \emptyset, \{c_1\}, \text{'n'})$, $solve(6, 0, \{x_1\}, \{x_1\}, \{x_3\}, \{c_1\}, \text{'n'})$ |
| 5 | $solve(5, 0, \emptyset, \{x_1\}, \emptyset, \{c_1\}, \text{'n'})$, $solve(5, 0, \emptyset, \emptyset, \{x_1\}, \{c_1\}, \text{'n'})$, $solve(5, 0, \emptyset, \emptyset, \{x_1\}, \emptyset, \text{'n'})$, $solve(5, 0, \{x_1\}, \{x_1\}, \emptyset, \{c_1\}, \text{'n'})$ |
| 4 | $solve(4, 0, \emptyset, \{x_1, x_2\}, \emptyset, \{c_1\}, \text{'n'})$, $solve(4, 0, \emptyset, \{x_1\}, \{x_2\}, \{c_1\}, \text{'n'})$, $solve(4, 0, \{x_2\}, \{x_1, x_2\}, \emptyset, \{c_1\}, \text{'n'})$, $solve(4, 0, \emptyset, \{x_2\}, \{x_1\}, \{c_1\}, \text{'n'})$, $solve(4, 0, \emptyset, \emptyset, \{x_1, x_2\}, \{c_1\}, \text{'n'})$, $solve(4, 0, \{x_2\}, \{x_2\}, \{x_1\}, \{c_1\}, \text{'n'})$, $solve(4, 0, \emptyset, \{x_2\}, \{x_1\}, \emptyset, \text{'n'})$, $solve(4, 0, \{x_2\}, \{x_2\}, \{x_1\}, \emptyset, \text{'n'})$, $solve(4, 0, \{x_1\}, \{x_1, x_2\}, \emptyset, \{c_1\}, \text{'n'})$, $solve(4, 0, \{x_1\}, \{x_1\}, \{x_2\}, \{c_1\}, \text{'n'})$, $solve(4, 0, \{x_1, x_2\}, \{x_1, x_2\}, \emptyset, \{c_1\}, \text{'n'})$ |
| 3 | $solve(3, 0, \emptyset, \{x_1, x_2\}, \emptyset, \emptyset, \text{'n'})$, $solve(3, 0, \emptyset, \{x_1\}, \{x_2\}, \emptyset, \text{'n'})$, $solve(3, 0, \{x_2\}, \{x_1, x_2\}, \emptyset, \emptyset, \text{'n'})$, $solve(3, 0, \emptyset, \{x_2\}, \{x_1\}, \emptyset, \text{'n'})$, $solve(3, 0, \emptyset, \emptyset, \{x_1, x_2\}, \emptyset, \text{'n'})$, $solve(3, 0, \{x_2\}, \{x_2\}, \{x_1\}, \emptyset, \text{'n'})$, $solve(3, 0, \{x_1\}, \{x_1, x_2\}, \emptyset, \emptyset, \text{'n'})$, $solve(3, 0, \{x_1\}, \{x_1\}, \{x_2\}, \emptyset, \text{'n'})$, $solve(3, 0, \{x_1, x_2\}, \{x_1, x_2\}, \emptyset, \emptyset, \text{'n'})$ |
| 1 | $solve(1, 0, \emptyset, \{x_1, x_2, x_4\}, \emptyset, \emptyset, \text{'n'})$, $solve(1, 0, \emptyset, \{x_1, x_2\}, \{x_4\}, \emptyset, \text{'n'})$, $solve(1, 0, \emptyset, \{x_1, x_4\}, \{x_2\}, \emptyset, \text{'n'})$, $solve(1, 0, \emptyset, \{x_1\}, \{x_2, x_4\}, \emptyset, \text{'n'})$, $solve(1, 0, \{x_2\}, \{x_1, x_2, x_4\}, \emptyset, \emptyset, \text{'n'})$, $solve(1, 0, \{x_2\}, \{x_1, x_2\}, \{x_4\}, \emptyset, \text{'n'})$, $solve(1, 0, \emptyset, \{x_2, x_4\}, \{x_1\}, \emptyset, \text{'n'})$, $solve(1, 0, \emptyset, \{x_2\}, \{x_1, x_4\}, \emptyset, \text{'n'})$, $solve(1, 0, \emptyset, \{x_4\}, \{x_1, x_2\}, \emptyset, \text{'n'})$, $solve(1, 0, \emptyset, \emptyset, \{x_1, x_2, x_4\}, \emptyset, \text{'n'})$, $solve(1, 0, \{x_2\}, \{x_2, x_4\}, \{x_1\}, \emptyset, \text{'n'})$, $solve(1, 0, \{x_2\}, \{x_2\}, \{x_1, x_4\}, \emptyset, \text{'n'})$, $solve(1, 0, \{x_1\}, \{x_1, x_2, x_4\}, \emptyset, \emptyset, \text{'n'})$, $solve(1, 0, \{x_1\}, \{x_1, x_2\}, \{x_4\}, \emptyset, \text{'n'})$, $solve(1, 0, \{x_1\}, \{x_1, x_4\}, \{x_2\}, \emptyset, \text{'n'})$, $solve(1, 0, \{x_1\}, \{x_1\}, \{x_2, x_4\}, \emptyset, \text{'n'})$, $solve(1, 0, \{x_1, x_2\}, \{x_1, x_2, x_4\}, \emptyset, \emptyset, \text{'n'})$, $solve(1, 0, \{x_1, x_2\}, \{x_1, x_2\}, \{x_4\}, \emptyset, \text{'n'})$ |

and then evaluating the latter in linear time [18,42] w.r.t. the size of the ground program and the size of $\mathcal{A}_{td}$. Analogously to the proof of Theorem 4.2, all this is feasible within the time bound $\mathcal{O}(2^{5*3^{w+2}} * |\mathcal{P}|)$. □

**Example 4.6.** Consider the PAP $\mathcal{P} = \langle V, H, M, \mathcal{C} \rangle$ where $\mathcal{C}$ is the clause set corresponding to the CNF-formula $F$ from Example 2.1, i.e., $\mathcal{C} = \{(x_1 \vee \neg x_2 \vee x_3), (\neg x_1 \vee x_4 \vee \neg x_5), (x_2 \vee \neg x_4 \vee x_6)\}$. Additionally, we assume that $H = \{x_1, x_2\}$ and $M = \{x_5, x_6\}$. Finally, we have $V = \{x_1, x_2, x_3, x_4, x_5, x_6\}$. The tree decomposition $\mathcal{T}$ in Fig. 2 is also a tree decomposition of the structure representing this PAP. We shall run the Solvability program on this PAP. Again, we process the tree nodes of $\mathcal{T}$ in a bottom–up manner.

Table 2 shows the *solve*-facts derived at nodes 1, 3, 4, 5, 6 and 7. In principle, the processing is very similar to the SAT-program, except that we have to maintain the additional parameters $S$, $i$, and $d$. Below, we highlight some differences compared with the run of the SAT-program discussed in Example 4.3.

The number of *solve*-facts at the variable introduction node 6 is now three times the number of *solve*-facts at node 7. The reason for this is that we have 3 possibilities how to proceed with the new variable $x_1$ (which we assume to be a hypothesis): We can set $x_1$ to false (i.e., it is added to $N$) or we can set it to true without adding it to $S$ (i.e., it is added to $P$) or we can set it to true and also add it to $S$ (i.e., it is added to $P$ and $S$). Node 4 is a variable introduction node with new variable 4. Note that we only get 11 (rather than 12) *solve*-facts at node 4. The reason for this is that both facts, $solve(5, 0, \emptyset, \emptyset, \{x_1\}, \{c_1\}, \text{'n'})$ and $solve(5, 0, \emptyset, \emptyset, \{x_1\}, \emptyset, \text{'n'})$ give rise to the fact $solve(4, 0, \emptyset, \emptyset, \{x_1, x_2\}, \{c_1\}, \text{'n'})$. When traversing from node 4 to node 3, the number of *solve*-facts decreases from 11 to 9 due to the two *solve*-facts at node 4 where the parameter $C$ has the value $\emptyset$ (in particular, $C$ does not contain clause $c_1$ even though node 3 is a clause removal node with the removal of clause $c_1$). Finally, when traversing from node 3 to node 1, the number of *solve*-facts is multiplied by 2 (rather than by 3 as in node 6) since $x_4$ is not a hypothesis. Hence, there are only two possibilities how to proceed with $x_4$: We either add it to $P$ or to $N$. At tree node 1, we thus end up with 18 *solve*-facts.

The traversal from node 13 to 9 is illustrated in Table 3. This time we have a manifestation variable $x_5$, which has to be taken care of by the value of the parameter $d$. We leave out the illustration for the traversal from node 19 to 15, because it follows almost exactly the same pattern as that from node 13 to 9. We only present the *solve*-facts at node 15 (which are very similar to node 9). We very briefly discuss the derivation of the *solve*-facts at these nodes: At node 13, the $d$-value indicates if a manifestation (in this $x_5$) is set to false (i.e., $d =$ 'y') or not (i.e., $d =$ 'n'). The transition from node 13 to 12 is analogous to the transition from node 7 to 6 in Table 2. At the variable removal node 11, the number of *solve*-facts is not decreased compared with node 12, since we now also have to take the $d$-value into account. At the variable introduction node 10, we double the number of *solve*-facts (since the new variable $x_4$ is not a hypothesis). When traversing from node

