Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (2008)

Abduction with Bounded Treewidth: From Theoretical Tractability to Practically Efficient Computation

Georg Gottlob
Computing Laboratory
Oxford University
Oxford OX1 3QD,UK

Reinhard Pichler
Information Systems Institute
Vienna University of Technology
A-1040 Vienna, Austria

Fang Wei *
Department of Computer Science
University of Freiburg
D-79110 Freiburg, Germany

Abstract

Abductive diagnosis is an important method to identify explanations for a given set of observations. Unfortunately, most of the algorithmic problems in this area are intractable. We have recently shown (Gottlob, Pichler, and Wei 2006) that these problems become tractable if the underlying clausal theory has bounded treewidth. However, turning these theoretical tractability results into practically efficient algorithms turned out to be very problematical. In (Gottlob, Pichler, and Wei 2007), we have established a new method based on monadic datalog which remedies this unsatisfactory situation. Specifically, we designed an efficient algorithm for a strongly related problem in the database area. In the current paper, we show that these favorable results can be carried over to logic-based abduction.

Introduction

Abductive diagnosis aims at an explanation of some observed symptoms in terms of (minimal) sets of hypotheses (like failing components) which may have led to these symptoms (de Kleer, Mackworth, and Reiter 1992). Unfortunately, most of the algorithmic problems in logic-based abduction are intractable (Eiter and Gottlob 1995). For instance, both the Solvability problem (i.e., does there exist an explanation?) and the Relevance problem (i.e., is a given hypothesis part of some explanation?) are Σ_2^p-complete.

A very promising approach to deal with intractability comes from the area of parameterized complexity (Downey and Fellows 1999). In particular, it has been shown that many hard problems become tractable if some problem parameter is fixed or bounded by a constant. In the arena of graphs and, more generally, of finite structures, the treewidth is one such parameter which has served as the key to many fixed-parameter tractability (FPT) results. The most prominent method for establishing the FPT in case of bounded treewidth is via Courcelle's Theorem (Courcelle 1990): 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 treewidth of the structures is bounded by a fixed constant. As far as logic-based abduction is concerned, the FPT of the most relevant algorithmic problems was indeed shown by applying Courcelle's Theorem, see (Gottlob, Pichler, and Wei 2006).

Clearly, an MSO description as such is not an algorithm, but recipes to devise concrete algorithms based on Courcelle's Theorem can be found in the literature, see e.g. (Flum, Frick, and Grohe 2002). The basic idea of these algorithms is to transform the MSO evaluation problem into an equivalent tree language recognition problem and to solve the latter via an appropriate finite tree automaton (FTA). In theory, this generic method of turning an MSO description into a concrete algorithm looks very appealing. However, in practice, it has turned out that even relatively simple MSO formulae may lead to a "state explosion" of the FTA (Maryns 2006). Hence, it was already stated in (Grohe 1999) that the algorithms derived via Courcelle's Theorem are "useless for practical applications". The main benefit of Courcelle's Theorem is that it provides "a simple way to recognize a property as being linear time computable". In other words, proving the FPT of some problem by showing that it is MSO expressible is the starting point (rather than the end point) of the search for an efficient algorithm.

Indeed, we have made two unsuccessful attempts to tackle abduction with bounded treewidth via the standard MSO-to-FTA approach: In (Gottlob, Pichler, and Wei 2006) we experimented on a related logic programming problem with a prototype implementation using MONA for the MSO model checking (Klarlund, Møller, and Schwartzbach 2002). We have meanwhile extended these experiments to abduction (see section on "Implementation and Results"). But we ended up with "out-of-memory" errors already for really small input data. Alternatively, we also tried to implement the MSO-to-FTA mapping proposed in (Flum, Frick, and Grohe 2002). However, this attempt failed with a "state explosion" of the resulting FTA yet before we were able to feed any input data to the program.

In (Gottlob, Pichler, and Wei 2007) we proposed monadic datalog (i.e., datalog where all intensional predicate symbols are unary) as a practical tool for devising efficient algorithms in situations where the FPT has been established via Courcelle's Theorem. Above all, we proved that if some property of finite structures is expressible in MSO then this property can also be expressed by means of a monadic datalog pro-

*Work performed while the author was with Vienna University of Technology.

Copyright © 2008, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

gram over the structure plus the tree decomposition. We put this approach to work by designing an efficient algorithm for the PRIMALITY problem (i.e., the problem of deciding if some attribute is part of a key in a given relational schema). In the current paper, we show that these favorable results can be carried over to logic-based abduction.

