DBAI

TECHNICAL REPORT

INSTITUT FÜR INFORMATIONSSYSTEME

ABTEILUNG DATENBANKEN UND ARTIFICIAL INTELLIGENCE

Fast Counting with Bounded Treewidth

DBAI-TR-2008-61

Michael Jakl **Reinhard Pichler** **Stefan Rümmele**
Stefan Woltran

Institut für Informationssysteme
Abteilung Datenbanken und
Artificial Intelligence
Technische Universität Wien
Favoritenstr. 9
A-1040 Vienna, Austria
Tel: +43-1-58801-18403
Fax: +43-1-58801-18492
sekret@dbai.tuwien.ac.at
www.dbai.tuwien.ac.at

DBAI TECHNICAL REPORT
2008

TU
TECHNISCHE UNIVERSITÄT WIEN

DBAI TECHNICAL REPORT
DBAI TECHNICAL REPORT DBAI-TR-2008-61, 2008

Fast Counting with Bounded Treewidth

Michael Jakl **Reinhard Pichler** **Stefan Rümmele**
Stefan Woltran[1]

Abstract. Many intractable problems have been shown to become tractable if the treewidth of the underlying structure is bounded by a constant. An important tool for deriving such results is Courcelle's Theorem, which states that all properties defined by Monadic-Second Order (MSO) sentences are fixed-parameter tractable with respect to the treewidth. Arnborg et al. extended this result to counting problems defined via MSO properties. However, the MSO description of a problem is of course not an algorithm. Consequently, proving the fixed-parameter tractability of some problem via Courcelle's Theorem can be considered as the starting point rather than the endpoint of the search for an efficient algorithm. Gottlob et al. have recently presented a new approach via monadic datalog to actually devise efficient algorithms for decision problems whose tractability follows from Courcelle's Theorem. In this paper, we extend this approach and apply it to some fundamental counting problems in logic an artificial intelligence.

[1]Institute for Information Systems 184/2, Technische Universität Wien, Favoritenstrasse 9-11, 1040 Vienna, Austria. E-mail: {jakl | pichler | ruemmele | woltran }@dbai.tuwien.ac.at

Acknowledgements: The authors acknowledge support by the Austrian Science Fund (FWF) grant P20704-N18.

This is an extended version of a paper published in the Proceedings of the 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'08).

1 Introduction

Many problems which are, in general, intractable, have been shown to become tractable if the treewidth of the underlying structure is bounded by a constant. An important tool for deriving such results is Courcelle's Theorem [1]. It states that any property of finite structures, which is expressible by a Monadic Second Order (MSO) sentence, can be decided in linear time (data complexity) if the structures under consideration have bounded treewidth. Courcelle's Theorem has been successfully applied to derive tractability results in a great variety of fields. Recently, also its applicability to AI has been underlined by showing that many fundamental problems in the area of nonmonotonic reasoning and knowledge representation can be encoded as MSO sentences [2]. In [3], it was shown that the fixed-parameter tractability (FPT) via Courcelle's Theorem can be extended to counting problems defined via MSO properties.

Clearly, the MSO description of a problem is not an algorithm. Previous methods for constructing concrete algorithms from an MSO description [3, 4] first transform the MSO evaluation problem into a tree language recognition problem, which is then solved via a finite tree automaton (FTA). However, this approach has turned out to be only of theoretical value, since even very simple MSO formulae quickly lead to a "state explosion" of the FTA (see [5]). Consequently, it was already stated in [6] that the algorithms derived via Courcelle's Theorem are "useless for practical applications" and that the main benefit of Courcelle's Theorem is in providing "a simple way to recognize a property as being linear time computable". Of course, this also applies to the extension of Courcelle's Theorem to counting problems according to [3]. 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.

Recently, an alternative method to tackle this class of fixed-parameter tractable problems via *monadic datalog* has been proposed in [7]. In particular, it has been shown that if some property of finite structures is expressible in MSO then it can also be expressed by means of a monadic datalog program over the structure plus the tree decomposition. The monadic datalog approach has been applied to problems from different areas [7, 8] including propositional satisfiability (SAT) and abduction. In this paper, we show that the monadic datalog approach can be extended in such a way that it also provides concrete algorithms for some fundamental counting problems.

Results. We present new algorithms for the following problems: #SAT – the problem of counting *all* models of a propositional formula (without restriction, this is a classical #P-complete problem); #CIRCUMSCRIPTION – the problem of counting the (subset) *minimal* models of a propositional formula (this problem was, apart from the generic $\#\Pi_1$SAT-problem, one of the first problems to be shown #NP-complete [9]); and #HORN-ABDUCTION – the problem of counting the solutions of a propositional abduction problem where the underlying theory is given by a set of Horn clauses. The #P-completeness of this problem has been recently shown in [10]. Finally, we also report on experimental evaluations of the #SAT algorithm. In particular, we compare a dedicated implementation (where datalog serves as a "specification") with direct realizations of the datalog approach on top of the DLV-system [11]. Our experiments underline that our approach of counting indeed yields the expected fixed-parameter tractability and that – in great contrast to the MSO-to-FTA approach – there are no "hidden constants" in the runtime behavior to render these algorithms useless.

Related Work. As mentioned above, counting problems defined via MSO properties were shown in [3] to be FPT w.r.t. the treewidth of the input structures. In [12], this FPT result was extended to graphs with bounded clique-width. An algorithm for solving #SAT and #GENSAT in case of bounded treewidth or clique-width of the primal or incidence graph was presented in [13]. Moreover, it is sketched how this approach based on recursive splitting can be extended to other #P-complete problems. In [14], new #SAT-algorithms based on dynamic programming were presented for bounded treewidth of several graphs related to a propositional formula in CNF, namely the primal graph, dual graph, and incidence graph. Our notion of treewidth of a CNF-formula (see Section 2) corresponds to the treewidth of the incidence graph.

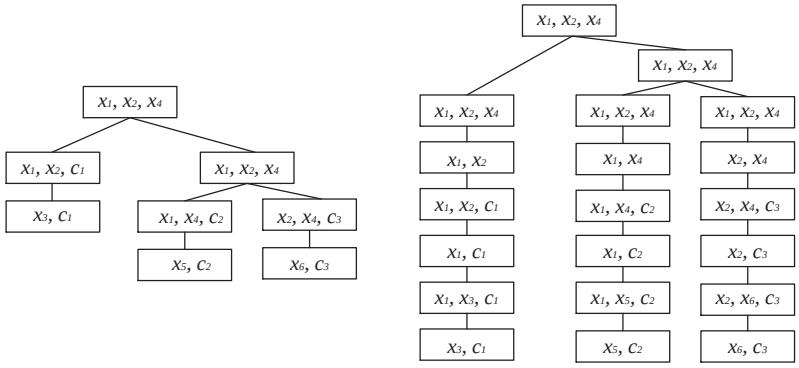
2 Preliminaries

Finite Structures and Treewidth. Let $\tau = \{R_1, \ldots, R_K\}$ be a set of predicate symbols. A *finite structure* $\mathcal{A}$ over τ (a *τ-structure*, for short) is given by a finite domain $A = dom(\mathcal{A})$ and relations $R_i^{\mathcal{A}} \subseteq A^\alpha$, where α is the arity of $R_i \in \tau$. A *tree decomposition* $\mathcal{T}$ of a τ-structure $\mathcal{A}$ is a pair $\langle T, (A_t)_{t \in T} \rangle$ where T is a tree and each A_t is a subset of A, s.t. the following properties hold: (1) Every $a \in A$ is contained in some A_t. (2) For every $R_i \in \tau$ and every tuple $(a_1, \ldots, a_\alpha) \in R_i^{\mathcal{A}}$, there exists a node $t \in T$ with $\{a_1, \ldots, a_\alpha\} \subseteq A_t$. (3) For every $a \in A$, $\{t \mid a \in A_t\}$ induces a subtree of T.

The sets A_t are called the *bags* 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 $tw(\mathcal{A})$. For given $w \geq 1$, it can be decided in linear time if some structure has treewidth $\leq w$. Moreover, in case of a positive answer, a tree decomposition of width w can be computed in linear time [15].

Example 1 ([2]). We can represent propositional formulae in CNF as finite structures over the alphabet $\tau = \{cl(.),\ var(.),\ pos(.,.),\ neg(.,.)\}$ where $cl(z)$ (resp. $var(z)$) means that z is a clause (resp. a variable) and $pos(x, c)$ (resp. $neg(x, c)$) means that x occurs unnegated (resp. negated) in the clause c. For instance, the formula $\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_4 \vee \neg x_5) \wedge (x_2 \vee \neg x_4 \vee x_6)$ corresponds to the structure $\mathcal{A}$ given by the set of ground atoms $\{var(x_1), var(x_2), var(x_3),\ var(x_4), var(x_5),\ var(x_6),\ cl(c_1),\ cl(c_2),\ cl(c_3),\ pos(x_1, c_1),\ pos(x_3, c_1),\ pos(x_4, c_2), pos(x_2, c_3),\ pos(x_6, c_3),\ neg(x_2, c_1),\ neg(x_1, c_2),\ neg(x_5, c_2),\ neg(x_4, c_3)\}$. Two tree decompositions $\mathcal{T}_1$ and $\mathcal{T}_2$ of $\mathcal{A}$ are given in Figure 1. Note that the maximal size of the bags is 3 in both decompositions. Hence, the treewidth is ≤ 2. On the other hand, it can be shown that these tree decompositions are optimal in the sense that we have $tw(\varphi) = tw(\mathcal{A}) = 2$. $\diamond$

In [7], it was shown that the following form of *normalized tree decompositions* can be obtained in linear time: (1) All bags contain either w or $w + 1$ pairwise distinct elements. W.l.o.g., we may assume that the domain contains at least w elements. (2) Every internal node $t \in T$ has either 1 or 2 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. In this case, we call t a *branch node*. In this paper, we only deal with finite structures representing propositional formulae in CNF (possibly Horn). Hence, the domain elements are either variables or clauses. Consequently, in case (3),



(a) tree decomposition $\mathcal{T}_1$ (b) tree decomposition $\mathcal{T}_2$

Fig. 1. Tree decompositions of formula φ of Example 1.

we call a node t in the tree decomposition a *variable removal node*, a *clause removal node*, a *variable introduction node*, or a *clause introduction node*, respectively.

The tree decomposition $\mathcal{T}_2$ in Figure 1 is normalized in this sense.

MSO and Monadic Datalog. MSO extends First Order logic (FO) by the use of *set variables* (denoted by upper case letters), which range over sets of domain elements. In contrast, the *individual variables* (denoted by lower case letters) range over single domain elements. An MSO formula $\varphi(x)$ with exactly one free individual variable is called a *unary query*. *Datalog* programs are function-free logic programs. The (minimal-model) semantics can be defined as the least fixpoint (lfp) of applying the immediate consequence operator. Predicates occurring only in the body of rules are called extensional. Predicates occurring also in the head of some rule are called intensional.

Let $\mathcal{A}$ be a τ-structure of treewidth $w \geq 1$. Then we define the extended signature $\tau_{td} = \tau \cup \{root, leaf, child_1, child_2, bag\}$, where the unary predicates $root$ and $leaf$ as well as the binary predicates $child_1$ and $child_2$ are used to represent the tree of a tree decomposition (of width w) in the obvious way. Finally, predicate bag has arity $k + 2$ with $k \leq w$, where $bag(t, a_0, \ldots, a_k)$ means that the bag at node t is $(a_0, \ldots, a_k)$.

In [7], the following connection between unary MSO queries over structures with bounded treewidth and monadic datalog was established:

Theorem 1. *Let τ and $w \geq 1$ be arbitrary but fixed. Every MSO-definable unary query over τ-structures of treewidth w is also definable by a monadic datalog program over τ_{td}. Moreover, the resulting program can be evaluated in linear time w.r.t. the size of the original τ-structure.*

3 Counting all models

We start our investigation of counting problems with the #SAT problem, i.e.: given a clause set $\mathcal{C}$ over variables V, count the number of all models $J \subseteq V$ of $\mathcal{C}$ (We identify an assignment with the set of atoms that are true in it). Suppose that an instance of #SAT is given as a τ_{td}-structure with $\tau_{td} = \{cl, var, pos, neg, root, leaf, child_1, child_2, bag\}$, encoding a clause set together with a tree decomposition $\mathcal{T}$ of width w (as explained Example 1). An extended datalog program for #SAT is displayed in Figure 2.

Program #SAT

/* leaf node. */
$sat(v, P, N, C_u, 1) \leftarrow leaf(v)$, $bag(v, X, C)$, $partition(X, P, N)$, $true(P, N, C_u, C)$.

/* variable removal node. */
$sat(v, P, N, C_u, j_1 + j_2) \leftarrow bag(v, X, C)$, $child_1(v_1, v)$, $bag(v_1, X \uplus \{x\}, C)$,
$sat(v_1, P \uplus \{x\}, N, C_u, j_1)$, $sat(v_1, P, N \uplus \{x\}, C_u, j_2)$.
$sat(v, P, N, C_u, j) \leftarrow bag(v, X, C)$, $child_1(v_1, v)$, $bag(v_1, X \uplus \{x\}, C)$,
$sat(v_1, P \uplus \{x\}, N, C_u, j)$, not $sat(v_1, P, N \uplus \{x\}, C_u, \_)$.
$sat(v, P, N, C_u, j) \leftarrow bag(v, X, C)$, $child_1(v_1, v)$, $bag(v_1, X \uplus \{x\}, C)$,
$sat(v_1, P, N \uplus \{x\}, C_u, j)$, not $sat(v_1, P \uplus \{x\}, N, C_u, \_)$.