**Table 3**
Solvability Example (2).

| Node | *solve* facts |
|---|---|
| 13 | $solve(13, 0, \emptyset, \emptyset, \{x_5\}, \{c_2\}, \text{'y'})$, $solve(13, 0, \emptyset, \{x_5\}, \emptyset, \emptyset, \text{'n'})$ |
| 12 | $solve(12, 0, \emptyset, \{x_1\}, \{x_5\}, \{c_2\}, \text{'y'})$, $solve(12, 0, \emptyset, \emptyset, \{x_1, x_5\}, \{c_2\}, \text{'y'})$, $solve(12, 0, \{x_1\}, \{x_1\}, \{x_5\}, \{c_2\}, \text{'y'})$, $solve(12, 0, \emptyset, \{x_1, x_5\}, \emptyset, \emptyset, \text{'n'})$, $solve(12, 0, \emptyset, \{x_5\}, \{x_1\}, \{c_2\}, \text{'n'})$, $solve(12, 0, \{x_1\}, \{x_1, x_5\}, \emptyset, \emptyset, \text{'n'})$ |
| 11 | $solve(11, 0, \emptyset, \{x_1\}, \emptyset, \{c_2\}, \text{'y'})$, $solve(11, 0, \emptyset, \emptyset, \{x_1\}, \{c_2\}, \text{'y'})$, $solve(11, 0, \{x_1\}, \{x_1\}, \emptyset, \{c_2\}, \text{'y'})$, $solve(11, 0, \emptyset, \{x_1\}, \emptyset, \emptyset, \text{'n'})$, $solve(11, 0, \emptyset, \emptyset, \{x_1\}, \{c_2\}, \text{'n'})$, $solve(11, 0, \{x_1\}, \{x_1\}, \emptyset, \emptyset, \text{'n'})$ |
| 10 | $solve(10, 0, \emptyset, \{x_1, x_4\}, \emptyset, \{c_2\}, \text{'y'})$, $solve(10, 0, \emptyset, \{x_1\}, \{x_4\}, \{c_2\}, \text{'y'})$, $solve(10, 0, \emptyset, \{x_4\}, \{x_1\}, \{c_2\}, \text{'y'})$, $solve(10, 0, \emptyset, \emptyset, \{x_1, x_4\}, \{c_2\}, \text{'y'})$, $solve(10, 0, \{x_1\}, \{x_1, x_4\}, \emptyset, \{c_2\}, \text{'y'})$, $solve(10, 0, \{x_1\}, \{x_1\}, \{x_4\}, \{c_2\}, \text{'y'})$, $solve(10, 0, \emptyset, \{x_1, x_4\}, \emptyset, \{c_2\}, \text{'n'})$, $solve(10, 0, \emptyset, \{x_1\}, \{x_4\}, \emptyset, \text{'n'})$, $solve(10, 0, \emptyset, \{x_4\}, \{x_1\}, \{c_2\}, \text{'n'})$, $solve(10, 0, \emptyset, \emptyset, \{x_1, x_4\}, \{c_2\}, \text{'n'})$, $solve(10, 0, \{x_1\}, \{x_1, x_4\}, \emptyset, \{c_2\}, \text{'n'})$, $solve(10, 0, \{x_1\}, \{x_1\}, \{x_4\}, \emptyset, \text{'n'})$ |
| 9 | $solve(9, 0, \emptyset, \{x_1, x_4\}, \emptyset, \emptyset, \text{'y'})$, $solve(9, 0, \emptyset, \{x_1\}, \{x_4\}, \emptyset, \text{'y'})$, $solve(9, 0, \emptyset, \{x_4\}, \{x_1\}, \emptyset, \text{'y'})$, $solve(9, 0, \emptyset, \emptyset, \{x_1, x_4\}, \emptyset, \text{'y'})$, $solve(9, 0, \{x_1\}, \{x_1, x_4\}, \emptyset, \emptyset, \text{'y'})$, $solve(9, 0, \{x_1\}, \{x_1\}, \{x_4\}, \emptyset, \text{'y'})$, $solve(9, 0, \emptyset, \{x_1, x_4\}, \emptyset, \emptyset, \text{'n'})$, $solve(9, 0, \emptyset, \{x_4\}, \{x_1\}, \emptyset, \text{'n'})$, $solve(9, 0, \emptyset, \emptyset, \{x_1, x_4\}, \emptyset, \text{'n'})$, $solve(9, 0, \{x_1\}, \{x_1, x_4\}, \emptyset, \emptyset, \text{'n'})$ |
| 15 | $solve(15, 0, \emptyset, \{x_2, x_4\}, \emptyset, \emptyset, \text{'n'})$, $solve(15, 0, \emptyset, \{x_2\}, \{x_4\}, \emptyset, \text{'n'})$, $solve(15, 0, \emptyset, \{x_4\}, \{x_2\}, \emptyset, \text{'n'})$, $solve(15, 0, \emptyset, \emptyset, \{x_2, x_4\}, \emptyset, \text{'n'})$, $solve(15, 0, \{x_2\}, \{x_2, x_4\}, \emptyset, \emptyset, \text{'n'})$, $solve(15, 0, \{x_2\}, \{x_2\}, \{x_4\}, \emptyset, \text{'n'})$, $solve(15, 0, \emptyset, \{x_2, x_4\}, \emptyset, \emptyset, \text{'y'})$, $solve(15, 0, \emptyset, \{x_2\}, \{x_4\}, \emptyset, \text{'y'})$, $solve(15, 0, \emptyset, \emptyset, \{x_2, x_4\}, \emptyset, \text{'y'})$, $solve(15, 0, \{x_2\}, \{x_2\}, \{x_4\}, \emptyset, \text{'y'})$, $solve(15, 0, \{x_2\}, \{x_2, x_4\}, \emptyset, \emptyset, \text{'y'})$ |

10 to node 9, the number of *solve*-facts is reduced by 2 since there are two *solve*-facts in node 10 with parameter value $C = \emptyset$. Node 15 is very similar to node 9; the role of $x_6$, $x_2$, and $c_3$ on the path from 19 up to 15 essentially corresponds to the role of $x_5$, $x_1$, and $c_2$ on the path from 13 up to 9. The most significant difference between these two paths is that $x_1$ and $x_5$ occur in $c_2$ negatively while $x_2$ and $x_6$ occur in $c_3$ positively. By the different sign of $x_5$ and $x_6$, the $d$-values 'y' and 'n' are swapped when we compare the *solve*-facts at node 9 with the *solve*-facts at node 15. The different sign of $x_1$ and $x_2$ has the effect that there is one more *solve*-facts at node 15 than at node 9. The reason for this is, that at node 15, only the fact $solve(15, 0, \emptyset, \{x_4\}, \{x_2\}, \emptyset, \text{'y'})$ is missing (i.e., if $x_4$ and $x_6$ are set to true and $x_2$ to false, then $c_3$ evaluates to false). On the other hand, at node 9, the two facts $solve(9, 0, \emptyset, \{x_1\}, \{x_4\}, \emptyset, \text{'n'})$ and $solve(9, 0, \{x_1\}, \{x_1\}, \{x_4\}, \emptyset, \text{'n'})$ are missing (i.e., when $x_1$ is set to true and $x_4$ and $x_5$ are set to false, then $c_2$ evaluates to false; since the hypothesis $x_1$ is true, it can be either added to $S$ or not).

The *solve*-facts at node 8 and 14 are presented in Table 4. We obtain these facts from the solve facts at node 9 (respectively 19) and allowing for $x_2$ (respectively $x_1$) the three possible cases that this new variable is added to $P$ only or $N$ only or to both $N$ and $P$ and $S$. We do not explicitly present the facts at the root node. However, it can now be easily verified that the *success*-fact cannot be derived because the *solve*$'$-predicate will be true for any value of $(i, S)$. Note that $i = 0$ in all *solve*-facts at the root, since all hypotheses appear in the bag at node 0. Hence, for every value of $S$ at node 0, there exists exactly 1 extension to the hypotheses at bags below node 0 (namely $S$ itself). Intuitively, the *solve*$'$-predicate is true for any value of $(i, S)$ with $i = 0$ and $S \in \{\emptyset, \{x_1\}, \{x_2\}, \{x_1, x_2\}\}$, because any truth assignment on $\{x_1, x_2, x_4\}$ can be extended to the remaining variables by setting $x_3$ and $x_6$ to true and $x_5$ to false, s.t. all clauses in $\mathcal{C}$ are true but one manifestation (namely $x_5$) is false.

Formally, this property can be seen as follows: Note that there are 18 facts in node 8 where the $d$-value is 'y'. These 18 facts correspond to all possible values of the parameters $S, P, N$ for the variables $x_1, x_2$, and $x_4$. The $d$-value indicates that on the considered extensions of these truth assignments, some manifestation (namely $x_5$) is assigned the truth value false. At node 2, through the datalog rule for branch nodes, we will again get 18 facts with all possible assignments of $x_1, x_2$ and $x_4$, whose $d$-value is 'y'. This same set of value combinations will also be present in the *solve*-facts at the root node 0. But then, for every value of $S$, the *solve*$'$-predicate will be true. Thus, the *success*-fact cannot be derived by the program and we conclude that the PAP $\mathcal{P}$ has no solution.

### 4.3. Relevance enumeration problem

The problem of computing all relevant hypotheses can be clearly expressed as a unary MSO-query and, thus, by a monadic datalog program. Indeed, it is straightforward to extend our Solvability program to a program for the Relevance