Treewidth, MSO, and Monadic Datalog

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 denotes 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 some node $t \in T$ with $\{a_1, \ldots, a_\alpha\} \subseteq A_t$. (3) For every $a \in A$, the set $\{t \mid a \in A_t\}$ 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 $tw(\mathcal{A})$. Note that trees and forests are precisely the structures with treewidth 1.

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 (Bodlaender 1996). W.l.o.g., we may assume that all nodes in the resulting tree decomposition have at most 2 child nodes. In fact, a much more restrictive *normalized form* of tree decompositions will be given below when we discuss our new abduction algorithms.

MSO extends First Order logic (FO) by the use of *set variables* (usually denoted by upper case letters), which range over sets of domain elements. In contrast, the *individual variables* (usually 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 of applying the immediate consequence operator. Predicates occurring only in the body of rules in $\mathcal{P}$ are called extensional, while predicates occurring also in the head of some rule are called intensional.

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

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

where the unary predicates $root$, and $leaf$ as well as the binary predicates $child_1$ and $child_2$ are used to represent the tree T of the tree decomposition in the obvious way. For instance, we write $child_1(s_1, s)$ to denote that s_1 is either the first child or the only child of s. Finally, bag has arity $k + 2$ with $k \leq w$, where $bag(t, a_0, \ldots, a_k)$ means that the bag at node t is $(a_0, \ldots, a_k)$. By slight abuse of notation we tacitly assume that bag is overloaded for various values of k. Note that the possible values of k are bounded by a fixed constant w. For any τ-structure $\mathcal{A}$ with tree decomposition $\mathcal{T} = \langle T, (A_t)_{t \in T} \rangle$ of width w, we denote by $\mathcal{A}_{td}$ the τ_{td}-structure representing $\mathcal{A}$ plus $\mathcal{T}$ in the following way: The domain of $\mathcal{A}_{td}$ is the union of $dom(\mathcal{A})$ and the nodes of T. In addition to the relations $R_i^{\mathcal{A}}$ with $R_i \in \tau$, the structure $\mathcal{A}_{td}$ also contains relations for each predicate $root$, $leaf$, $child_1$, $child_2$, and bag thus representing the tree decomposition $\mathcal{T}$. By (Bodlaender 1996), one can compute $\mathcal{A}_{td}$ from $\mathcal{A}$ in linear time w.r.t. the size of $\mathcal{A}$.

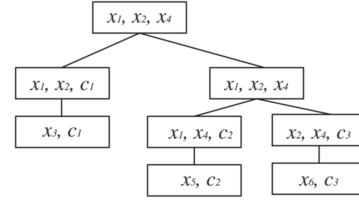

Figure 1: Tree decomposition $\mathcal{T}$ of formula φ.

Example 1 *We can represent propositional formulae φ 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) in φ and $pos(x, c)$ (resp. $neg(x, c)$) means that x occurs unnegated (resp. negated) in the clause c, e.g.: 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 $\mathcal{A} = \{var(x_1), var(x_2), var(x_3), var(x_4), var(x_5), var(x_6), cl(c_1), cl(c_2), cl(c_3), pos(x_1, c_1), pos(x_3, c_1), pos(x_4, c_2), pos(x_2, c_3), pos(x_6, c_3), neg(x_2, c_1), neg(x_1, c_2), neg(x_5, c_2), neg(x_4, c_3)\}$.

A tree decomposition $\mathcal{T}$ of this structure is given in Figure 1. Note that the maximal size of the bags in $\mathcal{T}$ is 3. Hence, the tree-width is ≤ 2. On the other hand, it is easy to check that the tree-width of $\mathcal{T}$ cannot be smaller than 2. This tree decomposition is, therefore, optimal and we have $tw(\varphi) = tw(\mathcal{A}) = 2$.

In order to represent $\mathcal{A}_{td}$, the following ground atoms have to be added to $\mathcal{A}$: $\{root(t_1), child_1(t_2, t_1), child_2(t_3, t_1), child_1(t_4, t_2), \ldots, leaf(t_4), leaf(t_7), leaf(t_8), bag(t_1, x_1, x_2, x_4), bag(t_2, x_1, x_2, c_1), bag(t_3, x_1, x_2, x_4), bag(t_4, x_3, c_1), \ldots\}$. (The numbering of the nodes $t_1, t_2, t_3, \ldots$ corresponds to a breadth-first traversal of $\mathcal{T}$.)