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

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

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

/* branch node. */
$sat(v, P, N, C_u, \text{SUM}(j)) \leftarrow child_1(v_1, v)$, $bag(v_1, X, C)$, $sat(v_1, P, N, C_1, j_1)$,
$child_2(v_2, v)$, $bag(v_2, X, C)$, $sat(v_2, P, N, C_2, j_2)$,
$bag(v, X, C)$, $C_1 \cup C_2 = C_u$, $j_1 * j_2 = j$.

/* result (at the root node). */
$count(\text{SUM}(j)) \leftarrow root(v)$, $bag(v, X, C)$, $sat(v, P, N, C, j)$.

Fig. 2. #SAT program.

In this program, we adhere to the following notational conventions: Lower case letters v, c, x, and j (possibly with subscripts) are used as datalog variables for a single node in $\mathcal{T}$, for a single clause, for a single propositional variable, or for an integer number, respectively. 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). In particular, for the sake of readability, we present the extensional predicate bag in the form $bag(v, X, C)$, where X (resp. C) denotes the set of variables (resp. clauses) in the bag at node v in $\mathcal{T}$. Note that all these 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 ultimately feed these sets to the datalog system DLV in the form of individual arguments of appropriate variants of the predicates involved, see Section 6.

We are also using non-datalog expressions involving the $\cup$- and $\uplus$-operator for ordinary resp. disjoint union. They could be easily replaced by "proper" datalog expressions, e.g., $C_1 \cup C_2 = C_u$ can of course be replaced by union(C_1, C_2, C_u). Moreover, we need arithmetic expressions $j_1 + j_2$ and $j_1 * j_2$ as well as the SUM-operator for the counting. The SUM-operator occurs as the expression SUM(j) in the rule heads

only. Its semantics is like the SUM aggregate function in ordinary SQL, where we first apply a GROUP BY over all remaining head variables to the result of evaluating the conjunctive query in the body of the rule.

For the discussion of the #SAT program below, it is convenient to 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 clauses in the bag of v while $Cl(\mathcal{T}_v)$ denotes the 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 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 four intensional predicates sat, $true$, $partition$, and $count$. The crucial predicate is $sat(v, P, N, C, j)$ with the following intended meaning: v denotes a node in $\mathcal{T}$. P and N form a partition of $Var(v)$ representing a truth assignment on $Var(v)$, s.t. all variables in P are true and all variables in N are false. C denotes a subset of $Cl(v)$ and j denotes a positive integer. For arbitrary values of v, P, N, C, we define the following set of truth assignments:

$$\begin{aligned} S(v, P, N, C) = \{ J \mid\ & J \text{ is an extension of } (P, N) \text{ to } Var(\mathcal{T}_v), \\ & \text{for each } c \in (Cl(\mathcal{T}_v) \setminus Cl(v)) \cup C,\ c \text{ is true in } J, \\ & \text{for each } c \in Cl(v) \setminus C,\ c|_{Var(\mathcal{T}_v)} \text{ is false in } J. \} \end{aligned}$$

We can now characterize the least fixpoint (lfp) of the #SAT program as follows.

Property A. If $S(v, P, N, C) = \emptyset$ then no atom $sat(v, P, N, C, \_)$ is in the lfp of #SAT. If $S(v, P, N, C) \neq \emptyset$ then the following equivalence holds: $sat(v, P, N, C, j)$ is in the lfp of #SAT iff $|S(v, P, N, C)| = j$.

This property implies that, for any given values v, P, N, C, we can derive at most one fact $sat(v, P, N, C, j)$. The main task of the program is the computation of all facts $sat(v, P, N, C, j)$ by means of a bottom-up traversal of the tree decomposition $\mathcal{T}$. Indeed, all the rules only allow us to derive sat-facts for some node v in $\mathcal{T}$ from sat-facts at the child node(s) of v. Consequently, on the ground level, the program contains only stratified negation, since the not-operator in the rules of variable removal nodes is only applied to sat-facts of the child node v_1 of v.

The other predicates have the following meaning: $true(P, N, C_u, C)$ means that C_u 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. A fact $partition(X, P, N)$ expresses that (P, N) is a partition of X. The predicate $count$ holds the final result. The datalog program in Figure 2 solves the #SAT problem in the following way.

Theorem 2. *Let $\mathcal{C}$ be an instance of #SAT, encoded by a τ_{td}-structure $\mathcal{A}_{td}$. Then, $count(j)$ with $j \geq 1$ is in the lfp of the #SAT-program evaluated on $\mathcal{A}_{td}$ iff $\mathcal{C}$ is satisfiable and has exactly j models. Moreover, both the construction of the τ_{td}-structure $\mathcal{A}_{td}$ and the evaluation of the program take time $\mathcal{O}(f(tw(\mathcal{C})) * \|\mathcal{C}\|)$ for some function f, if we assume constant runtime for the arithmetic operations.*

Proof. Suppose that the predicate sat indeed fulfills Property A, which can be proved by structural induction on $\mathcal{T}$. The case distinction over all possible kinds of nodes is

rather straightforward – the only non-trivial case being the case of branch nodes. A detailed proof is given in Appendix A.

Now consider the root node v of the tree decomposition $\mathcal{T}$ with $bag(v, X, C)$. A fact $sat(v, P, N, C, j)$ in the lfp means that the assignment (P, N) on the variables X has exactly j extensions to all variables, s.t. all clauses in $\mathcal{C}$ are true. But then, by the semantics of the SUM-operator explained above, the rule with head $count(\text{SUM}(j))$ indeed means that a fact $count(j')$ with $j' \geq 1$ is in the lfp iff j' is the number of assignments that satisfy all clauses in $\mathcal{C}$, i.e., j' is the sum of j over all possible partitions (P, N) of X, s.t. $sat(v, P, N, C, j)$ is in the lfp. We are thus using that the root v of $\mathcal{T}$ is unique and the values of X and C in the bag at v are uniquely determined by v. Moreover, for every pair of (P, N) (together with C, which is fixed for v), the value of j is also uniquely determined.

The linear time data complexity is due to the fact that our #SAT program is essentially a succinct representation of a monadic datalog program extended by a counter j. For instance, in the atom $sat(v, P, N, C, j)$, the sets P, N, and C are subsets of bounded size of the bag of v. Hence, each combination P, N, C could be represented by sets $r, s, t \subseteq \{0, \ldots, w\}$ referring to indices of elements in the bag of v. Recall that w is a fixed constant. Hence, $sat(v, P, N, C, j)$ is simply a succinct representation of constantly many predicates of the form $sat_{r,s,t}(v, j)$. Hence, without the counter j, the linear time bound is implicit in Theorem 1. Moreover, j is uniquely determined for every combination of v, P, N, C, and the concrete value of j is computed by simple addition and multiplication of the corresponding values in sat-facts at the child node(s) of v. Hence, maintaining this additional argument j does not destroy the linearity. □

4 Counting the minimal models

We now extend the #SAT program in order to solve the #CIRCUMSCRIPTION problem, i.e.: given a propositional formula φ, count the number of *minimal* models of φ. The goal of the program in Figure 3 and 4 is, on the one hand, to keep track of *all* models of a formula φ given by the input τ_{td}-structure. This is done by the *sat*-predicate which works essentially as in the #SAT program. However, at the end of the day, we may only count the *minimal* models. Our #CIRCUMSCRIPTION program therefore also contains an *unsat*-predicate, which is used to propagate "unsat"-conditions in the sense that some model J is minimal only if all strictly smaller assignments $J' \subset J$ do not satisfy φ. Recall that we identify an assignment with the set of atoms that are true in it.

A complication which our program has to overcome is that we have to keep track which unsat-conditions refer to which sat-condition. Thus the *sat*-predicate has an index $i \in \{0, 1, 2, \ldots\}$ as additional argument. The first four arguments v, i, P, N allow us to associate each *unsat*-fact with the correct *sat*-fact. The *sat*- and *unsat*-predicates have the following meaning: Let v denote a node in the tree decomposition $\mathcal{T}$. Let the sets P and N (resp. P' and N') denote a partition of $Var(v)$ representing a truth assignment on $Var(v)$, s.t. all variables in P (resp. in P') are true and all variables in N (resp. in N') are false. Let C and C' denote subsets of $Cl(v)$. Furthermore let $i \in \{0, 1, 2, \ldots\}$ be an index used to distinguish different extensions of a truth assignment and let j be a positive integer used for counting extensions of a truth assignment. Moreover, let the set $S(v, P, N, C)$ of truth assignments be defined as in Section 3. Then, occurrences of the ground facts $sat(v, i, P, N, C, j)$ and $unsat(v, i, P, N, P', N', C')$ in the least fixpoint (lfp) of #CIRCUMSCRIPTION are determined as follows:

Program #CIRCUMSCRIPTION

/* leaf node. */
$sat(v, 0, P, N, C_u, 1) \leftarrow leaf(v)$, $bag(v, X, C)$, $partition(X, P, N)$, $true(P, N, C_u, C)$.
$unsat(v, 0, P, N, P', N', C'_u) \leftarrow leaf(v)$,
$sat(v, 0, P, N, \_, 1)$, $sat(v, 0, P', N', C'_u, 1)$, $P' \subset P$.

/* variable removal node. */
$auxsat(v, i, 0, P, N, C_u, j) \leftarrow bag(v, X, C)$, $child_1(v_1, v)$, $bag(v_1, X \uplus \{x\}, C)$,
$sat(v_1, i, P \uplus \{x\}, N, C_u, j)$.
$auxsat(v, i, 1, P, N, C_u, j) \leftarrow bag(v, X, C)$, $child_1(v_1, v)$, $bag(v_1, X \uplus \{x\}, C)$,
$sat(v_1, i, P, N \uplus \{x\}, C_u, j)$.
$auxunsat(v, i, 0, P, N, P' \setminus \{x\}, N' \setminus \{x\}, C'_u) \leftarrow bag(v, X, C)$, $child_1(v_1, v)$,
$bag(v_1, X \uplus \{x\}, C)$, $unsat(v_1, i, P \uplus \{x\}, N, P', N', C'_u)$.
$auxunsat(v, i, 1, P, N, P', N' \setminus \{x\}, C'_u) \leftarrow bag(v, X, C)$, $child_1(v_1, v)$,
$bag(v_1, X \uplus \{x\}, C)$, $unsat(v_1, i, P, N \uplus \{x\}, P', N', C'_u)$.

/* clause removal node. */
$sat(v, i, P, N, C_u, j) \leftarrow bag(v, X, C)$, $child_1(v_1, v)$, $bag(v_1, X, C \uplus \{c\})$,
$sat(v_1, i, P, N, C_u \uplus \{c\}, j)$.
$unsat(v, i, P, N, P', N', C'_u) \leftarrow bag(v, X, C)$, $child_1(v_1, v)$, $bag(v_1, X, C \uplus \{c\})$,
$sat(v_1, i, P, N, C_u \uplus \{c\}, \_)$, $unsat(v_1, i, P, N, P', N', C'_u \uplus \{c\})$.

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

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

Fig. 3. #CIRCUMSCRIPTION program.

Property B. There exists an atom $sat(v, \_, P, N, C, \_)$ in the lfp of the #CIRCUMSCRIPTION program iff $S(v, P, N, C) \neq \emptyset$. Moreover, a fact $unsat(v, i, P, N, \_, \_, \_, \_)$ is in the lfp only if also a fact $sat(v, i, P, N, \_, \_)$ is. Finally, if $S(v, P, N, C) \neq \emptyset$ then there exists a partition $\{S_{i_1}, \ldots, S_{i_n}\}$ with $n \geq 1$ of $S(v, P, N, C)$ which fulfills the following conditions:

1. A fact $sat(v, i, P, N, C, \_)$ is contained in the lfp iff $i \in \{i_1, \ldots, i_n\}$.

Program #CIRCUMSCRIPTION (continued)

/* branch node. */
$auxsat(v, i_1, i_2, P, N, C_1 \cup C_2, j_1 * j_2) \leftarrow bag(v, X, C)$, $child_1(v_1, v)$, $bag(v_1, X, C)$,
$sat(v_1, i_1, P, N, C_1, j_1)$, $child_2(v_2, v)$, $bag(v_2, X, C)$, $sat(v_2, i_2, P, N, C_2, j_2)$.
$auxunsat(v, i_1, i_2, P, N, P', N', C_1 \cup C_2) \leftarrow bag(v, X, C)$,
$child_1(v_1, v)$, $bag(v_1, X, C)$, $unsat(v_1, i_1, P, N, P', N', C_1)$,
$child_2(v_2, v)$, $bag(v_2, X, C)$, $unsat(v_2, i_2, P, N, P', N', C_2)$.
$auxunsat(v, i_1, i_2, P, N, P, N, C_1 \cup C_2) \leftarrow bag(v, X, C)$,
$child_1(v_1, v)$, $bag(v_1, X, C)$, $sat(v_1, i_1, P, N, C_1, \_)$,
$child_2(v_2, v)$, $bag(v_2, X, C)$, $unsat(v_2, i_2, P, N, P, N, C_2)$.
$auxunsat(v, i_1, i_2, P, N, P, N, C_1 \cup C_2) \leftarrow bag(v, X, C)$,
$child_1(v_1, v)$, $bag(v_1, X, C)$, $unsat(v_1, i_1, P, N, P, N, C_1)$,
$child_2(v_2, v)$, $bag(v_2, X, C)$, $sat(v_2, i_2, P, N, C_2, \_)$.