**Table 4**
Solvability Example (3).

| Node | *solve* facts |
|---|---|
| 8 | *solve*(8, 0, ∅, {$x_1, x_2, x_4$}, ∅, ∅, 'y'), *solve*(8, 0, ∅, {$x_1, x_4$}, {$x_2$}, ∅, 'y'),<br>*solve*(8, 0, {$x_2$}, {$x_1, x_2, x_4$}, ∅, ∅, 'y'), *solve*(8, 0, ∅, {$x_1, x_2$}, {$x_4$}, ∅, 'y'),<br>*solve*(8, 0, ∅, {$x_1$}, {$x_2, x_4$}, ∅, 'y'), *solve*(8, 0, {$x_2$}, {$x_1, x_2$}, {$x_4$}, ∅, 'y'),<br>*solve*(8, 0, ∅, {$x_2, x_4$}, {$x_1$}, ∅, 'y'), *solve*(8, 0, ∅, {$x_4$}, {$x_1, x_2$}, ∅, 'y'),<br>*solve*(8, 0, {$x_2$}, {$x_2, x_4$}, {$x_1$}, ∅, 'y'), *solve*(8, 0, ∅, {$x_2$}, {$x_1, x_4$}, ∅, 'y'),<br>*solve*(8, 0, ∅, ∅, {$x_1, x_2, x_4$}, ∅, 'y'), *solve*(8, 0, {$x_2$}, {$x_2$}, {$x_1, x_4$}, ∅, 'y'),<br>*solve*(8, 0, {$x_1$}, {$x_1, x_2, x_4$}, ∅, ∅, 'y'), *solve*(8, 0, {$x_1$}, {$x_1, x_4$}, {$x_2$}, ∅, 'y'),<br>*solve*(8, 0, {$x_1, x_2$}, {$x_1, x_2, x_4$}, ∅, ∅, 'y'), *solve*(8, 0, {$x_1$}, {$x_1, x_2$}, {$x_4$}, ∅, 'y'),<br>*solve*(8, 0, {$x_1$}, {$x_1$}, {$x_2, x_4$}, ∅, 'y'), *solve*(8, 0, {$x_1, x_2$}, {$x_1, x_2$}, {$x_4$}, ∅, 'y'),<br>*solve*(8, 0, ∅, {$x_1, x_2, x_4$}, ∅, ∅, 'n'), *solve*(8, 0, ∅, {$x_1, x_4$}, {$x_2$}, ∅, 'n'),<br>*solve*(8, 0, {$x_2$}, {$x_1, x_2, x_4$}, ∅, ∅, 'n'), *solve*(8, 0, ∅, {$x_2, x_4$}, {$x_1$}, ∅, 'n'),<br>*solve*(8, 0, ∅, {$x_4$}, {$x_1, x_2$}, ∅, 'n'), *solve*(8, 0, {$x_2$}, {$x_2, x_4$}, {$x_1$}, ∅, 'n'),<br>*solve*(8, 0, ∅, {$x_2$}, {$x_1, x_4$}, ∅, 'n'), *solve*(8, 0, ∅, ∅, {$x_1, x_2, x_4$}, ∅, 'n'),<br>*solve*(8, 0, {$x_2$}, {$x_2$}, {$x_1, x_4$}, ∅, 'n'), *solve*(8, 0, {$x_1$}, {$x_1, x_2, x_4$}, ∅, ∅, 'n'),<br>*solve*(8, 0, {$x_1$}, {$x_1, x_4$}, {$x_2$}, ∅, 'n'), *solve*(8, 0, {$x_1, x_2$}, {$x_1, x_2, x_4$}, ∅, ∅, 'n') |
| 14 | *solve*(14, 0, ∅, {$x_1, x_2, x_4$}, ∅, ∅, 'n'), *solve*(14, 0, ∅, {$x_2, x_4$}, {$x_1$}, ∅, 'n'),<br>*solve*(14, 0, {$x_1$}, {$x_1, x_2, x_4$}, ∅, ∅, 'n'), *solve*(14, 0, ∅, {$x_1, x_2$}, {$x_4$}, ∅, 'n'),<br>*solve*(14, 0, ∅, {$x_2$}, {$x_1, x_4$}, ∅, 'n'), *solve*(14, 0, {$x_1$}, {$x_1, x_2$}, {$x_4$}, ∅, 'n'),<br>*solve*(14, 0, ∅, {$x_1, x_4$}, {$x_2$}, ∅, 'n'), *solve*(14, 0, ∅, {$x_4$}, {$x_1, x_2$}, ∅, 'n'),<br>*solve*(14, 0, {$x_1$}, {$x_1, x_4$}, {$x_2$}, ∅, 'n'), *solve*(14, 0, ∅, {$x_1$}, {$x_2, x_4$}, ∅, 'n'),<br>*solve*(14, 0, ∅, ∅, {$x_1, x_2, x_4$}, ∅, 'n'), *solve*(14, 0, {$x_1$}, {$x_1$}, {$x_2, x_4$}, ∅, 'n'),<br>*solve*(14, 0, {$x_2$}, {$x_1, x_2, x_4$}, ∅, ∅, 'n'), *solve*(14, 0, {$x_2$}, {$x_2, x_4$}, {$x_1$}, ∅, 'n'),<br>*solve*(14, 0, {$x_1, x_2$}, {$x_1, x_2, x_4$}, ∅, ∅, 'n'), *solve*(14, 0, {$x_2$}, {$x_1, x_2$}, {$x_4$}, ∅, 'n'),<br>*solve*(14, 0, {$x_2$}, {$x_2$}, {$x_1, x_4$}, ∅, 'n'), *solve*(14, 0, {$x_1, x_2$}, {$x_1, x_2$}, {$x_4$}, ∅, 'n'),<br>*solve*(14, 0, ∅, {$x_1, x_2, x_4$}, ∅, ∅, 'y'), *solve*(14, 0, ∅, {$x_2, x_4$}, {$x_1$}, ∅, 'y')<br>*solve*(14, 0, {$x_1$}, {$x_1, x_2, x_4$}, ∅, ∅, 'y'), *solve*(14, 0, ∅, {$x_1, x_2$}, {$x_4$}, ∅, 'y')<br>*solve*(14, 0, ∅, {$x_2$}, {$x_1, x_4$}, ∅, 'y'), *solve*(14, 0, {$x_1$}, {$x_1, x_2$}, {$x_4$}, ∅, 'y'),<br>*solve*(14, 0, ∅, {$x_1$}, {$x_2, x_4$}, ∅, 'y'), *solve*(14, 0, ∅, ∅, {$x_1, x_2, x_4$}, ∅, 'y'),<br>*solve*(14, 0, {$x_1$}, {$x_1$}, {$x_2, x_4$}, ∅, 'y'), *solve*(14, 0, {$x_2$}, {$x_1, x_2$}, {$x_4$}, ∅, 'y'),<br>*solve*(14, 0, {$x_2$}, {$x_2$}, {$x_1, x_4$}, ∅, 'y'), *solve*(14, 0, {$x_1, x_2$}, {$x_1, x_2$}, {$x_4$}, ∅, 'y'),<br>*solve*(14, 0, {$x_2$}, {$x_1, x_2, x_4$}, ∅, ∅, 'y'), *solve*(14, 0, {$x_2$}, {$x_2, x_4$}, {$x_1$}, ∅, 'y'),<br>*solve*(14, 0, {$x_1, x_2$}, {$x_1, x_2, x_4$}, ∅, ∅, 'y') |

enumeration problem: Suppose that some hypothesis $h$ occurs in the bag of the root $r$ of $\mathcal{T}$. Then $h$ is relevant if and only if there exists a subset $S \subseteq H(r)$ and an index $i$, s.t. $h \in S$ and $S$ can be extended to a solution $\bar{S}_i \subseteq H(\mathcal{T}_r)$ of the PAP $\mathcal{P}$. Naively, one can compute all relevant hypotheses by considering the tree decomposition $\mathcal{T}$ as rooted at various nodes, s.t. each $h \in H$ is contained in the bag of at least one such root node. Obviously, this method has *quadratic* time complexity w.r.t. the data size.

However, one can do better by computing the *solve*-facts at each node $v$ in $\mathcal{T}$ simultaneously both for a bottom–up traversal of $\mathcal{T}$ and for a top–down traversal of $\mathcal{T}$ (by means of a new predicate *solve* $\downarrow$). The tree decomposition can of course be modified in such a way that every hypothesis $h \in H$ occurs in at least one leaf node of $\mathcal{T}$. Moreover, for every branch node $v$ in the tree decomposition, we insert a new node $u$ as new parent of $v$, s.t. $u$ and $v$ have identical bags. Hence, together with the two child nodes of $v$, each branch node is "surrounded" by three neighboring nodes with identical bags. It is thus guaranteed that a branch node always has two child nodes with identical bags – no matter where $\mathcal{T}$ is rooted.

Then the relevant hypotheses can be obtained via the *solve* $\downarrow$ $(v, \ldots)$ facts which are true in the minimal model of the interpreted program for all leaf nodes $v$ of $\mathcal{T}$ (since these facts correspond precisely to the *solve*$(v, \ldots)$ facts if $\mathcal{T}$ were rooted at $v$). The details of the additional datalog rules required compared with the Solvability test are given in Fig. 6. Of course, we now also need a new predicate *aux* $\downarrow$, which plays the analogous role for the definition of the *solve* $\downarrow$-predicate as the *aux*-predicate in Fig. 5 did for the definition of *solve*. Further analogies between the definition of *solve* $\downarrow$ and *solve*concern the various node types: For a top–down traversal, the root node plays the role that the leaf node formerly played. The child node of a variable (respectively clause) introduction node plays in the top–down traversal the role of a variable (respectively clause) removal node in the bottom–up traversal, i.e.: the bag at such a node has one variable (respectively clause) less than the bag of the previously visited node. Likewise, the child node of a variable (respectively clause) removal node plays in the top–down traversal the role of a variable (respectively clause) introduction node. The only tricky case is that of a branch node. Note that our program in Fig. 6 does not derive any *solve* $\downarrow$-facts for the branch nodes themselves. For deriving the *solve* D-facts at the child nodes of a branch node, we distinguish two cases: If the top–down traversal continues with the first child $v$ of a branch node $b$, then the parent of $b$ plays the role of $v_1$ and the second child of $b$ plays the role of $v_2$. Care has to be taken that the *solve* $\downarrow$-predicate at node $v$ is defined in terms of the *solve* $\downarrow$-predicate at the parent of $b$ but in terms of the *solve* $\downarrow$-predicate at the second child of $b$. Analogously, the *solve* $\downarrow$-predicate at $v$ is defined if $v$ is the second child of a branch node.

**Program** Relevance

/* root node. */
$solve\downarrow(v,S,0,P,N,C_1,d) \leftarrow root(v),\ bag(v,X,C),\ svar(v,S),\ S\subseteq P,$
$P\cup N=X,\ P\cap N=\emptyset,\ check(P,N,C_1,C,d).$

/* child of variable introduction node. */
$aux\downarrow(v,S,i,0,P,N,C_1,d) \leftarrow bag(v,X,C),\ child_1(v,v_1),\ bag(v_1,X\uplus\{x\},C),$
$solve\downarrow(v_1,S,i,P\uplus\{x\},N,C_1,d),\ x\notin S.$

$aux\downarrow(v,S,i,0,P,N,C_1,d) \leftarrow bag(v,X,C),\ child_1(v_1,v),\ bag(v_1,X\uplus\{x\},C),$
$solve\downarrow(v_1,S,i,P,N\uplus\{x\},C_1,d).$

$aux\downarrow(v,S,i,1,P,N,C_1,d) \leftarrow bag(v,X,C),\ child_1(v_1,v),\ bag(v_1,X\uplus\{x\},C),$
$solve\downarrow(v_1,S\uplus\{x\},i,P\uplus\{x\},N,C_1,d).$

/* child of clause introduction node. */
$solve\downarrow(v,S,i,P,N,C_1,d) \leftarrow bag(v,X,C),\ child_1(v,v_1),\ bag(v_1,X,C\uplus\{c\}),$
$solve\downarrow(v_1,S,i,P,N,C_1\uplus\{c\},d).$

/* child of variable removal node. */
$solve\downarrow(v,S,i,P\uplus\{x\},N,C_1\cup C_2,d_1) \leftarrow bag(v,X\uplus\{x\},C),\ child_1(v,v_1),$
$bag(v_1,X,C),\ solve\downarrow(v_1,S,i,P,N,C_1,d_1),\ check(\{x\},\emptyset,C_2,C,d_2).$

$solve\downarrow(v,S,i,P,N\uplus\{x\},C_1\cup C_2,d_1 \text{ or } d_2) \leftarrow bag(v,X\uplus\{x\},C),\ child_1(v,v_1),$
$bag(v_1,X,C),\ solve\downarrow(v_1,S,i,P,N,C_1,d_1),\ check(\emptyset,\{x\},C_2,C,d_2).$