In (Gottlob, Pichler, and Wei 2007), the following connection between unary MSO queries over structures with bounded treewidth and monadic datalog was established:

Theorem 2 *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.*

New Abduction Algorithms

A *propositional abduction problem* (PAP) $\mathcal{P}$ consists of a tuple $\langle V, H, M, \mathcal{C} \rangle$, 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 clause

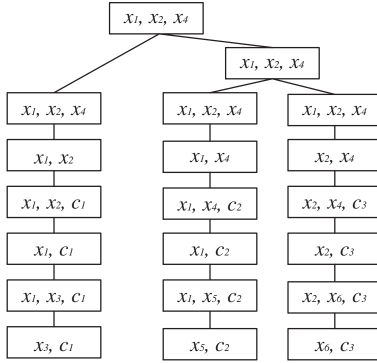

Figure 2: Normalized tree decomposition $\mathcal{T}'$ of formula φ.

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. A hypothesis $h \in H$ is called *relevant* if h is contained in at least one solution $\mathcal{S}$ of $\mathcal{P}$.

In an abductive diagnosis problem, the manifestations M are the observed symptoms (e.g. describing some erroneous behavior of the system) and the clausal theory $\mathcal{C}$ constitutes the system description. The solutions $\mathcal{S} \subseteq H$ are the possible explanations for the observed symptoms.

In this paper, we assume that a PAP $\mathcal{P}$ is given as a τ-structure with $\tau = \{cl, var, pos, neg, hyp, man\}$. The intended meaning of the predicates cl, var, pos, neg is as in Example 1, i.e., $cl(z)$ (resp. $var(z)$) means that z is a clause (resp. a variable) in $\mathcal{C}$ and $pos(x, c)$ (resp. $neg(x, c)$) means that x occurs unnegated (resp. negated) in the clause c. Moreover, the unary predicates hyp and man identify the hypotheses and manifestations.

In (Gottlob, Pichler, and Wei 2007), 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 $a_0, \ldots, a_w$, where a_i denotes a variable or a clause in the structure representing the PAP $\mathcal{P}$. (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 (i.e., variable or clause) or by introducing a new element. Consequently, we call t a *variable removal node*, a *clause removal node*, a *variable introduction node*, or a *clause introduction node*, respectively. (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*.

Example 3 *Recall the tree decomposition $\mathcal{T}$ from Figure 1. Clearly, $\mathcal{T}$ is not normalized in the above sense. However, in can be easily transformed into a normalized tree decomposition $\mathcal{T}'$, see Figure 2.*

In this section, we discuss new algorithms for the Solvability problem (i.e., does there exist a solution) and for the Relevance enumeration problem (i.e., enumerate all relevant hypotheses of a given PAP). Since these problems heavily depend on the SAT-problem, we start our exposition with an FPT algorithm for the SAT-problem. Note that the tractability of the SAT-problem with bounded treewidth was already shown in (Szeider 2004).

The SAT Problem

Suppose that a clause set together with a tree decomposition $\mathcal{T}$ of width w is given as a τ_{td}-structure with $\tau_{td} = \{cl, var, pos, neg, root, leaf, child_1, child_2, bag\}$. Of course, we do not need the predicates hyp and man for the SAT problem. In Figure 3, we describe a datalog program which decides the SAT-problem.

Program SAT

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

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

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

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

Figure 3: SAT-Test.

Some words on the notation used in this program are in order: We are using lower case letters v, c, and x (possibly with subscripts) as datalog variables for a single node in $\mathcal{T}$, for a single clause, or for a single propositional variable, respectively. In contrast, upper case letters are used as datalog variables denoting sets of variables (in the case of X, P, N) or sets of clauses (in the case of C). Note that the sets are not sets in the general sense, since their cardinality is restricted by the maximal size $w + 1$ of the bags, where w is a fixed constant. Indeed, we have implemented these "fixed-size" sets by means of k-tuples with $k \leq (w + 1)$ over $\{0, 1\}$. For the sake of readability, we are using non-datalog expressions involving the $\cup$-operator, which one could easily replace by "proper" datalog expressions. For instance, $P \cup N = X$ can of course be replaced by union(P, N, X).