/* variable removal and branch node: $aux \Rightarrow sat$ */
$sat(v, i, P, N, C_u, j) \leftarrow auxsat(v, i_1, i_2, P, N, C_u, \_)$, $reduce(v, P, N, i, i_1, i_2, j)$.
$unsat(v, i, P, N, P', N', C'_u) \leftarrow auxunsat(v, i_1, i_2, P, N, P', N', C'_u)$,
$reduce(v, P, N, i, i_1, i_2, \_)$.

/* result (at the root node). */
$count(\text{SUM}(j)) \leftarrow root(v)$, $bag(v, X, C)$, $sat(v, i, P, N, C, j)$,
not $unsat(v, i, P, N, P', N', C)$.

Fig. 4. #CIRCUMSCRIPTION program.

2. The fact $sat(v, i, P, N, C, j)$ is contained in the lfp iff $|S_i| = j$.
3. For every partition (P', N') of $Var(v)$ and every subset $C' \subseteq Cl(v)$, the following two equivalences hold:

   The fact $unsat(v, i, P, N, P', N', C')$ is contained in the lfp $\Leftrightarrow$
   there exists a $J \in S_i$ and an assignment $J' \subset J$, s.t. $J' \in S(v, P', N', C') \Leftrightarrow$
   for all $J \in S_i$ there exists an assignment $J' \subset J$, s.t. $J' \in S(v, P', N', C')$.

Condition 2 above implies that, for any values v, i, P, N, C, there is at most one fact $sat(v, i, P, N, C, \_)$ in the lfp. Condition 3 ensures that, at the root node v of $\mathcal{T}$, either *all* j models described by a fact $sat(v, i, P, N, C, j)$ are minimal or *none* of them is.

The predicates *true* and *partition* have the same meaning as in the #SAT program. In addition, we have the predicates *auxsat*, *auxunsat*, and *reduce* with the following meaning: Recall that the index i in $sat(v, i, P, N, C, \_)$ is used to keep different assignments $J \in S(v, P, N, C)$ apart. Of course, in principle, there can be exponentially many such J. Nonetheless, the predicates *auxsat*, *auxunsat*, and *reduce* guarantee the fixed-parameter tractability in the following way. In the first place, we compute facts $auxsat(v, i_1, i_2, P, N, C, \_)$ and $auxunsat(v, i_1, i_2, P, N, P', N', C')$, where we use *pairs of indices* (i_1, i_2) rather than a single index i to associate the *auxunsat*-facts with the correct *auxsat*-fact. Now suppose that for two distinct pairs (i_1, i_2) and (i'_1, i'_2) a fact $auxsat(v, i_1, i_2, P, N, C, \_)$ and $auxsat(v, i'_1, i'_2, P, N, C, \_)$ exists in the lfp and, moreover, the *auxunsat*-facts for (v, i_1, i_2, P, N) and (v, i'_1, i'_2, P, N) are the same, i.e., for indices i, j, let $Val(v, i, j, P, N) = \{(P', N', C') \mid$ there exists a fact $auxunsat(v, i, j, P, N, P', N', C')$ in the lfp $\}$. Then $Val(v, i_1, i_2, P, N) = Val(v, i'_1, i'_2, P, N)$ holds. Intuitively, this means that the pairs of indices (i_1, i_2) and (i'_1, i'_2)

are not distinguishable by the *sat*- and *unsat*-conditions for this particular combination of (v, P, N). The purpose of the *reduce*-predicate is, in such a situation, to contract (i_1, i_2) and (i'_1, i'_2) to a single index i and to take care of the actual counting and summation. More precisely, a fact $reduce(v, P, N, i, i_1, i_2, j)$ means that the pair of indices (i_1, i_2) is mapped to the single index i and that j is the sum of all j' in facts $auxsat(v, i'_1, i'_2, P, N, C, j')$, s.t. (i'_1, i'_2) is mapped to i. In principle, the *reduce*-predicate predicate can be realized in datalog (see Appendix B). However, in the long run, an efficient implementation via hash tables inside the datalog processor is clearly preferable. The datalog program in Figure 3 and 4 solves the #CIRCUMSCRIPTION problem in the following way:

Theorem 3. *Let $\mathcal{C}$ be an instance of* #CIRCUMSCRIPTION*, encoded by a τ_{td}-structure $\mathcal{A}_{td}$. Then, $count(j)$ with $j \geq 1$ is in the lfp of the* #CIRCUMSCRIPTION*-program evaluated on $\mathcal{A}_{td}$ iff $\mathcal{C}$ is satisfiable and has exactly j (subset) minimal models. Moreover, both the construction of the τ_{td}-structure $\mathcal{A}_{td}$ and the evaluation of the program take time $\mathcal{O}(f(tw(\mathcal{C})) * \|\mathcal{C}\|)$ for some function f, if we assume constant runtime for the arithmetic operations.*

Proof. The proof is based on essentially the same ideas as the proof of Theorem 2. In particular, the correctness follows easily as soon as the correctness of Property B is established, which can be done by structural induction (see Appendix C). The linear time bound is again shown via Theorem 1 and the fact that the arithmetic operations required for the counting do not destroy the linear time data complexity. □

5 Horn abduction

Abduction is an important method in artificial intelligence and, in particular, in diagnosis. A propositional abduction problem (PAP) is given by a tuple $\mathcal{P} = \langle V, H, M, \mathcal{C} \rangle$, where V is a finite set of 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 propositional 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 [8], the decision problem (i.e., does a given PAP have a solution) of propositional abduction with bounded treewidth was considered. In order to illustrate the wide applicability of the datalog approach, we concentrate on the special case of #HORN-ABDUCTION, i.e., given a PAP $\mathcal{P}$ whose theory is a set of Horn clauses, count the number of solutions $\mathcal{S}$ of $\mathcal{P}$. The datalog program in Figure 5 has a significantly different flavour than the ones in the previous sections and can be considered as prototypical for rule-based problems.

Before we explain this program, we introduce some useful terminology and conventions: In general, Horn clauses are either rules, facts, or goals. For our purposes, it is convenient to consider every clause r of $\mathcal{C}$ as a *rule* consisting of a head (denoted as $head(r)$) and a body (denoted as $body(r)$). Goals of the form $\neg p_1 \vee \cdots \vee \neg p_k$ are thus considered as rules of the form $p_1 \wedge \cdots \wedge p_k \rightarrow \bot$ and a fact q in $\mathcal{C}$ is considered as a rule of the form $\rightarrow q$ with an empty body. A PAP is represented by a τ-structure with $\tau = \{cl, var, neg, pos, hyp, man\}$, where the predicates hyp and man indicate that some variable a is a hypothesis (i.e., $hyp(a)$) or a manifestation (i.e., $man(a)$). By the above consideration, $var(\bot)$ is now also fulfilled. Moreover, $neg(a, r)$ (resp. $pos(a, r)$)

Program #HORN-ABDUCTION

/* leaf node. */

$solve(v, S, 0, C^o, RC, \Delta C, RO_1 \cup RO_2, 1) \leftarrow leaf(v)$, $bag(v, X, R)$, $S \cap C^o = \emptyset$, $C^o \subseteq X$, $RC \subseteq R$, $svar(v, S)$, $explains(v, S \cup C^o)$, $consistent(RC, S, C^o, X)$, $derived(\Delta C, C^o, RC)$, $outside(RO_1, R, X \setminus (S \cup C^o))$, $inside(RO_2, R, S \cup C^o)$.

/* variable removal node. */

$aux(v, S, i, 0, C^o, RC, \Delta C, RO, j) \leftarrow bag(v, X, R)$, $child_1(v_1, v)$, $bag(v_1, X \uplus \{x\}, R)$, $solve(v_1, S \uplus \{x\}, i, C^o, RC, \Delta C, RO, j)$.

$aux(v, S, i, 1, C^o, RC, \Delta C, RO, j) \leftarrow bag(v, X, R)$, $child_1(v_1, v)$, $bag(v_1, X \uplus \{x\}, R)$, $solve(v_1, S, i, C^o \uplus \{x\}, RC, \Delta C \uplus \{x\}, RO, j)$.

$aux(v, S, i, 1, C^o, RC, \Delta C, RO, j) \leftarrow bag(v, X, R)$, $child_1(v_1, v)$, $bag(v_1, X \uplus \{x\}, R)$, $solve(v_1, S, i, C^o, RC, \Delta C, RO, j)$, $x \notin S$, $x \notin C^o$.

/* rule removal node. */

$solve(v, S, i, C^o, RC, \Delta C, RO, j) \leftarrow bag(v, X, R)$, $child_1(v_1, v)$, $bag(v_1, X, R \uplus \{r\})$, $solve(v_1, S, i, C^o, RC \uplus \{r\}, \Delta C, RO \uplus \{r\}, j)$.

$solve(v, S, i, C^o, RC, \Delta C, RO, j) \leftarrow bag(v, X, R)$, $child_1(v_1, v)$, $bag(v_1, X, R \uplus \{r\})$, $solve(v_1, S, i, C^o, RC, \Delta C, RO \uplus \{r\}, j)$.

/* variable introduction node. */

$solve(v, S \uplus \{x\}, i, C^o, RC, \Delta C, RO, j) \leftarrow bag(v, X \uplus \{x\}, R)$, $child_1(v_1, v)$, $bag(v_1, X, R)$, $solve(v_1, S, i, C^o, RC, \Delta C, RO, j)$, $hyp(x)$.

$solve(v, S, i, C^o \uplus \{x\}, RC, \Delta C, RO, j) \leftarrow bag(v, X \uplus \{x\}, R)$, $child_1(v_1, v)$, $bag(v_1, X, R)$, $solve(v_1, S, i, C^o, RC, \Delta C, RO, j)$, $consistent(RC, S, C^o \uplus \{x\}, X \uplus \{x\})$.

$solve(v, S, i, C^o, RC, \Delta C, RO_1 \cup RO_2, j) \leftarrow bag(v, X \uplus \{x\}, R)$, $child_1(v_1, v)$, $bag(v_1, X, R)$, $solve(v_1, S, i, C^o, RC, \Delta C, RO_1, j)$, not $man(x)$, $outside(RO_2, R, \{x\})$, $consistent(RC, S, C^o, X \uplus \{x\})$.

/* rule introduction node. */

$solve(v, S, i, C^o, RC \uplus \{r\}, \Delta C \uplus \{x\}, RO \uplus \{r\}, j) \leftarrow bag(v, X, R \uplus \{r\})$, $child_1(v_1, v)$, $bag(v_1, X, R)$, $solve(v_1, S, i, C^o, RC, \Delta C, RO, j)$, $consistent(RC \uplus \{r\}, S, C^o, X)$, $pos(x, r)$, $x \notin \Delta C$, $x \neq \bot$.

$solve(v, S, i, C^o, RC, \Delta C, RO_1 \cup RO_2 \cup RO_3, j) \leftarrow bag(v, X, R \uplus \{r\})$, $child_1(v_1, v)$, $bag(v_1, X, R)$, $solve(v_1, S, i, C^o, RC, \Delta C, RO_1, j)$, $outside(RO_2, R \uplus \{r\}, X \setminus (S \uplus C^o))$, $inside(RO_3, R \uplus \{r\}, S \uplus C^o)$.

/* branch node. */

$aux(v, S, i_1, i_2, C^o, RC, \Delta C_1 \uplus \Delta C_2, RO_1 \uplus RO_2, j_1 * j_2) \leftarrow bag(v, X, R)$, $child_1(v_1, v)$, $bag(v_1, X, R)$, $solve(v_1, S, i_1, C^o, RC, \Delta C_1, RO_1, j_1)$, $child_2(v_2, v)$, $bag(v_2, X, R)$, $solve(v_2, S, i_2, C^o, RC, \Delta C_2, RO_2, j_2)$, $derived(\Delta C, C^o, RC)$, $\Delta C_1 \cap \Delta C_2 = \Delta C$.

/* variable removal and branch node: $aux \Rightarrow solve$ */

$solve(v, S, i, C^o, RC, \Delta C, RO, j) \leftarrow aux(v, S, i_1, i_2, C^o, RC, \Delta C, RO, j')$, $reduce(v, S, i, i_1, i_2, j)$.

/* result (at the root node). */

$count(\text{SUM}(j)) \leftarrow root(v)$, $bag(v, X, R)$, $solve(v, S, i, C^o, RC, \Delta C, RO, j)$, $C^o = \Delta C$, $RO = R$, not $unsuccess(S, i, C^o)$.

$unsuccess(S, i, C_1^o) \leftarrow root(v)$, $bag(v, X, R)$, $solve(v, S, i, C_2^o, RC, \Delta C, RO, j)$, $C_2^o = \Delta C$, $RO = R$, $C_2^o < C_1^o$.

Fig. 5. #HORN-ABDUCTION program.

means that a occurs in the body of r (resp. in the head of r). For the input tree decomposition, we assume that a bag containing some rule r also contains the variable a in the head of r. This will greatly simplify the presentation of our datalog program and can, in the worst-case, only double the width of the resulting decomposition.