$solve\downarrow(v,S\uplus\{x\},i,P\uplus\{x\},N,C_1\cup C_2,d_1) \leftarrow bag(v,X\uplus\{x\},C),\ child_1(v,v_1),$
$bag(v_1,X,C),\ solve\downarrow(v_1,S,i,P,N,C_1,d_1),\ check(\{x\},\emptyset,C_2,C,d_2),\ hyp(x).$

/* child of clause removal node. */
$solve\downarrow(v,S,i,P,N,C_1\cup C_2,d_1) \leftarrow bag(v,X,C\uplus\{c\}),\ child_1(v,v_1),$
$bag(v_1,X,C),\ solve\downarrow(v_1,S,i,P,N,C_1,d_1),\ check(P,N,C_2,\{c\},d_2).$

/* first child of branch node. */
$aux\downarrow(v,S,i_1,i_2,P,N,C_1\cup C_2,d_1 \text{ or } d_2) \leftarrow bag(v,X,C),$
$child_1(v,b),\ child_1(b,v_1),\ bag(v_1,X,C),\ child_2(v_2,b),\ bag(v_2,X,C),$
$solve\downarrow(v_1,S,i_1,P,N,C_1,d_1),\ solve(v_2,S,i_2,P,N,C_2,d_2).$

/* second child of branch node. */
$aux\downarrow(v,S,i_1,i_2,P,N,C_1\cup C_2,d_1 \text{ or } d_2) \leftarrow bag(v,X,C),$
$child_2(v,b),\ child_1(b,v_1),\ bag(v_1,X,C),\ child_1(b,v_2),\ bag(v_2,X,C),$
$solve\downarrow(v_1,S,i_1,P,N,C_1,d_1),\ solve(v_2,S,i_2,P,N,C_2,d_2).$

/* child nodes of variable introduction and branch nodes: $aux\Rightarrow solve$ */
$solve\downarrow(v,S,i,P,N,C,d) \leftarrow aux\downarrow(v,S,i_1,i_2,P,N,C,d),\ reduce\downarrow(v,S,i,i_1,i_2).$

/* result (at leaf nodes). */
$relevant(h) \leftarrow leaf(v),\ bag(v,X,C),\ solve\downarrow(v,S,i,P,N,C,\text{'n'}),$
$\text{not } solve'\downarrow(v,S,i),\ h\in S.$

$solve'\downarrow(v,S,i) \leftarrow bag(v,X,C),\ solve\downarrow(v,S,i,P,N,C,\text{'y'}).$

**Fig. 6.** Relevance enumeration program.

The resulting algorithm works in linear time since it essentially just doubles the computational effort of the Solvability program. In total, we thus get the following result.

**Theorem 4.7.** *The datalog program in Fig.* 6 *solves the Relevance enumeration problem of PAPs, i.e.: Suppose that the program is applied to a $\tau_{td}$-structure $\mathcal{A}_{td}$ encoding a PAP $\mathcal{P}=\langle V,H,M,\mathcal{C}\rangle$ together with a tree decomposition $\mathcal{T}$ of the clause set $\mathcal{C}$. Then, for every $h\in H$, the fact relevant(h) is true in the minimal model of this program and the input $\tau_{td}$-structure $\mathcal{A}_{td}$ if and only if the hypothesis h is relevant in the PAP $\mathcal{P}$. Moreover, for any PAP $\mathcal{P}=\langle V,H,M,\mathcal{C}\rangle$ and tree decomposition $\mathcal{T}$ of width at most w, the program can be evaluated in time $\mathcal{O}(2^{5*3^{w+2}}*|\mathcal{P}|)$.*

Note that in all programs presented in Section 4, we consider the tree decomposition as part of the input. It has already been mentioned in Section 2.1 that, in theory, for every given value $w\geqslant 1$, it can be decided in linear time (w.r.t. the size of the input structure), if some structure has treewidth at most $w$. Moreover, in case of a positive answer, a tree decomposition of width $w$ can also be computed in linear time, see [8]. We have also mentioned in Section 2.1, that the practical usefulness of this linearity is limited due to excessively big constants [39]. At any rate, the improvement of tree decomposition algorithms is an area of very active research and considerable progress has recently been made in developing heuristic-based tree decomposition algorithms [39,9,49,10].

**Table 5**
Processing time in ms for the SAT problem.

| tw | #V | #Cl | #tn | MD (ms) | MiniSat (ms) |
|---|---|---|---|---|---|
| 3 | 5 | 9 | 24 | 0.04 | < 10 |
| 3 | 31 | 48 | 195 | 0.2 | < 10 |
| 3 | 347 | 522 | 2157 | 0.8 | < 10 |
| 3 | 3955 | 5934 | 23 793 | 8.1 | < 10 |
| 3 | 32 765 | 49 149 | 207 438 | 65.5 | 40 |
| 3 | 337 859 | 506 790 | 2 120 361 | 643.9 | 430 |

**Table 6**
Processing time in ms for the Solvability problem.

| tw | #H | #M | #V | #Cl | #tn | MD | MONA |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 1 | 3 | 1 | 3 | 0.3 | 870 |
| 3 | 2 | 2 | 6 | 2 | 12 | 0.5 | 1710 |
| 3 | 3 | 3 | 9 | 3 | 21 | 0.6 | 12 160 |
| 3 | 4 | 4 | 12 | 4 | 34 | 0.9 | – |
| 3 | 7 | 7 | 21 | 7 | 69 | 1.5 | – |
| 3 | 11 | 11 | 33 | 11 | 105 | 2.3 | – |
| 3 | 15 | 15 | 45 | 15 | 141 | 2.9 | – |
| 3 | 19 | 19 | 57 | 19 | 193 | 3.9 | – |
| 3 | 23 | 23 | 69 | 23 | 229 | 4.7 | – |
| 3 | 27 | 27 | 81 | 27 | 265 | 5.3 | – |
| 3 | 31 | 31 | 93 | 31 | 301 | 6.1 | – |

## 5. Implementation and experimental results

### 5.1. Datalog approach

To test the performance and, in particular, the scalability of our new algorithms, we have implemented our SAT and Solvability programs in C++.

The SAT experiments were conducted on a PC with Intel(R) Pentium(R) D 3.00 GHz CPU, 1 GB of main memory, 2 GB of virtual memory, running Linux (Ubuntu) with kernel 2.6.22-14-generic. The experiments of the Solvability program were conducted on Linux kernel 2.6.17 with a 1.60 GHz Intel Pentium(M) processor and 512 MB of memory. We measured the processing time on different input parameters such as the number of variables, clauses, hypotheses, and manifestations. The treewidth in all the test cases was 3.

Due to the lack of available test data, we generated a balanced normalized tree decomposition and test data sets with increasing input parameters by expanding the tree in a breadth-first style. We have ensured that all different kinds of nodes occur evenly in the tree decomposition.

We compared the performance of our program with the open-source SAT solver MiniSat, see http://minisat.se/. The outcome of the tests for the SAT problem are shown in Table 5. In the table, #V, #Cl and #tn represent the number of variables, the number of clauses and the number of tree nodes, respectively. Note that the time measurement for MiniSat is restricted to the unit of seconds, thus the time consumption under 10 ms could not be obtained.

The results show that, for low treewidth of the underlying formulae, the performance of our method is comparable with an up-to-date SAT solver – with the SAT solver being faster by a factor of approximately 1.5 in most cases. Of course, this comparison should take into account that the current SAT solvers have evolved from intensive research and numerous competitions over 20 years, whereas our system is just a prototype implementation. Moreover, it has been recently observed that SAT solvers actually do take advantage of low treewidth without being specifically tuned for such instances, see [4]. The potential of our datalog-based approach is therefore not to be seen as a competitor of up-to-date SAT solvers but rather as a method that is applicable to problems that go far beyond SAT. In particular, our method is not restricted to NP-problems. This is illustrated by the application to abduction in our paper or to circumscription and disjunctive logic programming in [36,37]. Without the restriction to bounded treewidth, all these problems lie on the second level of the polynomial hierarchy.

The outcome of the tests for the Solvability problem is shown in Table 6 where tw stands for the treewidth; #H, #M, #V, #Cl and #tn stand for the number of hypotheses, manifestations, all variables, clauses and tree nodes, respectively. The processing time (in ms) obtained with our implementation of the monadic datalog approach are displayed in the column labeled “MD”. The measurements nicely reflect an essentially linear increase of the processing time with the size of the input. Moreover, there is obviously no “hidden” constant which would render the linearity useless.

We also wanted to compare our implementation with the standard MSO-to-FTA approach [3,25]. For this purpose, we carried out some experiments with a prototype implementation using MONA [38] for the MSO model checking. The time measurements of these experiments are shown in the last column of Table 6. Due to problems discussed in Section 5.2, MONA does not ensure linear data complexity. Hence, all tests below line 3 of the table failed with “out-of-memory” errors.

**General Generic MSO-to-FTA approach**
**Input:** MSO-formula $\varphi$, treewidth $w$, structure $\mathcal{A}$
**Output:** if $tw(\mathcal{A}) \leqslant w$ then output "Yes" / "No", else output "$tw(\mathcal{A}) > w$".

*Transformation of the MSO-formula $\varphi$.*
I. from $\varphi$ and $w$, compute the MSO-formula $\varphi^*$;

*Computation of a colored, binary tree $\mathcal{T}^*$.*
II. compute a tree decomposition $\mathcal{T}$ of $\mathcal{A}$ with width $w$;
if $tw(\mathcal{A}) > w$ then HALT with result "$tw(\mathcal{A}) > w$";
III. from $\mathcal{A}$ and $\mathcal{T}$, compute a colored, binary tree $\mathcal{T}^*$;

*MSO-model checking.*
IV. check if $\varphi^*$ evaluates to true over $\mathcal{T}^*$:
(a) compute an FTA equivalent to $\varphi^*$;
(b) check if the FTA accepts the tree $\mathcal{T}^*$;

**Fig. 7.** Generic MSO-to-FTA approach.

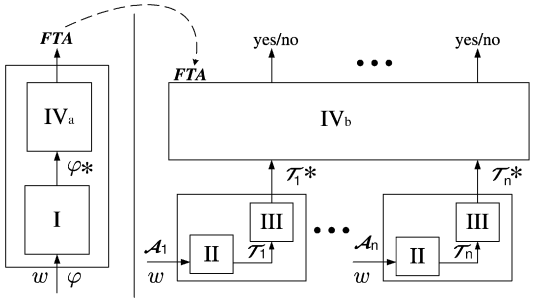


**Fig. 8.** System architecture of MSO-to-FTA approach.

Moreover, also in cases where the exponential data complexity does not yet "hurt", our datalog approach outperforms the MSO-to-FTA approach by a factor of 100 or even more.