In order to facilitate the discussion, we introduce the following notation. Let $\mathcal{C}$ denote the input clause set with variables in V and tree decomposition $\mathcal{T}$. For any node v in $\mathcal{T}$,

we write $\mathcal{T}_v$ to denote the subtree of $\mathcal{T}$ rooted at v. By $Cl(v)$ we denote the 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 only three intensional predicates $solve$, $true$, and $success$. The crucial predicate is $solve(v, P, N, C)$ with the following intended meaning: v denotes a node in $\mathcal{T}$. P and N are disjoint subsets of $Var(v)$ representing a truth value assignment on $Var(v)$, s.t. all variables in P are true and all variables in N are false. C denotes a subset of $Cl(v)$. For all values of v, P, N, C, the ground fact $solve(v, P, N, C)$ shall be in the fixpoint of the program, iff the following condition holds:

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

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

Theorem 4 *The datalog program in Figure 3 decides the SAT problem, i.e., the fact "success" is in the fixpoint of this program iff the input τ_{td}-structure encodes a satisfiable clause set $\mathcal{C}$. Moreover, for any clause set $\mathcal{C}$ with treewidth w, the computation of the τ_{td}-structure and the evaluation of the datalog program can be done in time $\mathcal{O}(f(w) * |\mathcal{C}|)$ for some function f.*

Proof. Suppose that the predicate $solve$ indeed has the meaning described above. Then the rule with head $success$ reads as follows: $success$ is in the fixpoint, iff v denotes the root of $\mathcal{T}$ and there exists an assignment (P, N) on the variables in $Var(v)$, s.t. for some extension J of (P, N) to $Var(\mathcal{T}_v)$, all clauses in $Cl(\mathcal{T}_v) = (Cl(\mathcal{T}_v) \setminus Cl(v)) \cup C$ are true in J. But this simply means that J is a satisfying assignment of $\mathcal{C} = Cl(\mathcal{T}_v)$. The correctness of the $solve$ predicate can be proved by structural induction on $\mathcal{T}$.

For the linear time data complexity, the crucial observation is that our program in Figure 3 is essentially a succinct representation of a monadic datalog program. For instance, in the atom $solve(v, P, N, C)$, 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 3 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, $solve(v, P, N, C)$ is simply a succinct representation of constantly many monadic predicates of the form $solve_{r,s,t}(v)$. Thus, the linear time bound is implicit in Theorem 2. □

The Solvability Problem

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

The program in Figure 4 realizes this SAT/UNSAT-intuition. For the discussion of this program, it is convenient to introduce the following additional notation. We shall write $H(v)$, $M(v)$, $H(\mathcal{T}_v)$ and $M(\mathcal{T}_v)$ to denote the restriction of H and M to the variables in the bag of v or in any bag in $\mathcal{T}_v$, respectively. Of course, the unary predicates hyp and man are now contained in τ_{td}.

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

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

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

Program Solvability

/* **leaf node.** */

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

/* **internal node.** */

/* variable removal node. */

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

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

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

/* clause removal node. */

$solve(v, S, i, P, N, C_1, d) \leftarrow bag(v, X, C)$, $child_1(v_1, v)$,
$bag(v_1, X, C \cup \{c\})$, $solve(v_1, S, i, P, N, C_1 \cup \{c\}, d)$.

/* variable introduction node. */

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

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

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

/* clause introduction node. */

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

/* branch node. */

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

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

$solve(v, S, i, P, N, C, d) \leftarrow$
$aux(v, S, i_1, i_2, P, N, C, d)$, $reduce(v, S, i, i_1, i_2)$.

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

$success \leftarrow root(v)$, $bag(v, X, C)$,
$solve(v, S, i, P, N, C,$ 'n'$)$, not $solve'(v, S, i)$.

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

Figure 4: Solvability.

The predicate $svar$ in Figure 4 is used to select sets of hypotheses, i.e., $svar(v, S)$ is true for every subset $S \subseteq H(v)$. The predicate $check$ extends the predicate $true$ from the SAT program by additionally setting the d-Bit, i.e., $check(P, N, C_1, C, d)$ iff $true(P, N, C_1, C)$. Moreover, d = 'y' iff N contains some manifestation.

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

Analogously to Theorem 4, the following properties of the Solvability program can be shown.