For $\mathcal{S} \subseteq V \cup \{\bot\}$, we write $\mathcal{S}^+$ to denote the *closure* of $\mathcal{S}$ w.r.t. the theory $\mathcal{C}$, i.e.: An element $q \in V \cup \{\bot\}$ is contained in $\mathcal{S}^+$ iff either $q \in \mathcal{S}$ or there exists a "derivation sequence" of q from $\mathcal{S}$ in $\mathcal{C}$ of the form $\mathcal{S} \to \mathcal{S} \cup \{q_1\} \to \mathcal{S} \cup \{q_1, q_2\} \to \ldots \to \mathcal{S} \cup \{q_1, \ldots, q_n\}$, s.t. $q_n = q$ and for every $i \in \{1, \ldots, n\}$, there exists a rule $r_i \in \mathcal{C}$ with $body(r_i) \subseteq \mathcal{S} \cup \{q_1, \ldots, q_{i-1}\}$ and $head(r_i) = q_i$. Hence, a subset $\mathcal{S} \subseteq H$ is a solution of the PAP $\mathcal{P}$ iff $\bot \notin \mathcal{S}^+$ and $M \subseteq \mathcal{S}^+$. Our #HORN-ABDUCTION program searches for the number of solutions $\mathcal{S} \subseteq H$ by applying precisely this criterion. The predicate $solve(v, S, i, C^o, RC, \Delta C, RO, j)$, which is at the heart of the #HORN-ABDUCTION program, has the following intended meaning: v denotes a node in the tree decomposition $\mathcal{T}$. S is the projection of a solution $\mathcal{S}$ onto $Hyp(v)$ and C^o is the projection of $\mathcal{S}^+ \setminus \mathcal{S}$ onto $Var(v)$. We consider $\mathcal{S}^+ \setminus \mathcal{S}$ as well as C^o as ordered (which is indicated by the superscript o) w.r.t. some derivation sequence of $\mathcal{S}^+$ from $\mathcal{S}$. The arguments $RC, \Delta C$, and RO are used to check that C^o is indeed the projection of $\mathcal{S}^+ \setminus \mathcal{S}$ onto $Var(v)$. Informally, the arguments RC and ΔC ensure that C^o is not too big, while RO ensures that C^o is not too small. These tasks are accomplished as follows: RC contains those rules in v which are used in the above derivation sequence. Furthermore, the set ΔC contains those variables of C^o, for which we have already found the corresponding derivation rule. Of course, in the bottom-up traversal of the tree decomposition, every element of C^o ultimately has to end up in ΔC. On the other hand, RO contains those rules r in the bag of v which do not constitute a contradiction with the closedness of $\mathcal{S}^+$, i.e., either the head of r is contained in $\mathcal{S}^+$ anyway or we have already encountered in $Var(\mathcal{T}_v)$ a variable in $body(r)$ which is not contained in $\mathcal{S}^+$. The last argument j is used to count the number of different solutions.

In the program, we again use $\cup$ and $\uplus$ to denote ordinary union resp. disjoint union. By $C^o \uplus \{x\}$, we mean that x is arbitrarily "inserted" into C^o, leaving the order of the remaining elements unchanged. Analogously to the #CIRCUMSCRIPTION program, we need an index i in order to distinguish between different derivation sequences leading to different orderings on the elements in $\mathcal{S}^+ \setminus \mathcal{S}$. Moreover, we need an *aux*-predicate maintaining pairs of indices in case of variable removable and branch nodes. Moreover, we also need a *reduce*-predicate to contract *aux*-facts $aux(v, S, i_1, i_2, \ldots)$ and $aux(v, S, i'_1, i'_2, \ldots)$ for partial solutions which are indistinguishable by the *aux*-facts in the lfp. The actual counting and summation is again done in the *reduce*-predicate (see Appendix D).

Formally, the correctness of the #HORN-ABDUCTION program can be shown via the Property C defined below. Let $Hyp(v)$, $Man(v)$, $Hyp(\mathcal{T}_v)$, and $Man(\mathcal{T}_v)$ denote the restriction of H and M to the variables in the bag of v or in any bag in $\mathcal{T}_v$, respectively. For arbitrary values of $v, S, C^o, RC, \Delta C$, and RO, we define the following set of extensions $\overline{S}$ of S to $Hyp(\mathcal{T}_v)$:

$Sol(v, S, C^o, RC, \Delta C, RO) = \{\ \overline{S} \mid S \subseteq \overline{S} \subseteq Hyp(\mathcal{T}_v)$ and $\exists \overline{C^o}\, \exists \overline{RC}$ with
$C^o \subseteq \overline{C^o} \subseteq Var(\mathcal{T}_v)$ and $RC \subseteq \overline{RC} \subseteq Cl(\mathcal{T}_v)$, s.t.

1. $\overline{S} \cap \overline{C^o} = \emptyset$, $\bot \notin \overline{C^o}$, and $Man(\mathcal{T}_v) \subseteq \overline{S} \cup \overline{C^o}$.
2. $\forall r \in \overline{RC}$, $head(r) \in \overline{C^o}$ and $\forall p \in body(r) \cap Var(\mathcal{T}_v)$: either $p \in \overline{S}$ or $p \in \overline{C^o}$ with $p < head(r)$.

3. $RO = \{r \in Cl(v) \mid body(r) \cap Var(\mathcal{T}_v) \not\subseteq \overline{S} \cup \overline{C^o}\} \cup \{r \in Cl(v) \mid head(r) \in S \cup C^o\}$ and $\forall r \in Cl(\mathcal{T}_v) \setminus Cl(v)$, if $head(r) \notin \overline{S} \cup \overline{C^o}$ then $body(r) \not\subseteq \overline{S} \cup \overline{C^o}$.
4. $\Delta C = \{p \in C^o \mid r \in \overline{RC}, head(r) = p\}$ and $\forall p \in \overline{C^o} \setminus C^o$, $\exists r \in \overline{RC}$ with $head(r) = p$. }

Then, occurrences of the ground facts $solve(v, S, i, C^o, RC, \Delta C, RO, j)$ in the lfp of #HORN-ABDUCTION are determined as follows:

Property C. If $Sol(v, S, C^o, RC, \Delta C, RO) = \emptyset$ then no atom $solve(v, S, \_, C^o, RC, \Delta C, RO, \_)$ is in the lfp of #HORN-ABDUCTION. On the other hand, if $Sol(v, S, C^o, RC, \Delta C, RO) \neq \emptyset$ then the following conditions are fulfilled:

(a) A fact $solve(v, S, \_, C^o, RC, \Delta C, RO, j)$ is in the lfp of #HORN-ABDUCTION iff $|Sol(v, S, C^o, RC, \Delta C, RO)| = j$.

(b) For any further tuple of values $(C_1^o, RC_1, \Delta C_1, RO_1)$ we have $Sol(v, S, C^o, RC, \Delta C, RO) = Sol(v, S, C_1^o, RC_1, \Delta C_1, RO_1)$ iff there exists an index i and a value j, s.t. there are facts $solve(v, S, i, C^o, RC, \Delta C, RO, j)$ and $solve(v, S, i, C_1^o, RC_1, \Delta C_1, RO_1, j)$ in the lfp of #HORN-ABDUCTION.

The other predicates have the following intended meaning: $svar(v, S)$ is used to select sets of hypotheses. It is true for every subset $S \subseteq Hyp(v)$. A fact $explains(v, X)$ is in the lfp iff $Man(v) \subseteq X$. These two predicates are only used to ease the notation at the leaf nodes of $\mathcal{T}$. The remaining predicates *consistent*, *outside*, *inside*, and *derived* take care of the conditions 2 – 4 of the definition of $Sol(v, S, C^o, RC, \Delta C, RO)$ in the following way: A fact $consistent(RC, S, C^o, X)$ is in the lfp iff $\forall r \in RC$ we have $head(r) \in C^o$ and $\forall p \in body(r) \cap X$ it holds that either $p \in S$ or $p \in C^o$ with $p < head(r)$, i.e. the rules in RC are only used to derive greater variables from smaller ones (plus variables from S), cf. condition 2 above. A fact $outside(RO, R, X)$ is in the lfp iff $RO = \{r \in R \mid body(r) \cap X \neq \emptyset\}$. Hence, for $X \subseteq V \setminus \mathcal{S}^+$, the rules in RO do not constitute a contradiction with the closedness of $\mathcal{S}^+$ because their bodies have a variable of X (and, therefore, outside $\mathcal{S}^+$) in their body. A fact $inside(RO, R, X)$ is in the lfp iff $RO = \{r \in R \mid head(r) \in X\}$. Hence, for $X \subseteq \mathcal{S}^+$, the rules in RO do not constitute a contradiction with the closedness of $\mathcal{S}^+$ because their head is inside this set. A fact $derived(\Delta C, C^o, RC)$ means that ΔC contains those variables of C^o for which RC already contains the rule which is used in the last step of the derivation, i.e., $\Delta C = \{q \in C^o \mid r \in RC, q = head(r)\}$. Analogously to Theorems 2 and 3, the #HORN-ABDUCTION-program in Figure 5 has the following properties:

Theorem 4. *Let $\mathcal{P} = \langle V, H, M, \mathcal{C}\rangle$ be an instance of* #HORN-ABDUCTION*, encoded by a τ_{td}-structure $\mathcal{A}_{td}$. Then, $count(j)$ with $j \geq 1$ is in the lfp of the* #HORN-ABDUCTION*-program evaluated on $\mathcal{A}_{td}$ iff the PAP $\mathcal{P}$ is solvable and has j solutions. Moreover, both the construction of the τ_{td}-structure $\mathcal{A}_{td}$ and the evaluation of the program take time $\mathcal{O}(f(tw(\mathcal{P})) * \|\mathcal{P}\|)$ for some function f, if we assume constant runtime for the arithmetic operations.*

6 Experimental Evaluation

A practical evaluation of the monadic datalog approach presented in earlier work [7] is still missing. So far, datalog programs (like the ones established in [7, 8]) only served as a "specification" for an implementation in C++, rather than being used as a method

tw	# vars	# clauses	# nodes	# models	Haskell	datalog
3	75	25	220	2.1E13	0.00	5.67
3	150	50	439	2.2E25	0.00	22.22
3	300	100	949	4.6E54	0.00	177.90
4	75	25	214	9.8E11	0.00	6.07
4	150	50	453	9.0E28	0.00	22.72
4	300	100	950	2.6E52	0.01	233.24
5	300	100	913	2.3E51	0.01	166.72
6	300	100	981	1.7E53	0.02	141.20
7	300	100	979	3.6E52	0.04	259.97
10	309	103	1044	5.1E48	4.12	2841.10

Table 1. Processing Time in sec. for #SAT.

of its own for solving the problem. However, using the datalog approach directly would be very appealing, for instance, for rapid prototyping. Below we report on some first lessons learned when experimenting with implementations of the #SAT program.

When evaluating the #SAT-program on a datalog engine, several obstacles have to be overcome: First, encodings for the non-standard datalog operations, especially those for set arithmetic, are non-trivial and must be done very carefully (avoiding the introduction of cycles, etc.). A recent extension of the DLV-system [11], which is called DLV-Complex (see http://www.mat.unical.it/dlv-complex), provides special built-in predicates for set arithmetic. Our experiments showed that such built-ins normally lead to a better performance than a direct realization of the #SAT-program in "pure" datalog. Another interesting observation was that the DLV-system did not recognize that the $solve()$-predicate can be evaluated without any cycles by a bottom-up traversal of the tree-decomposition. We therefore relaxed the separation of the program and the data and generated the programs using predicates $solve_v()$, for each node v in the tree-decomposition instead of having v as an argument in $solve()$ – thus making the acyclicity explicit. This led to a significant speed-up. Note that we could have put more and more computation tasks into the generation of the datalog program. However, to keep the method generic (w.r.t. different problems) we restricted ourselves to exploit only *structural information*, i.e. the shape of the tree decomposition. Further, DLV is handicapped in the way that no values bigger than 10^{10} can be processed.

We carried out experiments with two implementations of our #SAT program: one executing the datalog program directly on DLV-complex (compiling the tree structure into the program as discussed above) and one using a general-purpose, Turing complete programming language (in contrast to [7, 8], we used Haskell rather than C++, because we found it more convenient). Table 1 shows a glimpse of our results for various values of the treewidth (tw), number of variables (# vars), clauses (# clauses) and nodes in the tree decomposition (# nodes). The experiments were done on a recent Core2Duo processor with 2GB of RAM and two cores at 1.86 GHz. The time was measured with the Unix tool "time". DLV was called with the default optimization parameters. Haskell was compiled with increased optimization levels. Comparing a compiled program with an interpreted program might be "unfair", but the Haskell program does not need to be recompiled when the tree changes whereas the DLV program has to be generated for each instance.

In theory, our #SAT algorithm specified in terms of a datalog program is fixed-parameter linear whenever the program is evaluated in an "optimal" way. This is what our Haskell implementation does. For the time being, it is unclear how the design of the datalog program (or the underlying datalog engine) has to be changed such that the datalog engine yields similar results. This is subject of ongoing research. Nevertheless the datalog approach scales reasonably for instances of medium size. Therefore, already now, datalog engines can be employed as tools for rapid prototyping and to verify specifications, which are planned to be realized by a program in another language.

7 Conclusion

We have shown that the monadic datalog approach of [7] can be extended to counting problems defined via MSO. It should be noted that – as opposed to [13, 14] – our ultimate goal is not an efficient algorithm for the #SAT problem. Instead we are aiming at a general-purpose method which allows us to systematically turn theoretical tractability results based on Courcelle's Theorem and generalizations thereof into efficient computations. The experiments with our proof-of-concept implementation demonstrate that our goal is realistic even though there is still a lot of work ahead of us.