### 5.2. MSO-to-FTA approach

Recipes to devise concrete algorithms based on Courcelle's Theorem can be found in the literature, see e.g. [3,12,11,25]. The principal steps of these approaches are described in Fig. 7.

The problem of evaluating an MSO-formula $\varphi$ over a finite structure $\mathcal{A}$ of treewidth at most $w$ is transformed into an equivalent problem of evaluating an MSO-formula $\varphi^*$ over colored, binary trees $\mathcal{T}^*$. To this end, the formula $\varphi^*$, which depends on the original formula $\varphi$ and the fixed treewidth $w$, has to be computed (step I). The computation of the colored, binary tree $\mathcal{T}^*$ proceeds in two steps: First, a tree decomposition $\mathcal{T}$ of the input structure $\mathcal{A}$ with width $w$ is computed (step II). Then a colored, binary tree $\mathcal{T}^*$ corresponding to $\mathcal{A}$ and $\mathcal{T}$ is computed (step III). In particular, $\mathcal{T}^*$ has the same tree structure as $\mathcal{T}$. The final step is the actual model-checking (step IV). The problem of evaluating the MSO-formula over the tree $\mathcal{T}^*$ is thus reduced to an equivalent tree language recognition problem. To this end, the formula $\varphi^*$ is transformed into an equivalent FTA (step IVa). Then it is checked if $\mathcal{T}^*$ is accepted by this FTA (step IVb).

We have implemented the Solvability problem of propositional abduction based on the MSO-formula $\varphi$ in the proof of Theorem 3.10. This MSO-formula was transformed into $\varphi^*$ in an ad hoc manner – depending on the chosen treewidth $w$. Rather than computing explicitly the tree decomposition $\mathcal{T}$, we simply considered the tree decomposition as an additional part of the input. Our transformation of $\mathcal{T}$ into $\mathcal{T}^*$ essentially implements the algorithm from [25] – apart from some simplifications which are possible here (e.g., due to the fact that we have no predicates over sets of constants in the input database). For the last step, we decided to take advantage of an existing MSO model checking tool, namely MONA [38]. To the best of our knowledge, MONA is the only existing such tool.

Experimental results with our prototype implementation on several instances of the Solvability problem are reported in the last column of Table 6. The observed runtime is in sharp contrast to the linear time behavior according to the theoretical results shown in Section 3. An analysis of the various components of our prototype has revealed that the way how MONA evaluates an MSO-formula $\varphi^*$ over a tree $\mathcal{T}^*$ is very problematical in that it combines the MSO-formula $\varphi^*$ and (a representation of) the tree $\mathcal{T}^*$ to a single formula whose validity is then checked. The correct way of handling this model checking problem according to [25] is depicted in Fig. 8. The steps I, II, and III are clear. However, it is crucial to divide step IV into two substeps: step IVa, which computes a finite tree automaton from the MSO-formula $\varphi^*$, and step IVb, which runs the FTA on every tree $\mathcal{T}^*$. The important point to notice is that steps I and IVa (depicted on the left-hand side of Fig. 8) are carried out once and for all while the steps II, III, and IVb (shown on the right-hand side) are repeated for every single input structure $\mathcal{A}_i$.

In contrast, MONA also considers the ground atoms encoding $\mathcal{T}^*$ as part of the formula and then compiles the resulting MSO-formula (consisting of $\varphi^*$ and the encoding of $\mathcal{T}^*$) into an FTA. In other words, the size of the FTA grows (exponentially!) with the size of $\mathcal{T}^*$. Consequently, on the one hand, the runtime needed by MONA is exponential. On the other hand, the state explosion of the FTA led to an "out-of-memory" error of MONA already for very small problem sizes. In the experiments described in Table 6, this negative effect already happened for problem instances with 12 variables and 4 clauses.

In fact, we have also used the algorithm of transforming the MSO formulae together with the ground atoms of the finite structure into finite tree automata. Surprisingly, the transformation failed with an "out-of-memory" error. This is counter-intuitive because the formula itself should is clearly smaller than the formula together with the ground atoms of the structure. According to our test results, it worked at least for instances with small sizes. The reason was that, with a very small set of ground atoms, the optimization module of MONA managed to remove many useless states during the transformation. However, when using MONA with the MSO-formula only, the optimization approach seems to be not in effect.

We therefore made another attempt following the MSO-to-FTA approach by implementing step IVa in Fig. 8 ourselves. Alas, this attempt led to failure yet before we were able to feed any input data to the program. The principal problem with the MSO-to-FTA approach is the "state explosion" of the resulting FTA (cf. [26,41]), which tends to occur even if one wants to evaluate comparatively simple MSO-formulae over trees. The situation gets even worse if we consider MSO *on structures with bounded treewidth*, since the original (possibly simple) MSO-formula $\varphi$ over a finite structure first has to be transformed into an equivalent MSO-formula $\varphi^*$ over trees. Hence, this transformation (e.g., by the algorithm in [25]) leads to a much more complex formula (in general, even with additional quantifier alternations) than the original formula. In summary, we have come to the same conclusion as Grohe in [32], namely, that the algorithms derived via Courcelle's Theorem are "useless for practical applications" and that the main benefit of Courcelle's Theorem is that it provides "a simple way to recognize a property as being linear time computable".

## 6. Conclusions and future work

In this paper, we have used Courcelle's Theorem to prove the fixed-parameter tractability of a whole range of problems in knowledge representation and reasoning. More precisely, we have shown that many problems in the area of abduction, closed world reasoning, circumscription, and disjunctive logic programming are solvable in linear time if the treewidth of the considered formulae or programs is bounded by some constant. We have also experimented with a prototype implementation based on the standard MSO-to-FTA approach [25]. However, in accordance with [32], we have come to the conclusion that this generic way of turning the MSO characterization of a problem into an algorithm is not feasible in practice – despite the theoretical fixed-parameter linearity of this approach.

Thus, in the second part of this work, we have investigated an alternative method for turning theoretical tractability results obtained via Courcelle's Theorem into feasible computations. Based on the monadic datalog approach presented in [30], we have constructed new algorithms for logic-based abduction. The experimental results obtained with an implementation of these algorithms underline the feasibility of this approach.

The datalog programs presented in this paper were obtained by an ad hoc construction rather than via a generic transformation from MSO. Nevertheless, we are convinced that the idea of a bottom–up propagation of certain SAT and UNSAT conditions is quite generally applicable. As a next step, we are therefore planning to devise new algorithms based on our monadic datalog approach also for other problems in the area of knowledge representation and reasoning. In fact, for circumscription and for disjunctive logic programming, related algorithms have already been presented recently, see [36,37].

In this paper, the monadic datalog approach from [30] has only been applied to decision problems (e.g., is a given PAP solvable?) or to the enumeration problem of a unary query with one free individual variable (e.g., for a given PAP, compute all relevant hypotheses). In the future, we are planning to extend this approach to other kinds of problems like counting problems (e.g., how many solutions does a given PAP have?) or enumeration problems for more general queries (e.g., for a given PAP, compute all solutions). In fact, first steps in this direction have already been made in [36,37].

Our notion of treewidth of the problems considered here corresponds to the treewidth of the *primal graph* of the underlying structures, i.e.: this graph has as vertices all domain elements occurring in the structure; moreover, two vertices are adjacent in the primal graph if the corresponding domain elements jointly occur in some tuple in the structure. Clearly, also other mappings of the knowledge representation and reasoning problems to graphs – in particular, to directed graphs – are conceivable. Moreover, recently, other interesting notions of decompositions and width have been developed, see e.g. [7,6,43,35]. We are planning to investigate their applicability to the problems studied here.

## Acknowledgements

We are very grateful to the anonymous referees as well as to Michael Jakl, Stefan Rümmele, and Stefan Woltran for their valuable comments on previous versions of this article.

## Appendix A. Proof of Lemma 4.1

We have to show that the *solve*-predicate computed by the SAT-program has the intended meaning: For all values of $v, P, N, C$, the ground fact $solve(v, P, N, C)$ is true in the minimal model of the interpreted program if and only if the following condition holds:

**Property A.** *There exists an extension $J$ of the assignment $(P, N)$ to $Var(\mathcal{T}_v)$, s.t. $(Cl(\mathcal{T}_v) \setminus Cl(v)) \cup C$ is true in $J$ while for all clauses $c \in Cl(v) \setminus C$, the restriction $c|_{Var(\mathcal{T}_v)}$ is false in $J$.*

The proof goes by structural induction on the tree decomposition.

*Base case.* For a leaf node $v$, exactly those facts $solve(v, P, N, C)$ with $C \subseteq Cl(v)$ are true in the minimal model, for which the following condition holds: $\forall c \in C$, $c|_{Var(v)}$ evaluates to true in the assignment $(P, N)$ and $\forall c \in Cl(v) \setminus C$, $c|_{Var(v)}$ evaluates to false in $(P, N)$.

The only extension $J$ of $(P, N)$ to $Var(\mathcal{T}_v)$ is $(P, N)$ itself. Hence, $solve(v, P, N, C)$ is true in the minimal model if and only if Property A holds.

*Induction step – "only if"-direction.* Suppose that for arbitrary values of $v, P, N, C$, the ground fact $solve(v, P, N, C)$ is true in the minimal model of the interpreted program. In order to show that Property A holds, we distinguish five cases according to the five types of internal nodes.

(1) Variable removal node with removal of variable $x$. Let $solve(v, P, N, C)$ be true in the minimal model and let $v_1$ denote the child of $v$. Then either $solve(v_1, P \uplus \{x\}, N, C)$ or $solve(v_1, P, N \uplus \{x\}, C)$ is also true in the minimal model. We restrict ourselves to the first case here. The latter is treated analogously. By the induction hypothesis, there exists an extension $J$ of the assignment $(P \uplus \{x\}, N)$ to $Var(\mathcal{T}_{v_1}) = Var(\mathcal{T}_v)$, s.t. $(Cl(\mathcal{T}_v) \setminus Cl(v)) \cup C$ is true in $J$ while for all clauses $c \in C$, the restriction $c|_{Var(\mathcal{T}_v)}$ is false in $J$. Then $J$ is also the desired extension of $(P, N)$.