Analogously to [13, 14], our datalog programs ultimately follow a dynamic programming approach. This is not surprising if we keep the crucial observation underlying Courcelle's Theorem in mind: Consider a structure $\mathcal{A}$ with tree decomposition $\mathcal{T}$ and some node s in $\mathcal{T}$. If a domain element a in some bag above s and an element b below s jointly occur in some tuple in $\mathcal{A}$ then – by the definition of tree decompositions – b also occurs in the bag of s. Hence, the essential properties of the substructure induced by the subtree rooted at s can be described in terms of the elements in the bag of s – without taking the concrete form of the subtree rooted at s into account. Indeed, our #SAT program behaves very similar to the dynamic programming algorithm in [14] for the incidence graph. Nevertheless, we find the declarative style of datalog appealing and it has proved convenient in tackling not only $\#\mathsf{P}$ problems but also the $\#\mathsf{NP}$-problem #CIRCUMSCRIPTION. Moreover, the use of datalog allows us to take advantage of all future improvements of datalog engines, which is a very active research area [16].

As future work in this area, we are planning to prove a general expressivity result as to how monadic datalog has to be extended in order to be applicable to any MSO-based counting problem over structures with bounded treewidth. Moreover, we also want to integrate further extensions of Courcelle's Theorem (like sum, minimum, and maximum, which are studied in [3]) into the monadic datalog approach of [7]. As far as our implementation on top of DLV is concerned, we have already identified some directions of future work in Section 6. Note that we have so far used DLV only as a "black box" by converting a #SAT problem instance plus the extended datalog program for #SAT into the DLV syntax. Integrating some of the extensions into the datalog system itself (e.g., an efficient implementation of the *reduce*-predicate in Figures 4 and 5 via hash tables) would clearly help to improve the performance.

References

1. Courcelle, B.: Graph rewriting: An algebraic and logic approach. In: Handbook of Theoretical Computer Science, Volume B. Elsevier Science Publishers (1990) 193–242

2. Gottlob, G., Pichler, R., Wei, F.: Bounded treewidth as a key to tractability of knowledge representation and reasoning. In: Proc. AAAI'06. (2006) 250–256
3. Arnborg, S., Lagergren, J., Seese, D.: Easy problems for tree-decomposable graphs. J. Algorithms **12** (1991) 308–340
4. Flum, J., Frick, M., Grohe, M.: Query evaluation via tree-decompositions. J. ACM **49** (2002) 716–752
5. Frick, M., Grohe, M.: The complexity of first-order and monadic second-order logic revisited. In: Proc. LICS'02. (2002) 215–224
6. Grohe, M.: Descriptive and parameterized complexity. In: Proc. CSL'99. Volume 1683 of LNCS. (1999) 14–31
7. Gottlob, G., Pichler, R., Wei, F.: Monadic datalog over finite structures with bounded treewidth. In: Proc. PODS'07. (2007) 165–174
8. Gottlob, G., Pichler, R., Wei, F.: Abduction with bounded treewidth: From theoretical tractability to practically efficient computation. In: Proc. AAAI'08. (2008) 1541–1546
9. Durand, A., Hermann, M., Kolaitis, P.G.: Subtractive reductions and complete problems for counting complexity classes. Theor. Comput. Sci. **340** (2005) 496–513
10. Hermann, M., Pichler, R.: Counting complexity of propositional abduction. In: Proc. IJCAI 2007. (2007) 417–422
11. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. ACM Trans. Comput. Log. **7** (2006) 499–562
12. Courcelle, B., Makowsky, J.A., Rotics, U.: On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. Discrete Applied Mathematics **108** (2001) 23–52
13. Fischer, E., Makowsky, J.A., Ravve, E.V.: Counting truth assignments of formulas of bounded tree-width or clique-width. Discrete Applied Mathematics **156** (2008) 511–529
14. Samer, M., Szeider, S.: Algorithms for propositional model counting. In: Proc. LPAR'07. Volume 4790 of LNCS., Springer (2007) 484–498
15. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput. **25** (1996) 1305–1317
16. Gebser, M., Liu, L., Namasivayam, G., Neumann, A., Schaub, T., Truszczyński, M.: The first answer set programming system competition. In: Proc. LPNMR'07. Volume 4483 of LNCS., Springer (2007) 3–17

A Correctness Proof of the #SAT Program

The only part missing in the proof of Theorem 2 is that the predicate $sat()$ indeed has the intended meaning, i.e. Property A holds. We prove the "if"- and the "only if"-direction separately, and proceed in each part by structural induction on the tree decomposition:

"only if"-direction. Let the fact $sat(v, P, N, C_u, j)$ be in the lfp of the program. We have to show that j is an integer with $j \geq 1$ and the set $S(v, P, N, C_u)$ indeed has the cardinality $|S(v, P, N, C_u)| = j$.

base case. Let v be a leaf node. Then $j = 1$ and the facts $partition(X, P, N)$ with $X = Var(v)$ and $true(P, N, C_u, C)$ with $C = Cl(v)$ are also in the lfp. This means, P and N form a partition of $Var(v)$ and therefore represent a truth value assignment (P, N). Since v is a leaf node, $Var(v) = Var(\mathcal{T}_v)$ and thus the only extension J of (P, N) to $Var(\mathcal{T}_v)$ is (P, N) itself. Furthermore the atom $true$ ensures that C_u contains exactly those clauses $c \in Cl(v)$, for which c is true in J. Therefore $S(v, P, N, C_u) = \{J\}$ and hence, $|S(v, P, N, C_u)| = 1 = j$.

induction step. We distinguish between the five different types of inner nodes of $\mathcal{T}$ assuming that Property A holds for all child nodes.

(1) Let v be a *variable removal node* with removal of variable x and let v_1 denote the child of v. We have to distinguish three cases. In the first case the two atoms $sat(v_1, P \uplus \{x\}, N, C_u, j_1)$ and $sat(v_1, P, N \uplus \{x\}, C_u, j_2)$ are both in the lfp and $j = j_1 + j_2$. By the induction hypothesis both $j_1 \geq 1$ and $j_2 \geq 1$. Furthermore the sets $S_1 = S(v_1, P \uplus \{x\}, N, C_u)$ and $S_2 = S(v_1, P, N \uplus \{x\}, C_u)$ have cardinality $|S_1| = j_1$ and $|S_2| = j_2$ respectively. By definition of $S = S(v, P, N, C_u)$ it holds that $S_1 \subseteq S$ and $S_2 \subseteq S$. Note that $S_1 \cap S_2 = \emptyset$ since their elements differ on the truth value assignment of x. Let $J \in S$ be an extension of (P, N) to $Var(\mathcal{T}_v)$. Then either $J(x) =$ true or $J(x) =$ false. But then by definition of S, either $J \in S_1$ or $J \in S_2$ and therefore $S = S_1 \uplus S_2$. Hence $|S| = j_1 + j_2 = j \geq 2$.

In the second case $sat(v_1, P \uplus \{x\}, N, C_u, j_1)$ is in the lfp but the fact $sat(v_1, P, N \uplus \{x\}, C, j_2)$ is not and $j = j_1$. By the induction hypothesis $j_1 \geq 1$. Furthermore the sets $S_1 = S(v_1, P \uplus \{x\}, N, C_u)$ and $S_2 = S(v_1, P, N \uplus \{x\}, C_u)$ have cardinality $|S_1| = j_1$ and $|S_2| = 0$ respectively. By definition of $S = S(v, P, N, C_u)$ it holds that $S_1 \subseteq S$. Let $J \in S$ be an extension of (P, N) to $Var(\mathcal{T}_v)$. Then either $J(x) =$ true or $J(x) =$ false. But then by definition of S, either $J \in S_1$ or $J \in S_2$. Since $S_2 = \emptyset$, $S = S_1$ and $|S| = j_1 = j \geq 1$.

In the third case $sat(v_1, P, N \uplus \{x\}, C_u, j_1)$ is in the lfp but the predicate $sat(v_1, P \uplus \{x\}, N, C_u, j_2)$ is not and $j = j_1$. This case is treated analogously to the second one.

(2) Let v be a *clause removal node* with removal of clause c and let v_1 denote the child of v. Then the fact $sat(v_1, P, N, C_u \uplus \{c\}, j)$ is also in the lfp. By the induction hypothesis the set $S_1 = S(v_1, P, N, C_u \uplus \{c\})$ has cardinality $|S_1| = j \geq 1$. Note that $(Cl(\mathcal{T}_{v_1}) \setminus Cl(v_1)) \cup C_u \uplus \{c\} = (Cl(\mathcal{T}_v) \setminus Cl(v)) \cup C_u$ and $Cl(v_1) \setminus (C_u \uplus \{c\}) = Cl(v) \setminus C_u$. Hence by the definition of $S = S(v, P, N, C_u)$ it holds that $S_1 = S$ and therefore $|S| = j \geq 1$.

(3) Let v be a *variable introduction node* with introduction of variable x and let v_1 denote the child of v. We have to distinguish two cases depending on whether $x \in P$ or $x \in N$. In the first case P is of the form $P' \uplus \{x\}$ with a set $P' \subseteq Var(v_1)$ and in the second case N is of the form $N' \uplus \{x\}$ with a set $N' \subseteq Var(v_1)$. We only treat the first

case here. The second one is symmetrical. In the first case the atom $true(\{x\}, \emptyset, C', C)$ with $C = Cl(v)$ is also in the lfp. This means C' contains exactly those clauses in $Cl(v)$ where the variable x occurs in unnegated form. Additionally the following mutually distinct facts are in the lfp: $sat(v_1, P', N, C_1, j_1)$, $sat(v_1, P', N, C_2, j_2), \ldots,$ $sat(v_1, P', N, C_n, j_n)$ with $n \geq 1$. In particular, it holds that $C_k \neq C_l$ for all $k \neq l$ with $k, l \in \{1, \ldots, n\}$. Furthermore we have $\sum_{k=1}^{n} j_k = j$ and it holds that $C_k \cup C' = C_u$ for $k = 1, \ldots, n$. By the induction hypothesis the sets $S_k = S(v_1, P', N, C_k)$ have cardinality $|S_k| = j_k \geq 1$. We define a function f that transforms a truth value assignment J on $Var(\mathcal{T}_{v_1})$ into a truth value assignment on $Var(\mathcal{T}_v)$ in the following way: $f(J)(y) = J(y)$ for all $y \in Var(\mathcal{T}_{v_1})$ and $f(J)(x) =$ true. Now let the sets $S'_k = \{f(J) \mid J \in S_k\}$. Of course it holds that $|S_k| = |S'_k|$. By definition of $S = S(v, P, N, C_u)$ it is true that for all S'_k we have $S'_k \subseteq S$.

We claim that $S'_k \cap S'_l = \emptyset$ for all $k \neq l$. To see this, let $J' \in S'_k \cap S'_l$ and $k \neq l$. This means, there exists a truth value assignment J on $Var(\mathcal{T}_{v_1})$ with $J' = f(J)$. Therefore it holds that $J \in S_k \cap S_l$. But then, by the construction of S_k and S_l, we have that $C_k = C_l$ which implies that $k = l$ and therefore is a contradiction to the assumption that $k \neq l$.

Now let $J' \in S$ be an extension of (P, N) to $Var(\mathcal{T}_v)$. We claim that there exists a k for which $J' \in S'_k$. To see this, let J be the extension of (P', N) to $Var(\mathcal{T}_{v_1})$ for which $J' = f(J)$. Now let $C_k \subseteq Cl(v_1)$ contain exactly those clauses $c \in Cl(v_1)$, for which c is true in J. But then it holds that $J \in S_k$ and therefore $J' \in S'_k$. Hence S has the form $S = \biguplus_{k=1}^{n} S'_k$ with cardinality $|S| = (\sum_{k=1}^{n} j_k) = j \geq n$.

(4) Let v be a *clause introduction node* with introduction of clause c and let v_1 denote the child of v. Then the facts $sat(v_1, P, N, C_1, j)$ and $true(P, N, C_2, \{c\})$ with $C_u = C_1 \cup C_2$ are also in the lfp. The latter one means that $C_2 = \{c\}$ if $c|_{Var(v)}$ is true in (P, N) and $C_2 = \emptyset$ otherwise. Note that 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). By the induction hypothesis the set $S_1 = S(v_1, P, N, C_1)$ has cardinality $|S_1| = j \geq 1$.

We claim that $S_1 \subseteq S$ with $S = S(v, P, N, C_u)$. To see this, first consider the case that $c|_{Var(v)}$ is false in (P, N). Hence, $C_2 = \emptyset$ and, therefore, $C_u = C_1$. But then, by the definition of S, it holds that $S = S_1$. Now consider the case that $c|_{Var(v)}$ is true in (P, N). Then, $C_2 = \{c\}$ and C_u is of the form $C = C_1 \cup \{c\}$. Again, by the definition of S, we have $S = S_1$. Hence S has cardinality $|S| = j \geq 1$.

(5) Let v be a *branch node* and let v_1 and v_2 denote the children of v. Then the following mutually distinct pairs of atoms are also in the lfp: $\langle sat(v_1, P, N, C_{11}, j_{11}),$ $sat(v_2, P, N, C_{12}, j_{12})\rangle, \ldots, \langle sat(v_1, P, N, C_{n1}, j_{n1}),\ sat(v_2, P, N, C_{n2}, j_{n2})\rangle$ with $n \geq 1$ and $\sum_{k=1}^{n} j_{k1} * j_{k2} = j$ and it holds that $C_{k1} \cup C_{k2} = C_u$ for $k = 1, \ldots, n$. By the induction hypothesis the sets $S_{k1} = S(v_1, P, N, C_{k1})$ and $S_{k2} = S(v_2, P, N, C_{k2})$ have cardinality $|S_{k1}| = j_{k1} \geq 1$ and $|S_{k2}| = j_{k2} \geq 1$ respectively. We define a function f that combines an extension J_1 of (P, N) to $Var(\mathcal{T}_{v_1})$ and an extension J_2 of (P, N) to $Var(\mathcal{T}_{v_2})$ into an extension of (P, N) to $Var(\mathcal{T}_v)$ in the following way:

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

Note that $f(J_1, J_2)$ is proper defined for all $x \in Var(\mathcal{T}_v)$ since by the connectedness condition $Var(\mathcal{T}_{v_1}) \cap Var(\mathcal{T}_{v_2}) = Var(v)$.

Now let J_{k1} and J_{k2} be two arbitrary elements of S_{k1} and S_{k2} respectively. We claim that then $f(J_{k1}, J_{k2}) \in S$ with $S = S(v, P, N, C_u)$. To see this, note that for each $c \in (Cl(\mathcal{T}_v) \setminus Cl(v)) \cup C_u$ it holds either that $c \in (Cl(\mathcal{T}_{v_1}) \setminus Cl(v_1)) \cup C_{k1}$ or that $c \in (Cl(\mathcal{T}_{v_2}) \setminus Cl(v_2)) \cup C_{k2}$. Therefore c is true in $f(J_{k1}, J_{k2})$. On the other hand, for each $c \in Cl(v) \setminus C_u$, the restriction $c|_{Var(\mathcal{T}_{v_1})}$ is false in J_{k1} and the restriction $c|_{Var(\mathcal{T}_{v_2})}$ is false in J_{k2}. Hence $c|_{Var(\mathcal{T}_v)}$ is false in $f(J_{k1}, J_{k2})$. But then, by the definition of S, it holds that $f(J_{k1}, J_{k2}) \in S$. Since there are j_{k1} possibilities to choose J_{k1} and j_{k2} possibilities to choose J_{k2}, there are $j_{k1} * j_{k2}$ different extensions $f(J_{k1}, J_{k2})$ of (P, N) to $Var(\mathcal{T}_v)$ for each $k = 1, \ldots, n$.

Let $\langle S_{k1}, S_{k2} \rangle$ and $\langle S_{l1}, S_{l2} \rangle$ be two arbitrary pairs with $k, l \in \{1, \ldots, n\}$ and $k \neq l$. It is easy to see, that the extensions $f(J_{k1}, J_{k2})$ and $f(J_{l1}, J_{l2})$ of (P, N) to $Var(\mathcal{T}_v)$ that can be constructed by combining arbitrary $J_{k1} \in S_{k1}$, $J_{k2} \in S_{k2}$ and $J_{l1} \in S_{l1}$, $J_{l2} \in S_{l2}$ are indeed mutually distinct. Otherwise it holds that $J_{k1} = J_{l1}$ and $J_{k2} = J_{l2}$. But by the definition of S_{k1}, S_{k2}, S_{l1} and S_{l2} this means that $S_{k1} = S_{l1}$ and $S_{k2} = S_{l2}$. This implies that the corresponding pairs of facts $\langle sat(v_1, P, N, C_{k1}, j_{k1}), sat(v_2, P, N, C_{k2}, j_{k2}) \rangle$ and $\langle sat(v_1, P, N, C_{l1}, j_{l1}), sat(v_2, P, N, C_{l2}, j_{l2}) \rangle$ are equal, which in turn implies that $k = l$. Therefore we have a contradiction with our assumption that $k \neq l$.

Finally let $J \in S$ be an arbitrary extension of (P, N) to $Var(\mathcal{T}_v)$. Then there exists an extension J_1 of (P, N) to $Var(\mathcal{T}_{v_1})$ and an extension J_2 of (P, N) to $Var(\mathcal{T}_{v_2})$, s.t. $J = f(J_1, J_2)$. Furthermore let $C_1 \subseteq C_u$ contain exactly those clauses $c \in C_u$ which are true in J_1 and let $C_2 \subseteq C_u$ contain exactly those clauses $c \in C_u$ which are true in J_2. Clearly we have $C_1 \cup C_2 = C_u$. But then the lfp contains a pair of facts $\langle sat(v_1, P, N, C_1, j_1), sat(v_2, P, N, C_2, j_2) \rangle$ and the truth value assignments J_1 and J_2 are contained in the corresponding sets $S(v_1, P, N, C_1)$ and $S(v_2, P, N, C_2)$ respectively. Therefore and with the arguments above, we have that S has cardinality $|S| = \sum_{k=1}^{n} j_{k1} * j_{k2} = j \geq n$.

"if"-direction. Let the cardinality of the set $S = S(v, P, N, C_u)$ be $|S| = j$ with $j \geq 0$. We have to show that the fact $sat(v, P, N, C_u, j)$ is in the lfp of the program if $j \neq 0$ and that otherwise no fact of the form $sat(v, P, N, C_u, \_)$ is in the lfp.

base case. Let v be a leaf node and let $j \geq 1$. Since P and N form a partition of $Var(v)$, the only extension $J \in S$ of (P, N) to $Var(\mathcal{T}_v) = Var(v)$ is (P, N) itself. Therefore $j = 1$ and the fact $partition(X, P, N)$ with $X = Var(v)$ is in the lfp of the program. By the definition of S, C_u contains exactly those clauses $c \in Cl(v)$, for which $c|_{Var(v)}$ is true in $J = (P, N)$. Hence the atom $true(P, N, C_u, C)$ with $C = cl(v)$ is in the lfp, which implies that also the fact $sat(v, P, N, C_u, 1)$ is in the lfp.

On the other hand let $j = 0$. This means, C_u does not contain exactly those clauses $c \in Cl(v)$, for which $c|_{Var(v)}$ is true in (P, N). Therefore the atom $true(P, N, C_u, C)$ with $C = cl(v)$ is not in the lfp. Hence no fact of the form $sat(v, P, N, C_u, \_)$ can be derived.

induction step. We distinguish again between the five different types of inner nodes of $\mathcal{T}$ assuming that Property A holds for all child nodes.

(1) Let v be a *variable removal node* with removal of variable x and let v_1 denote the child of v. We define two subsets $S_1, S_2 \subseteq S$ in the following way: $S_1 = \{J \in S \mid J(x) = \text{true}\}$ and $S_2 = \{J \in S \mid J(x) = \text{false}\}$ with cardinality $|S_1| = j_1$ and $|S_2| =$

j_2. Clearly $S_1 \uplus S_2 = S$ and therefore $j = j_1 + j_2$. Note that by definition it holds that $S_1 = S(v_1, P \uplus \{x\}, N, C_u)$ and $S_2 = S(v_1, P, N \uplus \{x\}, C_u)$. Thus by the induction hypothesis, the facts $sat(v_1, P \uplus \{x\}, N, C_u, j_1)$ and $sat(v_1, P, N \uplus \{x\}, C_u, j_2)$ are in the lfp of the program if $j_1 \geq 1$ and $j_2 \geq 1$ respectively. Otherwise no atoms of the form $sat(v_1, P \uplus \{x\}, N, C_u, \_)$ and $sat(v_1, P, N \uplus \{x\}, C_u, \_)$ are in the lfp. Now let $j \geq 1$. We have to distinguish three cases depending on whether none or exactly one of the two cardinalities j_1 and j_2 equals 0. It is easy to see, that in all those cases the corresponding rule implies that $sat(v, P, N, C_u, j)$ is in the lfp of the program. On the other hand let $j = 0$. Since no atoms of the form $sat(v_1, P \uplus \{x\}, N, C_u, \_)$ and $sat(v_1, P, N \uplus \{x\}, C_u, \_)$ are in the lfp, no fact of the form $sat(v, P, N, C_u, \_)$ can be derived.

(2) Let v be a *clause removal node* with removal of clause c and let v_1 denote the child of v. Note that $(Cl(\mathcal{T}_v) \setminus Cl(v)) \cup C_u = (Cl(\mathcal{T}_{v_1}) \setminus Cl(v_1)) \cup C_u \uplus \{c\}$ and that $Cl(v) \setminus C_u = Cl(v_1) \setminus (C_u \uplus \{c\})$. Therefore by definition $S = S_1 = S(v_1, P, N, C_u \uplus \{c\})$ with $|S| = |S_1| = j$. Assume that $j \geq 1$. Then by the induction hypothesis, $sat(v_1, P, N, C_u \uplus \{c\}, j)$ is in the lfp of the program and therefore $sat(v, P, N, C_u, j)$ is also in the lfp. On the other hand, let $j = 0$. Then by the induction hypothesis, no fact of the form $sat(v_1, P, N, C_u \uplus \{c\}, \_)$ is in the lfp and therefore no atom of the form $sat(v, P, N, C_u, \_)$ can be derived.

(3) Let v be a *variable introduction node* with introduction of variable x and let v_1 denote the child of v. We have to distinguish two cases depending on whether $x \in P$ or $x \in N$. We only treat the first case here. The second one is symmetrical. We define a clause set $C(J)$ for all assignments $J \in S$ as follows: $C(J) = \{c \in C_u \mid c|_{Var(\mathcal{T}_{v_1})}$ is true in $J|_{Var(\mathcal{T}_{v_1})}\}$. Thereby $J|_{Var(\mathcal{T}_{v_1})}$ denotes the restriction of J to $Var(\mathcal{T}_{v_1})$. Assuming that $j \geq 1$, we construct a partition $\{S_1, \ldots, S_n\}$ of S with $n \geq 1$, s.t. for all $J_1, J_2 \in S_k$ with $k = 1, \ldots, n$ it holds that $C(J_1) = C(J_2)$ and that for all $J_1 \in S_k$ and $J_2 \in S_l$ with $k \neq l$ it holds that $C(J_1) \neq C(J_2)$. This means we can associate with each partition block S_k a clause set $C_k = C(J)$ with $J \in S_k$. Let the sets S_k have cardinality $|S_k| = j_k \geq 1$. Clearly we have $j = \sum_{k=1}^{n} j_k$. Let P be of the form $P = P' \uplus \{x\}$. Then it is easy to see, that $S_k = S(v_1, P', N, C_k)$. Therefore by the induction hypothesis, the fact $sat(v_1, P', N, C_k, j_k)$ is in the lfp of the program. Next let C' be defined as $C' = \{c \in Cl(v) : x$ occurs unnegated in $c\}$. Then $C' \subseteq C_u$ since otherwise there would exist a clause $c \in Cl(v) \setminus C_u$ with $c|_{Var(\mathcal{T}_v)}$ being true in each $J \in S$. Furthermore $C_u = C_k \cup C'$ holds for each $k = 1, \ldots, n$. This is due to the fact that all clauses in C_u are true in each $J \in S$. Hence, they are already true in $J|_{Var(\mathcal{T}_{v_1})}$ or they become true in J because of an unnegated occurrence of x. Thus the facts $true(\{x\}, \emptyset, C', C)$ with $C = Cl(v)$ and $C_k \cup C' = C_u$ are in the lfp. This implies that $sat(v, P, N, C_u, j)$ is also in the lfp.

Now assume that $S = \emptyset$ and let C' be defined as above. We claim that for each clause set C_k with $C_k \cup C' = C_u$ we have that $S_k = S(v_1, P', N, C_k) = \emptyset$. To see this, we define an extension J' of (P, N) to $Var(\mathcal{T}_v)$ by extending $J \in S_k$ by $J'(x) =$ true. But then we have $J' \in S$ which is a contradiction to our assumption. Therefore no fact of the form $sat(v_1, P', N, C_k, \_)$ is in the lfp and hence also no atom of the form $sat(v, P, N, C_u, \_)$ is in the lfp of the program.

(4) Let v be a *clause introduction node* with introduction of clause c and let v_1 denote the child of v. Let C_1 be defined as $C_1 = C_u \setminus \{c\}$. Then we have $(Cl(\mathcal{T}_{v_1}) \setminus Cl(v_1)) \cup C_1 = (Cl(\mathcal{T}_v) \setminus Cl(v)) \cup C_1 \subseteq (Cl(\mathcal{T}_v) \setminus Cl(v)) \cup C_u$ and furthermore $Cl(v_1) \setminus$

$C_1 = Cl(v) \setminus (C_1 \uplus \{c\}) \subseteq Cl(v) \setminus C_u$. Thus for every $J \in S$ it holds that $J \in S_1$ with $S_1 = (v_1, P, N, C_1)$ and hence $S \subseteq S_1$. Let C_2 be defined as $C_2 = \{c\}$ if $c|_{Var(v)}$ is true in (P, N) and $C_2 = \emptyset$ otherwise. Now assume that $C_u = C_1 \cup C_2$, i.e. $c \in C_u$ iff $c|_{Var(v)}$ is true in (P, N). But then clearly for all $J \in S_1$ it holds that $J \in S$. Therefore and since $S \subseteq S_1$, we have $S = S_1$ and $|S| = |S_1| = j$. By the induction hypothesis, if $j \geq 1$ the fact $sat(v_1, P, N, C_1, j)$ is in the lfp of the program. Additionally by our construction, the atoms $true(P, N, C_2, \{c\})$ and $C_1 \cup C_2 = C_u$ are also in the lfp. Therefore the fact $sat(v, P, N, C_u, j)$ will be derived. On the other hand, if $j = 0$ no fact of the form $sat(v_1, P, N, C_1, \_)$ is in the lfp. Hence no fact of the form $sat(v, P, N, C_u, \_)$ can be derived.