(2) Clause removal node with removal of clause $c$. Let $solve(v, P, N, C)$ be true in the minimal model and let $v_1$ denote the child of $v$. By construction, $C$ is of the form $C' \setminus \{c\}$ for some $C' \subseteq Cl(v_1)$, s.t. $solve(v_1, P, N, C')$ is true in the minimal model. By the induction hypothesis, there exists an extension $J$ of $(P, N)$ to $Var(\mathcal{T}_v) = Var(\mathcal{T}_{v_1})$, s.t. $(Cl(\mathcal{T}_{v_1}) \setminus Cl(v_1)) \cup C'$ is true in $J$ while for all clauses $d \in Cl(v_1) \setminus C'$, the restriction $d|_{Var(\mathcal{T}_v)}$ is false in $J$. Note that $Cl(v) \cup \{c\} = Cl(v_1)$ and $C \cup \{c\} = C'$. Hence, $J$ is also the desired extension of $(P, N)$ in the node $v$, i.e., $(Cl(\mathcal{T}_v) \setminus Cl(v)) \cup C = (Cl(\mathcal{T}_{v_1}) \setminus Cl(v_1)) \cup C'$ is true in $J$. Moreover, for every $d$ in $Cl(v) \setminus C = Cl(v_1) \setminus C'$, the restriction $d|_{Var(\mathcal{T}_v)}$ is false in $J$.

(3) Variable introduction node with introduction of variable $x$. Let $solve(v, P \uplus \{x\}, N, C)$ be true in the minimal model and let $v_1$ denote the child of $v$. The case $solve(v, P, N \uplus \{x\}, C)$ is treated analogously and, therefore, omitted here. There exist sets $C_1$ and $C_2$ with $C = C_1 \cup C_2$, s.t. $solve(v_1, P, N, C_1)$ and $true(\{x\}, \emptyset, C_2, Cl(v))$ are true in the minimal model. The latter condition means that $C_2$ contains exactly those clauses in $Cl(v)$ where the propositional variable $x$ occurs in unnegated form. Of course, we have $Cl(\mathcal{T}_{v_1}) = Cl(\mathcal{T}_v)$ and $Var(\mathcal{T}_{v_1}) = Var(\mathcal{T}_v) \setminus \{x\}$. By the induction hypothesis, there exists an extension $J_1$ of $(P, N)$ to $Var(\mathcal{T}_{v_1})$, s.t. $(Cl(\mathcal{T}_{v_1}) \setminus Cl(v_1)) \cup C_1$ is true in $J_1$ while for all clauses $c \in Cl(v_1) \setminus C_1$, the restriction $c|_{Var(\mathcal{T}_{v_1})}$ is false in $J_1$. Now consider the extension $J$ of $(P \uplus \{x\}, N)$ to $Var(\mathcal{T}_v)$ which is obtained by extending $J_1$ to $Var(\mathcal{T}_v) = Var(\mathcal{T}_{v_1}) \cup \{x\}$ with $J(x) = \text{true}$. By construction, all clauses in $C \setminus C_1$ have a positive occurrence of $x$ and are therefore true in $J$. Hence, all clauses in $(Cl(\mathcal{T}_v) \setminus Cl(v)) \cup C$ are indeed true in $J$. On the other hand, by the definition of $C$, for all clauses $c \in Cl(v) \setminus C$, the restriction $c|_{Var(\mathcal{T}_v)}$ is false in $J$. Hence, $J$ is indeed the desired extension of the assignment $(P \uplus \{x\}, N)$.

(4) Clause introduction node with introduction of clause $c$. We have $Var(\mathcal{T}_{v_1}) = Var(\mathcal{T}_v)$ and $Cl(\mathcal{T}_{v_1}) = Cl(\mathcal{T}_v) \setminus \{c\}$. By the connectedness condition, $c|_{Var(\mathcal{T}_v)} = c|_{Var(v)}$. Hence, the truth value of $c|_{Var(\mathcal{T}_v)}$ in any extension of $(P, N)$ to $Var(\mathcal{T}_v)$ coincides with the truth value of $c|_{Var(v)}$ in $(P, N)$. Now let $solve(v, P, N, C)$ be true in the minimal model and let $v_1$ denote the child of $v$. By construction, there exist facts $solve(v_1, P, N, C_1)$ and $true(P, N, C_2, \{c\})$ with $C = C_1 \cup C_2$ which are true in the minimal model. Thus, by the induction hypothesis, there exists an extension $J$ of $(P, N)$ to $Var(\mathcal{T}_{v_1}) = Var(\mathcal{T}_v)$, s.t. $(Cl(\mathcal{T}_{v_1}) \setminus Cl(v_1)) \cup C_1$ is true in $J$ while for all clauses $d \in Cl(v_1) \setminus C_1$, the restriction $d|_{Var(\mathcal{T}_v)}$ is false in $J$. We claim that $J$ is the desired extension of $(P, N)$ in $v$.

To see this, first consider the case that $c|_{Var(v)}$ is false in $(P, N)$. Hence, $C_2 = \emptyset$ and, therefore, $C = C_1$. Moreover, $(Cl(\mathcal{T}_v) \setminus Cl(v)) \cup C = (Cl(\mathcal{T}_{v_1}) \setminus Cl(v_1)) \cup C_1$ is true in $J$. On the other hand, for all clauses $d$ in $(Cl(v) \setminus C) = (Cl(v) \setminus C_1)$, we either have $d \in Cl(v_1) \setminus C_1$ or $d = c$. In either case, the restriction $d|_{Var(\mathcal{T}_v)}$ is false in $J$.

It remains to consider the case that $c|_{Var(v)}$ is true in $(P, N)$. In this case, $C_2 = \{c\}$ and $C$ is of the form $C = C_1 \cup \{c\}$. We have $(Cl(\mathcal{T}_v) \setminus Cl(v)) \cup C = (Cl(\mathcal{T}_{v_1}) \setminus Cl(v_1)) \cup C_1 \cup \{c\}$. We are considering the case that $c$ is true in $(P, N)$ and thus in $J$. Hence, $(Cl(\mathcal{T}_v) \setminus Cl(v)) \cup C$ is true in $J$. On the other hand, for all clauses $d$ in $(Cl(v) \setminus C) = (Cl(v_1) \setminus C_1)$, the restriction $d|_{Var(\mathcal{T}_v)}$ is false in $J$.

(5) Branch node. Suppose that $solve(v, P, N, C)$ is true in the minimal model with $C = C_1 \cup C_2$, s.t. $solve(v_1, P, N, C_1)$ and $solve(v_2, P, N, C_2)$ are also true in the minimal model. By the induction hypothesis, for each $i \in \{1, 2\}$, the assignment $(P, N)$ can be extended to some interpretation $J_i$ on $Var(\mathcal{T}_{v_i})$, s.t. $(Cl(\mathcal{T}_{v_i}) \setminus Cl(v)) \cup C_i$ is true in $J_i$ while for all clauses $c \in Cl(v) \setminus C_i$, the restriction $c|_{Var(\mathcal{T}_{v_i})}$ is false in $J_i$. We are using the fact that $Cl(v) = Cl(v_i)$ holds. Note that, by the definition of normalized tree decompositions and by the connectedness condition, $Var(\mathcal{T}_{v_1}) \cap Var(\mathcal{T}_{v_2}) = Var(v)$. Moreover,

$J_1$ and $J_2$ coincide on $Var(v)$ since they both extend $(P, N)$. Hence, we may define an extension $J$ of $(P, N)$ to $Var(\mathcal{T}_v)$ as follows

$$J(x) = \begin{cases} (P,N)(x) & \text{if } x \in Var(v), \\ J_1(x) & \text{if } x \in Var(\mathcal{T}_{v_1}) \setminus Var(v), \\ J_2(x) & \text{if } x \in Var(\mathcal{T}_{v_2}) \setminus Var(v). \end{cases}$$

We claim that $J$ has the desired property: Every clause $c$ in $(Cl(\mathcal{T}_v) \setminus Cl(v)) \cup C$ is contained in at least one of the sets $(Cl(\mathcal{T}_{v_i}) \setminus Cl(v)) \cup C_i$ with $i \in \{1, 2\}$. Hence, each such clause $c$ is true in $J$ by the induction hypothesis. Likewise, for every clause $c \in Cl(v) \setminus C$, both restrictions $c|_{Var(\mathcal{T}_{v_i})}$ are false in $J$. Hence, also $c|_{Var(\mathcal{T}_v)}$ is false in $J$.

*Induction step – "if"-direction.* Suppose that for arbitrary values of $v, P, N, C$, Property A holds, i.e., $v$ denotes a node in $\mathcal{T}$ and $(P, N)$ is an assignment on the variables $Var(v)$. Moreover, $C$ is an arbitrary clause set with $C \subseteq Cl(v)$ and there exists an extension $J$ of $(P, N)$ to $Var(\mathcal{T}_v)$, s.t. $(Cl(\mathcal{T}_v) \setminus Cl(v)) \cup C$ is true in $J$ while for all clauses $d \in Cl(v) \setminus C$, the restriction $d|_{Var(\mathcal{T}_v)}$ is false in $J$. We have to show that then the fact $solve(v, P, N, C)$ is indeed true in the minimal model of the interpreted program. Again, we distinguish the five cases according to the five types of internal nodes.

(1) Variable removal node with removal of variable $x$. Note that $Cl(\mathcal{T}_v) = Cl(\mathcal{T}_{v_1})$, $Cl(v) = Cl(v_1)$, $Var(\mathcal{T}_v) = Var(\mathcal{T}_{v_1})$, and $Var(v) = Var(v_1) \setminus \{x\}$, where $v_1$ is the child of $v$. We have to distinguish two cases depending on whether $J(x) = \text{true}$ or $J(x) = \text{false}$. We only treat the first case here. The second one goes analogously. Since $J(x) = \text{true}$, $J$ is also an extension of $(P \uplus \{x\}, N)$. By assumption, $(Cl(\mathcal{T}_v) \setminus Cl(v)) \cup C = (Cl(\mathcal{T}_{v_1}) \setminus Cl(v_1)) \cup C$ is true in $J$ while for all clauses $d \in Cl(v) \setminus C = Cl(v_1) \setminus C$, the restriction $d|_{Var(\mathcal{T}_v)} = d|_{Var(\mathcal{T}_{v_1})}$ is false in $J$. Hence, by the induction hypothesis, $solve(v_1, P \uplus \{x\}, N, C)$ is true in the minimal model and, therefore, $solve(v, P, N, C)$ is also true.

(2) Clause removal node with removal of clause $c$. Note that $Cl(\mathcal{T}_v) = Cl(\mathcal{T}_{v_1})$, $Cl(v) = Cl(v_1) \setminus \{c\}$, $Var(\mathcal{T}_v) = Var(\mathcal{T}_{v_1})$, and $Var(v) = Var(v_1)$, where $v_1$ is the child of $v$. Let $C' = C \cup \{c\}$. By assumption, all clauses in $(Cl(\mathcal{T}_v) \setminus Cl(v)) \cup C = (Cl(\mathcal{T}_{v_1}) \setminus Cl(v_1)) \cup C'$ are true in $J$. On the other hand, for all clauses $d$ in $Cl(v) \setminus C = Cl(v_1) \setminus C'$, the restriction $d|_{Var(\mathcal{T}_v)} = d|_{Var(\mathcal{T}_{v_1})}$ is false in $J$. Hence, by the induction hypothesis, $solve(v_1, P, N, C')$ is true in the minimal model and, therefore, $solve(v, P, N, C)$ is also true in the minimal model.

(3) Variable introduction node with introduction of variable $x$. In this case, we have $Cl(\mathcal{T}_{v_1}) = Cl(\mathcal{T}_v)$, $Cl(v_1) = Cl(v)$, $Var(\mathcal{T}_{v_1}) = Var(\mathcal{T}_v) \setminus \{x\}$, and $Var(v_1) = Var(v) \setminus \{x\}$, where $v_1$ is the child of $v$. We distinguish two cases depending on whether $x \in P$ or $x \in N$. We only treat the first case here. The second one goes analogously. Consider the assignment $(P \setminus \{x\}, N)$ on $Var(v_1) = Var(v) \setminus \{x\}$. Moreover, let $J_1$ denote the restriction of $J$ to $Var(\mathcal{T}_{v_1}) = Var(\mathcal{T}_v) \setminus \{x\}$. We define $C_1 = \{c \in C\colon\ c|_{Var(\mathcal{T}_{v_1})} \text{ is true in } J_1\}$. By the connectedness condition, the clauses in $(Cl(\mathcal{T}_{v_1}) \setminus Cl(v_1)) = (Cl(\mathcal{T}_v) \setminus Cl(v))$ do not contain the variable $x$. Hence, they have the same truth value in $J_1$ as in $J$, namely true. Thus, $(Cl(\mathcal{T}_{v_1}) \setminus Cl(v_1)) \cup C_1$ is true in $J_1$. On the other hand, for every $d \in Cl(v_1) \setminus C_1$, the restriction $d|_{Var(\mathcal{T}_{v_1})}$ is false in $J_1$. Thus, by the induction hypothesis, $solve(v_1, P \setminus \{x\}, N, C_1)$ is true in the minimal model.

Now let $C_2$ be defined as $C_2 = \{c \in C\colon\ x \text{ occurs unnegated in } c\}$. Then $C = C_1 \cup C_2$ holds. This is due to the fact that all clauses in $C$ are true in $J$. Hence, they are already true in $J_1$ or they become true in $J$ because of an unnegated occurrence of $x$. Moreover, by the definition of the *true*-predicate, the fact $true(\{x\}, \emptyset, C_2, C)$ is true in the minimal model. Thus, also $solve(v, P, N, C)$ is true in the minimal model.

(4) Clause introduction node with introduction of clause $c$. In this case, we have $Cl(\mathcal{T}_{v_1}) = Cl(\mathcal{T}_v) \setminus \{c\}$, $Cl(v_1) = Cl(v) \setminus \{c\}$, $Var(\mathcal{T}_{v_1}) = Var(\mathcal{T}_v)$ and $Var(v_1) = Var(v)$, where $v_1$ is the child of $v$. By the connectedness condition, $c|_{Var(\mathcal{T}_v)} = c|_{Var(v)}$ holds. Let $C_2$ be defined as $C_2 = \{c\}$ if $c|_{Var(v)}$ is true in $(P, N)$ and $C_2 = \emptyset$ otherwise. In either case, the fact $true(P, N, C_2, \{c\})$ is true in the minimal model.

Now let $C_1 = C \setminus C_2$. Clearly, $(Cl(\mathcal{T}_{v_1}) \setminus Cl(v_1)) \cup C_1 \subseteq (Cl(\mathcal{T}_v) \setminus Cl(v)) \cup C$ holds. By assumption, $(Cl(\mathcal{T}_v) \setminus Cl(v)) \cup C$ is true in $J$. Hence, $(Cl(\mathcal{T}_{v_1}) \setminus Cl(v_1)) \cup C_1$ is also true in $J$. Moreover, for all clauses $d$ in $Cl(v_1) \setminus C_1 \subseteq Cl(v) \setminus C$, the restriction $d|_{Var(\mathcal{T}_{v_1})} = d|_{Var(\mathcal{T}_v)}$ is false in $J$. Hence, by the induction hypothesis, $solve(v_1, P, N, C_1)$ is true in the minimal model. Thus, by construction, also $solve(v, P, N, C)$ is true in the minimal model.

(5) Branch node. Let $v_1, v_2$ denote the children of $v$ and let $i \in \{1, 2\}$. By the definition of branch nodes, we have $Cl(v) = Cl(v_1) = Cl(v_2)$ and $Var(v) = Var(v_1) = Var(v_2)$. Let $C_i$ be defined as $C_i = \{c \in C\colon\ c|_{Var(\mathcal{T}_{v_i})} \text{ is true in } J\}$. By assumption, for every $c \in C$, the restriction $c|_{Var(\mathcal{T}_v)}$ is true in $J$. Hence, for every $c \in C$, at least one of $c|_{Var(\mathcal{T}_{v_1})}$ and $c|_{Var(\mathcal{T}_{v_2})}$ is true in $J$. Thus, $C = C_1 \cup C_2$.

Let $J_i$ denote the restriction of $J$ to $Var(\mathcal{T}_{v_i})$. By the connectedness condition, for any clause $d \in (Cl(\mathcal{T}_{v_i}) \setminus Cl(v_i))$, the truth value of $d|_{Var(\mathcal{T}_{v_i})}$ in $J_i$ is identical to the truth value of $d$ in $J$, namely true. Hence, $(Cl(\mathcal{T}_{v_i}) \setminus Cl(v_i)) \cup C_i$ is true in $J_i$.

Now consider an arbitrary clause $d \in Cl(v_i) \setminus C_i = Cl(v) \setminus C_i \supseteq Cl(v) \setminus C$. If $d$ is in $Cl(v) \setminus C$ then $d|_{Var(\mathcal{T}_v)}$ is false in $J$ and, therefore, also $d|_{Var(\mathcal{T}_{v_i})}$ is false in $J_i$. On the other hand, if $d$ is in $C$ but not in $C_i$ then, by the definition of $C_i$, $d|_{Var(\mathcal{T}_{v_i})}$ is also false in $J_i$. Hence, in any case, the restriction $d|_{Var(\mathcal{T}_{v_i})}$ is false in $J_i$. Thus, by the induction hypothesis, $solve(v_i, P, N, C_i)$ is true in the minimal model for both $i \in \{1, 2\}$. Thus, $solve(v, P, N, C)$ with $C = C_1 \cup C_2$ is also true in the minimal model.

**Auxiliary Predicates**

$check(P, N, C_1, C, \text{'y'}) \leftarrow true(P, N, C_1, C),\ x \in N,\ man(x).$

$check(P, N, C_1, C, \text{'n'}) \leftarrow true(P, N, C_1, C),\ \text{not } check(P, N, C_1, C, \text{'y'}).$

$defined(v, S, i_1, i_2) \leftarrow aux(v, S, i_1, i_2, P, N, C, d),$

$equal(v, S, i_1, i_2, j_1, j_2) \leftarrow defined(v, S, i_1, i_2),\ defined(v, S, j_1, j_2),$
$\text{not } diff(v, S, i_1, i_2, j_1, j_2).$

$diff(v, S, i_1, i_2, j_1, j_2) \leftarrow defined(v, S, i_1, i_2),\ defined(v, S, j_1, j_2),$
$aux(v, S, i_1, i_2, P, N, C, d),\ \text{not } aux(v, S, j_1, j_2, P, N, C, d).$

$diff(v, S, i_1, i_2, j_1, j_2) \leftarrow defined(v, S, i_1, i_2),\ defined(v, S, j_1, j_2),$
$aux(v, S, j_1, j_2, P, N, C, d),\ \text{not } aux(v, S, i_1, i_2, P, N, C, d).$

$less(j_1, j_2, i_1, i_2) \leftarrow j_1 < i_1.$
$less(j, j_2, j, i_2) \leftarrow j_2 < i_2.$

$reduced\_smaller(v, S, i, i_1, i_2) \leftarrow less(j_1, j_2, i_1, i_2),\ reduce(v, S, i, j_1, j_2).$

$duplicate(v, S, i_1, i_2) \leftarrow less(j_1, j_2, i_1, i_2),\ equal(v, S, i_1, i_2, j_1, j_2).$

$open(v, S, i, i_1, i_2) \leftarrow j < i,\ \text{not } reduced\_smaller(v, S, j, i_1, i_2).$

$reduce(v, S, i, i_1, i_2) \leftarrow 0 \leqslant i < K,\ defined(v, S, i_1, i_2),$
$\text{not } reduced\_smaller(v, S, i, i_1, i_2),\ \text{not } duplicate(v, S, i_1, i_2),$
$\text{not } open(v, S, i, i_1, i_2).$

**Fig. B.1.** *reduce* and auxiliary predicates.

## Appendix B. Datalog definition of the *reduce*-predicate

As was mentioned in Section 4.2, we find it preferable to think of the *reduce*-predicate as a built-in predicate which can be implemented very efficiently via appropriate hash codes. However, this is not a necessity, since it could also be implemented in datalog (with negation). A datalog implementation of the *reduce*-predicate and of several auxiliary predicates is given in Fig. B.1.

The idea of $reduce(v, S, \ldots)$ is twofold: (1) We want to detect "duplicates": Two pairs $(i_1, i_2)$ and $(j_1, j_2)$ are duplicates if and only if for all values of $(P, N, C, d)$, a fact $aux(v, S, i_1, i_2, P, N, C, d)$ is true in the minimal model $\Leftrightarrow$ $aux(v, S, j_1, j_2, P, N, C, d)$ is true in the minimal model. In such a situation, we keep the lexicographically smallest pair $(i_1, i_2)$ and delete all other pairs. (2) We map the pairs of indices $(i_1, i_2)$ which are not deleted in (1) in lexicographic order to contiguous values $i$. The functionality of *reduce* can be provided very efficiently via hash codes for the facts $aux(v, S, i_1, i_2, \ldots)$. In Fig. B.1, we show that it is also definable in datalog.

The only rule with head predicate *reduce* guarantees that a pair $(i_1, i_2)$ of indices is mapped onto a single index $i$ if and only if the following conditions are fulfilled: (i) the pair $(i_1, i_2)$ is defined, i.e., there exists at least one *aux*-fact for this combination $(v, S, i_1, i_2)$, (ii) no lexicographically smaller pair $(j_1, j_2)$ is mapped to $i$, (iii) there is no lexicographically smaller pair $(j_1, j_2)$, s.t. $(i_1, i_2)$ and $(j_1, j_2)$ give rise to exactly the same set of ground atoms $aux(v, S, \_, \_, P, N, C, d)$ which are true in the minimal model of the interpreted program, and (iv) for all indices $j$ with $j < i$ there exists a pair $(j_1, j_2)$ smaller than $(i_1, i_2)$ that is mapped onto $j$.