The only case left, is the one where $C_u \neq C_1 \cup C_2$, i.e. $c \in C_u$ iff $c|_{Var(v)}$ is false in (P, N). In this case by definition $S = \emptyset$, i.e. $|S| = 0$. But then the fact $C_1 \cup C_2 = C_u$ is not in the lfp of the program and therefore no atom of the form $sat(v, P, N, C_u, \_)$ can be derived.

(5) Let v be a *branch node* and let v_1 and v_2 denote the children of v. We define the set Let $\mathcal{C} = \{(C'_1, C''_1), (C'_2, C''_2), \ldots, (C'_n, C''_n)\}$ of pairs of clause sets as follows: $\mathcal{C} = \{(C', C'') \mid C' \cup C'' = C_u, S(v_1, P, N, C') \neq \emptyset, S(v_2, P, N, C'') \neq \emptyset\}$. For every $\alpha \in \{1, \ldots, n\}$, let $S'_\alpha = S(v_1, P, N, C'_\alpha)$ and $S''_\alpha = S(v_2, P, N, C''_\alpha)$. Moreover, let $j'_\alpha = |S'_\alpha|$ and $j''_\alpha = |S''_\alpha|$. By the induction hypothesis, for every $\alpha \in \{1, \ldots, n\}$, there are pairs of facts in the lfp of the program, which have the form $\big(sat(v_1, P, N, C'_\alpha, j'_\alpha), sat(v_1, P, N, C''_\alpha, j''_\alpha)\big)$. Moreover, the lfp does not contain any other facts of this type, i.e., suppose that there exist clause sets C', C'' with $C' \cup C'' = C_u$ and positive integers j', j'', s.t. the lfp of the program contains two atoms $sat(v_1, P, N, C', j')$ and $sat(v_1, P, N, C'', j'')$, then there exists an index $\alpha \in \{1, \ldots, n\}$ with $(C', C'') = (C'_\alpha, C''_\alpha)$ and $(j', j'') = (j'_\alpha, j''_\alpha)$. Thus, the rule for branch nodes in the #SAT program produces the fact $sat(v, P, N, C_u, k)$ with $k = \sum_{\alpha=1}^{n} j'_\alpha * j''_\alpha$.

It remains to show that $j = k$, i.e., k indeed denotes the cardinality of the set S. Let S' be defined as

$$S' = S'_1 \times S''_1 \cup S'_2 \times S''_2 \cup \cdots \cup S'_n \times S''_n.$$

Clearly, any two sets $(S'_\alpha \times S''_\alpha)$ and $(S'_\beta \times S''_\beta)$ with $\alpha \neq \beta$ are disjoint. This follows immediately from the fact that any pair of interpretations $(J'_\alpha \times J''_\alpha) \in (S'_\alpha \times S''_\alpha)$ makes the clauses in $(C'_\alpha \times C''_\alpha)$ true while any pair of interpretations $(J'_\beta \times J''_\beta) \in (S'_\beta \times S''_\beta)$ makes the clauses in $(C'_\beta \times C''_\beta)$ true with $(C'_\alpha \times C''_\alpha) \neq (C'_\beta \times C''_\beta)$. Thus, $|S'| = k$ holds and it suffices to show that there exists a bijection $S' \cong S$.

Let (J', J'') be an arbitrary pair of assignments in S'. Then we write $J' \cup J''$ to denote the assignment J on $Var(\mathcal{T}_v)$, s.t. J restricted to $Var(\mathcal{T}_{v_1})$ is J' and J restricted to $Var(\mathcal{T}_{v_2})$ is J''. By the connectedness condition on tree decompositions and by the fact that J' and J'' coincide on the variables in $Var(v) = Var(v_1) = Var(v_2)$, this assignment J is well-defined. We claim that the desired bijective function $f\colon S' \to S$ is obtained by defining $f(J', J'') = J' \cup J''$. Clearly, f is well-defined and $f(J', J'') \in S$ holds.

Moreover, f is surjective and injective for the following reasons: Let J be an arbitrary assignment in S and let $J|_{Var(\mathcal{T}_{v_1})}$ (resp. $J|_{Var(\mathcal{T}_{v_2})}$) denote the restriction of J to the variables in $Var(\mathcal{T}_{v_1})$ (resp. $Var(\mathcal{T}_{v_2})$). Then it clearly holds that we have $(J|_{Var(\mathcal{T}_{v_1})}, J|_{Var(\mathcal{T}_{v_2})}) \in S'$ and $J = f(J|_{Var(\mathcal{T}_{v_1})}, J|_{Var(\mathcal{T}_{v_2})})$ is true. Hence, f is indeed surjective. Now let (J'_1, J''_1) and (J'_2, J''_2) be two distinct pairs in S'. It remains

to show that then $J'_1 \cup J''_1$ and $J'_2 \cup J''_2$ are also distinct. Since $(J'_1, J''_1) \neq (J'_2, J''_2)$, we either have $J'_1 \neq J'_2$ or $J''_1 \neq J''_2$. W.l.o.g., suppose that $J'_1 \neq J'_2$. Hence, there exists a variable x in $Var(\mathcal{T}_{v_1}) \setminus Var(v_1)$ which has different truth values in J'_1 and J'_2. But then x also has different truth values in $J'_1 \cup J''_1$ and $J'_2 \cup J''_2$. Hence, $J'_1 \cup J''_1 \neq J'_2 \cup J''_2$ indeed holds and, therefore, f is injective. □

B Auxiliary Predicates for the #CIRCUMSCRIPTION Program

A datalog implementation of the predicate *reduce* for the #CIRCUMSCRIPTION program plus several auxiliary predicates is given in Figure 6. The idea of the predicate $reduce(v, P, N, i, i_1, i_2, j)$ is to map pairs of indices (i_1, i_2) to single indices i and, in particular, to map "equal" (i.e., indistinguishable) pairs of indices (i_1, i_2) to the same index i. For given values of v, P, N, two pairs (i_1, i_2) and (i'_1, i'_2) are "equal" iff two facts of the form $auxsat(v, i_1, i_2, P, N, C, j)$ and the form $auxsat(v, i'_1, i'_2, P, N, C, j')$ are in the lfp of the program and it holds that $auxunsat(v, i_1, i_2, P, N, P', N', C')$ is in the lfp iff the lfp contains an atom $auxunsat(v, i'_1, i'_2, P, N, P', N', C')$.

Program Auxiliary Predicates for #CIRCUMSCRIPTION

$diff(v, P, N, i_1, i_2, i'_1, i'_2) \leftarrow auxunsat(v, i_1, i_2, P, N, P', N', C)$,
not $auxunsat(v, i'_1, i'_2, P, N, P', N', C)$, $0 \leq i'_1, i'_2 < K$.

$diff(v, P, N, i_1, i_2, i'_1, i'_2) \leftarrow diff(v, P, N, i'_1, i'_2, i_1, i_2)$.

$equal(v, P, N, i_1, i_2, i'_1, i'_2) \leftarrow auxsat(v, i_1, i_2, P, N, C, \_)$, $auxsat(v, i'_1, i'_2, P, N, C, \_)$,
not $diff(v, P, N, i_1, i_2, i'_1, i'_2)$.

$count\_reduced(v, P, N, i_1, i_2, \text{SUM}(j)) \leftarrow equal(v, P, N, i_1, i_2, i'_1, i'_2)$,
$auxsat(v, i'_1, i'_2, P, N, \_, j)$.

$less(i_1, i_2, i'_1, i'_2) \leftarrow i_1 < i'_1, 0 \leq i_1, i_2, i'_1, i'_2 < K$.
$less(i, i_2, i, i'_2) \leftarrow i_2 < i'_2, 0 \leq i_1, i_2, i'_1, i'_2 < K$.

$reduced\_smaller(v, P, N, i, i_1, i_2) \leftarrow less(i'_1, i'_2, i_1, i_2)$, $reduce(v, P, N, i, i'_1, i'_2, j)$.

$duplicate(v, P, N, i_1, i_2) \leftarrow less(i'_1, i'_2, i_1, i_2)$, $equal(v, P, N, i_1, i_2, i'_1, i'_2)$.

$open(v, P, N, i, i_1, i_2) \leftarrow i' < i$, not $reduced\_smaller(v, P, N, i', i_1, i_2)$, $0 \leq i, i' < K$.

$reduce(v, P, N, i, i_1, i_2, j) \leftarrow 0 \leq i < K$, $count\_reduced(v, P, N, i_1, i_2, j)$,
not $reduced\_smaller(v, P, N, i, i_1, i_2)$, not $duplicate(v, P, N, i_1, i_2)$,
not $open(v, P, N, i, i_1, i_2)$.

Fig. 6. *reduce* plus auxiliary predicates for #CIRCUMSCRIPTION.

In the body of the rule defining $reduce(v, P, N, i, i_1, i_2, j)$, we have the condition $0 \leq i < K$, where K denotes the maximum number of non-equal pairs (i_1, i_2). This K is a constant which depends solely on the treewidth w. A suitable value of K can be obtained by the following considerations: For given values of v, P, N, let the bag at v consist of the variables $Var(v)$ and clauses $Cl(v)$ with $Var(v) \cap Cl(v) = \emptyset$ and $|Var(v) \cup Cl(v)| \leq w + 1$. For any fact $auxsat(v, i_1, i_2, P, N, C, j)$ we have $C \subseteq Cl(v)$. Hence the values of C can be considered as bit vectors of length at most $w +$

1. Therefore there are at most 2^{w+1} different values for C. Furthermore for any fact $auxunsat(v, i_1, i_2, P, N, P', N', C')$ we have $|P'|+|N'| = |Var(v)|$ and $C' \subseteq Cl(v)$. Hence the values of (P', N', C') can also be considered as bit vectors of length at most $w + 1$. This means for given v, P, N, C, there are at most $2^{2^{w+1}}$ different subsets of facts of the form $auxunsat(v, i_1, i_2, P, N, P', N', C')$. Putting it all together, we get at most $K = 2^{w+1} * 2^{2^{w+1}}$ non-equal pairs (i_1, i_2). Thus the index i in the predicate $reduce(v, P, N, i, i_1, i_2, j)$ is in $\{0, \ldots, K-1\}$.

The purpose of the predicate $diff(v, P, N, i_1, i_2, i'_1, i'_2)$ is to check whether for given values of v, P, N, the pairs (i_1, i_2) and (i'_1, i'_2) differ in their corresponding $auxunsat$ facts. Recall that it was one of the two requirements for equal pairs, that they do not differ in these facts. Therefore we can use $diff$ to define the predicate $equal$, where we additionally assure the first requirement for equal pairs: they have to satisfy the same clause set C, i.e. there have to be two facts of the form $auxsat(v, i_1, i_2, P, N, C, j)$ and $auxsat(v, i'_1, i'_2, P, N, C, j')$ in the lfp of the program.

Since we are going to group all equal index pairs, we have to update our counter j. This is done by the predicate $count\_reduced$, which computes the sum over the counter of the corresponding $auxsat$ facts of all equal index pairs. The intended meaning of the atom $less(i_1, i_2, i'_1, i'_2)$ is that the pair (i_1, i_2) is lexicographically smaller than the pair (i'_1, i'_2). The fact $reduced\_smaller(v, P, N, i, i_1, i_2)$ denotes that there exists a pair of indices which is smaller than (i_1, i_2) and which is already mapped onto the index i, i.e. we have to prevent a mapping of (i_1, i_2) onto i.

The intended meaning of the atom $duplicate(v, P, N, i_1, i_2)$ is to indicate that there exists a smaller pair of indices which is equal to (i_1, i_2). By allowing the derivation of a $reduce$ fact only for the smallest of all equal pairs of indices, we can assure that there exists only one such fact for each equivalence class. Finally the intended purpose of the fact $open(v, P, N, i, i_1, i_2)$ is to assure that we choose the smallest free index i for our mapping, i.e. there exists no smaller index i' which we have not yet used in such a mapping.

C Correctness Proof of the #CIRCUMSCRIPTION Program

We are going to proof that the predicates sat and $unsat$ indeed have the intended meaning, i.e. Property B holds, by structural induction on the tree decomposition:

base case. Let v be a leaf node. Since (P, N) is a partition of $Var(v) = Var(\mathcal{T}_v)$, the only truth value assignment extending (P, N) to $Var(\mathcal{T}_v)$ is (P, N) itself. Therefore for each set $S = S(v, P, N, C_u)$, the cardinality of S is either $|S| = 0$ or $|S| = 1$. This implies that if $S \neq \emptyset$, then there is only one possible partition $\{S_0\}$ of S with $S_0 = S$. Clearly $|S| = 1$ iff the predicate $true(P, N, C_u, C)$ with $C = Cl(v)$ is in the lfp iff $sat(v, 0, P, N, C_u, 1)$ is contained in the lfp.

The implication that $unsat(v, i, P, N, \_, \_, \_)$ is in the lfp only if also an atom $sat(v, i, P, N, \_, \_)$ is contained in the lfp is trivially fulfilled, since the latter fact is part of the body of the rule for deriving the first one.

Furthermore note that a fact $unsat(v, i, P, N, P', N', C'_u)$ is contained in the lfp $\Leftrightarrow$ the three facts $sat(v, i, P, N, C_u, \_)$, $sat(v, \_, P', N', C'_u, \_)$ and $P' \subset P$ are in the lfp of the program $\Leftrightarrow$ there exists a $J \in S(v, P, N, C_u)$ and a $J' \in S(v, P', N', C'_u)$ with $P' \subset P$.

induction step. We distinguish between the five different types of inner nodes of $\mathcal{T}$ assuming that Property B holds for all child nodes.