The intended meaning of the predicates *check* and *reduce* has already been explained in Section 4.2. The other predicates in Fig. B.1 have the following meaning. $defined(v, S, i_1, i_2)$ means that there exists at least one *aux*-fact for this combination $(v, S, i_1, i_2)$. $equal(v, S, i_1, i_2, j_1, j_2)$ is true in the minimal model if and only if $(i_1, i_2)$ and $(j_1, j_2)$ give rise to exactly the same set of ground atoms $aux(v, S, \_, \_, P, N, C, d)$ which are true in the minimal model of the program, while $diff(v, S, i_1, i_2, j_1, j_2)$ means that $(i_1, i_2)$ and $(j_1, j_2)$ give rise to different sets of ground atoms $aux(v, S, \_, \_, P, N, C, d)$. $less(j_1, j_2, i_1, i_2)$ is true in the minimal model if $(j_1, j_2)$ is lexicographically smaller than $(i_1, i_2)$. With $reduced\_smaller(v, S, i, i_1, i_2)$ we can check if some index pair $(j_1, j_2)$, which is lexicographically smaller than $(i_1, i_2)$, is mapped onto $i$. $duplicate(v, S, i_1, i_2)$ means that there exists a lexicographically smaller pair $(j_1, j_2)$, s.t. $(i_1, i_2)$ and $(j_1, j_2)$ give rise to exactly the same set of ground atoms $aux(v, S, \_, \_, P, N, C, d)$ true in the minimal model; in other words, $(i_1, i_2)$ is a duplicate. Finally, $open(v, S, i, i_1, i_2)$ means that there exists an index $j < i$ which has not been used as the target of the mapping defined by the *reduce*-predicate, i.e., there exists no index pair $(j_1, j_2)$ smaller than $(i_1, i_2)$, s.t. $(j_1, j_2)$ is mapped onto $j$.

## References

[1] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison–Wesley, 1995.
[2] S. Arnborg, Efficient algorithms for combinatorial problems with bounded decomposability—A survey, BIT 25 (1) (1985) 2–23.
[3] S. Arnborg, J. Lagergren, D. Seese, Easy problems for tree-decomposable graphs, J. Algorithms 12 (2) (1991) 308–340.
[4] A. Atserias, J.K. Fichte, M. Thurley, Clause-learning algorithms with many restarts and bounded-width resolution, in: Proc. SAT'09, in: Lecture Notes in Computer Science, vol. 5584, Springer, 2009, pp. 114–127.

[5] M. Baaz, U. Egly, A. Leitsch, Normal form transformations, in: J.A. Robinson, A. Voronkov (Eds.), Handbook of Automated Reasoning, vol. 1, Elsevier Science, 2001, pp. 273–333, Chapter 5.
[6] D. Berwanger, A. Dawar, P. Hunter, S. Kreutzer, DAG-width and parity games, in: Proc. STACS'06, in: Lecture Notes in Computer Science, vol. 3884, 2006, pp. 524–536.
[7] D. Berwanger, E. Grädel, Entanglement—A measure for the complexity of directed graphs with applications to logic and games, in: Proc. LPAR'04, in: Lecture Notes in Computer Science, vol. 3452, 2005, pp. 209–223.
[8] H.L. Bodlaender, A linear-time algorithm for finding tree-decompositions of small treewidth, SIAM J. Comput. 25 (6) (1996) 1305–1317.
[9] H.L. Bodlaender, A.M.C.A. Koster, Safe separators for treewidth, Discrete Math. 306 (3) (2006) 337–350.
[10] H.L. Bodlaender, A.M.C.A. Koster, Combinatorial optimization on graphs of bounded treewidth, Comput. J. 51 (3) (2008) 255–269.
[11] R.B. Borie, Generation of polynomial-time algorithms for some optimization problems on tree-decomposable graphs, Algorithmica 14 (2) (1995) 123–137.
[12] R.B. Borie, R.G. Parker, C.A. Tovey, Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families, Algorithmica 7 (5–6) (1992) 555–581.
[13] M. Cadoli, M. Lenzerini, The complexity of propositional closed world reasoning and circumscription, J. Comput. Syst. Sci. 48 (2) (1994) 255–310.
[14] B. Courcelle, Graph rewriting: An algebraic and logic approach, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, vol. B, Elsevier Science Publishers, 1990, pp. 193–242.
[15] B. Courcelle, J.A. Makowsky, U. Rotics, On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic, Discrete Appl. Math. 108 (1-2) (2001) 23–52.
[16] B. Courcelle, M. Mosbah, Monadic second-order evaluations on tree-decomposable graphs, Theor. Comput. Sci. 109 (1–2) (1993) 49–82.
[17] J. Doner, Tree acceptors and some of their applications, J. Comput. Syst. Sci. 4 (5) (1970) 406–451.
[18] W.F. Dowling, J.H. Gallier, Linear-time algorithms for testing the satisfiability of propositional Horn formulae, J. Log. Program. 1 (3) (1984) 267–284.
[19] R.G. Downey, M.R. Fellows, Parameterized Complexity, Springer, New York, 1999.
[20] T. Eiter, G. Gottlob, Propositional circumscription and extended closed world reasoning are $\Pi_2^p$-complete, Theor. Comput. Sci. 114 (1993) 231–245.
[21] T. Eiter, G. Gottlob, The complexity of logic-based abduction, J. ACM 42 (1) (1995) 3–42.
[22] T. Eiter, G. Gottlob, On the computational cost of disjunctive logic programming: Propositional case, Ann. Math. Artif. Intell. 15 (3/4) (1995) 289–323.
[23] G. Filé, Tree automata and logic programs, in: Proc. STACS'85, in: Lecture Notes in Computer Science, vol. 182, 1985, pp. 119–130.
[24] E. Fischer, J.A. Makowsky, E.V. Ravve, Counting truth assignments of formulas of bounded tree-width or clique-width, Discrete Appl. Math. 156 (4) (2008) 511–529.
[25] J. Flum, M. Frick, M. Grohe, Query evaluation via tree-decompositions, J. ACM 49 (6) (2002) 716–752.
[26] M. Frick, M. Grohe, The complexity of first-order and monadic second-order logic revisited, Ann. Pure Appl. Logic 130 (1–3) (2004) 3–31.
[27] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: Proc. ICLP/SLP'88, MIT Press, 1988, pp. 1070–1080.
[28] M. Gelfond, H. Przymusinska, T.C. Przymusinski, On the relationship between circumscription and negation as failure, Artif. Intell. 38 (1) (1989) 75–94.
[29] G. Gottlob, R. Pichler, F. Wei, Bounded treewidth as a key to tractability of knowledge representation and reasoning, in: Proc. AAAI'06, AAAI Press, 2006, pp. 250–256.
[30] G. Gottlob, R. Pichler, F. Wei, Monadic datalog over finite structures with bounded treewidth, in: Proc. PODS'07, ACM, 2007, pp. 165–174.
[31] G. Gottlob, R. Pichler, F. Wei, Abduction with bounded treewidth: From theoretical tractability to practically efficient computation, in: Proc. AAAI'08, AAAI Press, 2008, pp. 1541–1546.
[32] M. Grohe, Descriptive and parameterized complexity, in: Proc. CSL'99, in: Lecture Notes in Computer Science, vol. 1683, 1999, pp. 14–31.
[33] J. Gustedt, O.A. Mæhle, J.A. Telle, The treewidth of Java programs, in: Proc. ALENEX'02: 4th International Workshop on Algorithm Engineering and Experiments, Revised Papers, in: Lecture Notes in Computer Science, vol. 2409, Springer, 2002, pp. 86–97.
[34] M. Hermann, R. Pichler, Counting complexity of minimal cardinality and minimal weight abduction, in: Proc. JELIA'08, in: Lecture Notes in Computer Science, vol. 5293, Springer, 2008, pp. 206–218.
[35] P. Hunter, S. Kreutzer, Digraph measures: Kelly decompositions, games, and ordering, Theor. Comput. Sci. 399 (2008) 206–219.
[36] M. Jakl, R. Pichler, S. Rümmele, S. Woltran, Fast counting with bounded treewidth, in: Proc. LPAR'08, in: Lecture Notes in Computer Science, vol. 5330, 2008, pp. 436–450.
[37] M. Jakl, R. Pichler, S. Woltran, Answer-set programming with bounded treewidth, in: Proc. IJCAI'09, 2009, pp. 816–822.
[38] N. Klarlund, A. Møller, M.I. Schwartzbach, MONA implementation secrets, Int. J. Found. Comput. Sci. 13 (4) (2002) 571–586.
[39] A.M.C.A. Koster, H.L. Bodlaender, S.P.M. van Hoesel, Treewidth: Computational experiments, Electronic Notes in Discrete Mathematics 8 (2001) 54–57.
[40] G. Marque-Pucheu, Rational set of trees and the algebraic semantics of logic programming, Acta Inf. 20 (1983) 249–260.
[41] H. Maryns, On the implementation of tree automata: Limitations of the naive approach, in: Proc. TLT'06: 5th Int. Treebanks and Linguistic Theories Conference, 2006, pp. 235–246.
[42] M. Minoux, LTUR: A simplified linear-time unit resolution algorithm for horn formulae and computer implementation, Inf. Process. Lett. 29 (1) (1988) 1–12.
[43] J. Obdržálek, DAG-width: Connectivity measure for directed graphs, in: Proc. SODA'06, ACM Press, 2006, pp. 814–821.
[44] C.H. Papadimitriou, Computational Complexity, Addison–Wesley, 1994.
[45] T.C. Przymusinski, Stable semantics for disjunctive programs, New Generation Comput. 9 (3/4) (1991) 401–424.
[46] J.W. Thatcher, J.B. Wright, Generalized finite automata theory with an application to a decision problem of second-order logic, Mathematical Systems Theory 2 (1) (1968) 57–81.
[47] W. Thomas, Languages, automata, and logic, in: Handbook of Formal Languages, vol. III, Springer, New York, 1997, pp. 389–455.
[48] M. Thorup, All structured programs have small tree-width and good register allocation., Inf. Comput. 142 (2) (1998) 159–181.
[49] F. van den Eijkhof, H.L. Bodlaender, A.M.C.A. Koster, Safe reduction rules for weighted treewidth, Algorithmica 47 (2) (2007) 139–158.