Let v be a *variable removal node* with removal of variable x and let v_1 denote the child of v. First we show that there exists an atom $sat(v, \_, P, N, C_u, \_)$ in the lfp of the program iff $S(v, P, N, C_u) \neq \emptyset$. Assuming that such a fact is in the lfp, a fact of the form $auxsat(v, \_, \_, P, N, C_u, \_)$ is also in the lfp. Therefore at least one of the facts $sat(v_1, \_, P \uplus \{x\}, N, C_u, \_)$ or $sat(v_1, \_, P, N \uplus \{x\}, C_u, \_)$ is in the lfp. We only treat the first case here, since the second one is symmetrical. By the induction hypothesis, $S(v_1, P \uplus \{x\}, N, C_u) \neq \emptyset$. Note that by the construction of S it holds that $S(v_1, P \uplus \{x\}, N, C_u) \subseteq S(v, P, N, C_u)$ and therefore $S(v, P, N, C_u) \neq \emptyset$. Now assume that $S(v, P, N, C_u) \neq \emptyset$. By the construction of S, at least one of the following two statements hold: $S(v_1, P \uplus \{x\}, N, C_u) \neq \emptyset$ or $S(v_1, P, N \uplus \{x\}, C_u) \neq \emptyset$. Again the two cases are symmetrical and we therfore only treat the first one. By the induction hypothesis, a fact $sat(v_1, \_, P \uplus \{x\}, N, C_u, \_)$ is in the lfp of the program. Therefore also a fact $auxsat(v, \_, \_, P, N, C_u, \_)$ and furthermore $sat(v, \_, P, N, C_u, \_)$ is in the lfp.

Next we show that if there exists an atom $unsat(v, i, P, N, \_, \_, \_, \_)$ in the lfp of the program, then there is also an atom $sat(v, i, P, N, \_, \_)$ in the lfp. Assuming that such an $unsat$ fact exists in the lfp, a predicate $reduce(v, P, N, i, i_1, i_2, \_)$ is also in the lfp. By the definition of $reduce$, there is also an atom $auxsat(v, i_1, i_2, P, N, \_, \_)$ in the lfp. This atom and the previous reduce predicate imply that $sat(v, i, P, N, \_, \_)$ exists in the lfp.

Finally we have to show that if $S(v, P, N, C) \neq \emptyset$ then there exists a partition of this set fulfilling the conditions 1 – 3 of Property B. Therefore assume that $S = S(v, P, N, C) \neq \emptyset$ and let $S' = S(v_1, P \uplus \{x\}, N, C_u)$ and $S'' = S(v_1, P, N \uplus \{x\}, C_u)$. Note that by the construction of S it holds that $S = S' \cup S''$ and $S' \cap S'' = \emptyset$. Therefore at least one of the two sets S' and S'' is not empty. Furthermore by the induction hypothesis, at least one of the following two partitions exists and fulfills the conditions 1 – 3 of Property B: $S' = \{S_{k_1}, \ldots, S_{k_n}\}$ and $S'' = \{S_{l_1}, \ldots, S_{l_m}\}$. Therefore we can define a partition of S as $\{S_{k_1}, \ldots, S_{k_n}, S_{l_1}, \ldots, S_{l_m}\}$.

Now we show that a fact $auxsat(v, i_1, i_2, P, N, C_u, \_)$ is contained in the lfp iff either $(i_1, i_2) \in \{(k_1, 0), \ldots, (k_n, 0)\}$ or $(i_1, i_2) \in \{(l_1, 1), \ldots, (l_m, 1)\}$. Assume that $auxsat(v, i_1, i_2, P, N, C_u, \_)$ is in the lfp. We have to distinguish two cases depending on whether $i_2 = 0$ or $i_2 = 1$. We only treat the first case here, since the second one is symmetrical. In this case a fact $sat(v_1, i_1, P \uplus \{x\}, N, C_u, \_)$ has to be in the lfp of the program. Therefore by the induction hypothesis, $i_1 \in \{k_1, \ldots, k_n\}$ and clearly $(i_1, i_2) \in \{(k_1, 0), \ldots, (k_n, 0)\}$. For the other direction, assume that $(i_1, i_2) \in \{(k_1, 0), \ldots, (k_n, 0)\}$ or $(i_1, i_2) \in \{(l_1, 1), \ldots, (l_m, 1)\}$. Again the two cases are similar, hence we only treat the first one here. In this case, by the construction of the partition and the induction hypothesis, a fact $sat(v_1, i_1, P \uplus \{x\}, N, C_u, \_)$ is in the lfp. But then also a fact $auxsat(v, i_1, 0, P, N, C_u, \_)$ is in the lfp.

Next we show that the fact $auxsat(v, i_1, i_2, P, N, C_u, j)$ is contained in the lfp iff $|S_{i_1}| = j$. Assume that $auxsat(v, i_1, i_2, P, N, C_u, j)$ indeed exists in the lfp. Again we only consider the case where $i_2 = 0$. But then the fact $sat(v_1, i_1, P \uplus \{x\}, N, C_u, j)$ is also in the lfp and by the induction hypothesis it holds that $|S_{i_1}| = j$. For the other direction assume that $|S_{i_1}| = j$. Depending on the index i_1 there are again two cases to consider. We treat only the one where $i_1 \in \{k_1, \ldots, k_n\}$. By the induction hypoth-

esis, the atom $sat(v_1, i_1, P \uplus \{x\}, N, C_u, j)$ is in the lfp and therefore also the fact $auxsat(v, i_1, 0, P, N, C_u, j)$.

Finally we show that condition 3 of Property B holds when we replace the predicate $unsat$ with $auxunsat$, i.e. we show that for every partition (P', N') of $Var(v)$ and every subset $C'_u \subseteq Cl(v)$, the following three statements are equivalent:

(i) The fact $auxunsat(v, i_1, i_2, P, N, P', N', C'_u)$ is contained in the lfp.
(ii) There exists a $J \in S_{i_1}$ and an assignment $J' \subset J$, s.t. $J' \in S(v, P', N', C'_u)$.
(iii) For all $J \in S_{i_1}$ there exists an assignment $J' \subset J$, s.t. $J' \in S(v, P', N', C'_u)$.

To facilitate the proof of these equivalences, let $S' = S(v, P', N', C'_u)$ and $S'_1 = S(v_1, P' \uplus \{x\}, N', C'_u)$. Note that by the construction of these sets, $S'_1 \subseteq S'$.

First we show that (i) $\Rightarrow$ (ii). Assume that $auxunsat(v, i_1, i_2, P, N, P', N', C'_u)$ is in the lfp. Again we only consider the case where $i_2 = 0$. Then there is either the fact $unsat(v_1, i_1, P \uplus \{x\}, N, P' \uplus \{x\}, N', C'_u)$ or the fact $unsat(v_1, i_1, P \uplus \{x\}, N, P', N' \uplus \{x\}, C'_u)$ in the lfp. These two cases are also symmetrical, hence we only consider the first one. By the induction hypothesis, there exists a $J \in S_{i_1}$ and an assignment $J' \subset J$, s.t. $J' \in S'_1$. Since $S'_1 \subseteq S'$, $J' \in S'$ and therefore (i) $\Rightarrow$ (ii) holds.

Next we show that (ii) $\Rightarrow$ (iii). Assume there exists a $J \in S_{i_1}$ and an assignment $J' \subset J$, s.t. $J' \in S'$. There are four possible cases depending on the truth value of $J(x)$ and $J'(x)$. Since they are all symmetrical, we only treat the case where both $J(x) =$ true and $J'(x) =$ true. Due to the construction of our partition of S, we know that $i_1 \in \{k_1, \ldots, k_n\}$. Furthermore it holds that $J' \in S'_1$. Therefore we can apply the induction hypothesis and hence for all $J \in S_{i_1}$ there exists an assignment $J' \subset J$, s.t. $J' \in S'_1$. Since $S'_1 \subseteq S'$, $J' \in S'$ and therefore (ii) $\Rightarrow$ (iii).

Now we show that (iii) $\Rightarrow$ (i). Assume that for all $J \in S_{i_1}$ there exists an assignment $J' \subset J$, s.t. $J' \in S'$. Note that for each J, the truth value of $J'(x)$ is the same, depending only on the partition (P', N'). Therefore there are also four possible cases distinguishable by the truth value of each $J(x)$ and each $J'(x)$. We only treat the case where both $J(x) =$ true and $J'(x) =$ true. Due to the construction of our partition of S, we know that $i_1 \in \{k_1, \ldots, k_n\}$. Since $J'(x) =$ true and by the definition of S, it holds for each J' that $J' \in S'_1$. Hence by the induction hypothesis, the fact $unsat(v, i_1, P \uplus \{x\}, N, P' \uplus \{x\}, N', C'_u)$ is contained in the lfp. Therefore also the predicate $auxunsat(v, i_1, 0, P, N, P', N', C'_u)$ is in the lfp.

What we have done so far, was basically proving the conditions 1 – 3 of Property B for the predicates $auxsat$ and $auxunsat$. Note that by the definition of the predicate $reduce$, the partition $\{S_{k_1}, \ldots, S_{k_n}, S_{l_1}, \ldots, S_{l_m}\}$ of S is transformed into the smallest possible new partition $\{S_1, \ldots, S_{n'}\}$ still satisfying the conditions 1 – 3 of Property B by combining sets $S_{i_1}, S_{i_2}, \ldots$ to a new set S_i with $i \in \{1, \ldots, n'\}$. Therefore Property B also holds for the predicates sat and $unsat$.

The proof of the induction step for the other node types works analogously and is therefore omitted. □

D Auxiliary Predicates for the #HORN-ABDUCTION Program

A datalog implementation of the predicate $reduce$ for the #HORN-ABDUCTION program plus several auxiliary predicates is given in Figure 7. The idea of the predicate $reduce(v, S, i, i_1, i_2, j)$ is basically the same as for the corresponding predicate of the

#CIRCUMSCRIPTION program. We want to detect "equal" pairs of indices (i_1, i_2) and map them to a new index i. For given values of v, S, two pairs (i_1, i_2) and (i'_1, i'_2) are equal iff it holds that a fact $aux(v, S, i_1, i_2, C^o, RC, \Delta C, RO, j)$ is in the lfp iff $aux(v, S, i'_1, i'_2, C^o, RC, \Delta C, RO, j')$ is in the lfp.

Program Auxiliary Predicates for #HORN-ABDUCTION

$diff(v, S, i_1, i_2, i'_1, i'_2) \leftarrow aux(v, S, i_1, i_2, C^o, RC, \Delta C, RO, \_)$,
not $aux(v, S, i'_1, i'_2, C^o, RC, \Delta C, RO, \_)$, $0 \leq i'_1, i'_2 < K$.

$diff(v, S, i_1, i_2, i'_1, i'_2) \leftarrow diff(v, S, i'_1, i'_2, i_1, i_2)$.

$equal(v, S, i_1, i_2, i'_1, i'_2) \leftarrow aux(v, S, i_1, i_2, C^o, RC, \Delta C, RO, \_)$,
$aux(v, S, i'_1, i'_2, C^o, RC, \Delta C, RO, \_)$, not $diff(v, S, i_1, i_2, i'_1, i'_2)$.

$count\_reduced(v, S, i_1, i_2, \text{SUM}(j)) \leftarrow equal(v, S, i_1, i_2, i'_1, i'_2)$,
$aux(v, S, i'_1, i'_2, C^o_1, \_, \_, \_, j)$, not $aux(v, S, i'_1, i'_2, C^o_2, \_, \_, \_, \_)$. $C^o_2 < C^o_1$,

$less(i_1, i_2, i'_1, i'_2) \leftarrow i_1 < i'_1, 0 \leq i_1, i_2, i'_1, i'_2 < K$.
$less(i, i_2, i, i'_2) \leftarrow i_2 < i'_2, 0 \leq i_1, i_2, i'_1, i'_2 < K$.

$reduced\_smaller(v, S, i, i_1, i_2) \leftarrow less(i'_1, i'_2, i_1, i_2), reduce(v, S, i, i'_1, i'_2, j)$.

$duplicate(v, S, i_1, i_2) \leftarrow less(i'_1, i'_2, i_1, i_2), equal(v, S, i_1, i_2, i'_1, i'_2)$.

$open(v, S, i, i_1, i_2) \leftarrow i' < i$, not $reduced\_smaller(v, S, i', i_1, i_2), 0 \leq i, i' < K$.

$reduce(v, S, i, i_1, i_2, j) \leftarrow 0 \leq i < K, count\_reduced(v, S, i_1, i_2, j)$,
not $reduced\_smaller(v, S, i, i_1, i_2)$, not $duplicate(v, S, i_1, i_2)$,
not $open(v, S, i, i_1, i_2)$.

Fig. 7. *reduce* plus auxiliary predicates for #HORN-ABDUCTION.

The intended meaning of the auxiliary predicates is the same as for the #CIRCUMSCRIPTION program except for the fact $count\_reduced$. Since it is possible that for given values of v, S, i_1, i_2 the lfp contains multiple atoms $aux(v, S, i_1, i_2, C^o, \dots)$ with different C^o, we have to ensure that we count only one of them. Therefore the summation is done over those facts with the smallest C^o according to some lexicographical ordering.