Theorem 5 *The datalog program in Figure 4 decides the Solvability problem of PAPs, i.e., the fact "success" is in the fixpoint of this program iff the input τ_{td}-structure encodes a solvable PAP $\mathcal{P}$. Moreover, for any PAP $\mathcal{P} = \langle V, H, M, T \rangle$ with treewidth w, the computation of the τ_{td}-structure and the evaluation of the datalog program can be done in time $\mathcal{O}(f(w) * |\mathcal{C}|)$ for some function f.*

The Relevance Problem

The problem of computing all relevant hypotheses can be clearly expressed as a unary MSO-query and, thus, by a monadic datalog program. Indeed, it is straightforward to extend our Solvability program to a program for the Relevance enumeration problem: Suppose that some hypothesis h occurs in the bag of the root r of $\mathcal{T}$. Then h is relevant iff there exists a subset $S \subseteq H(r)$ and an index i, s.t. $h \in S$ and S can be extended to a solution $\bar{S}_i \subseteq H(\mathcal{T}_r)$ of the PAP $\mathcal{P}$. Naively, one can compute all relevant hypotheses by considering the tree decomposition $\mathcal{T}$ as rooted at various nodes, s.t. each $h \in H$ is contained in the bag of at least one such root node. Obviously, this method has *quadratic* time complexity w.r.t. the data size.

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

tw	#H	#M	#V	#Cl	#tn	MD	MONA
3	1	1	3	1	3	0.3	870
3	2	2	6	2	12	0.5	1710
3	3	3	9	3	21	0.6	12160
3	4	4	12	4	34	0.9	–
3	7	7	21	7	69	1.5	–
3	11	11	33	11	105	2.3	–
3	15	15	45	15	141	2.9	–
3	19	19	57	19	193	3.9	–
3	23	23	69	23	229	4.7	–
3	27	27	81	27	265	5.3	–
3	31	31	93	31	301	6.1	–

Table 1: Processing Time in ms for Solvability Problem

Then the relevant hypotheses can be obtained via the $solve{\downarrow}(v, \ldots)$ facts in the fixpoint of the program for all leaf nodes v of $\mathcal{T}$ (since these facts correspond precisely to the $solve(v, \ldots)$ facts if $\mathcal{T}$ were rooted at v). The resulting algorithm works in linear time since it essentially just doubles the computational effort of the Solvability program.

Implementation and Results

To test the performance and, in particular, the scalability of our approach, we have implemented our Solvability program in C++. The experiments were conducted on Linux kernel 2.6.17 with a 1.60 GHz Intel Pentium(M) processor and 512 MB of memory. We measured the processing time on different input parameters such as the number of variables, clauses, hypotheses, and manifestations. The treewidth in all the test cases was 3. Due to the lack of available test data, we generated a balanced normalized tree decomposition and test data sets with increasing input parameters by expanding the tree in a depth-first style. We have ensured that all different kinds of nodes occur evenly in the tree decomposition.

The outcome of the tests is shown in Table 1, where tw stands for the treewidth; #H, #M, #V, #Cl and #tn stand for the number of hypotheses, manifestations, all variables, clauses and tree nodes, respectively. The processing time (in ms) obtained with our implementation of the monadic datalog approach are displayed in the column labelled “MD”. The measurements nicely reflect an essentially linear increase of the processing time with the size of the input. Moreover, there is obviously no “hidden” constant which would render the linearity useless.

In (Gottlob, Pichler, and Wei 2006), we proved the FPT of several non-monotonic reasoning problems via Courcelle’s Theorem. Moreover, we also carried out some experiments with a prototype implementation using MONA (Klarlund, Møller, and Schwartzbach 2002) for the MSO-model checking. We have now extended these experiments with MONA to the Solvability problem of abduction. The time measurements of these experiments are shown in the last column of Table 1. Due to problems discussed in (Gottlob, Pichler, and Wei 2006), MONA does not ensure linear data complexity. Hence, all testes below line 3 of the table failed with “out-of-memory” errors. Moreover, also in cases where the exponential data complexity does not yet “hurt”, our datalog approach outperforms the MSO-to-FTA approach by a factor of 100 or even more.

Conclusions and Future Work

In (Gottlob, Pichler, and Wei 2007) we presented a new method for turning theoretical tractability results (obtained via Courcelle’s Theorem) into practically efficient computations. In the current paper, we have shown how this result can be fruitfully applied to logic-based abduction.

The datalog programs presented in this paper were obtained by an ad hoc construction rather than via a generic transformation from MSO. Nevertheless, we are convinced that the idea of a bottom-up propagation of certain SAT- and UNSAT-conditions is quite generally applicable. Actually, many more hard problems in the area of knowledge representation and reasoning (e.g., with various kinds of closed world assumptions) were shown to be expressible in MSO (Gottlob, Pichler, and Wei 2006). As a next step, we are therefore planning to devise new efficient algorithms also for these problems based on our monadic datalog approach.

References

Bodlaender, H. L. 1996. A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth. *SIAM J. Comput.* 25(6):1305–1317.

Courcelle, B. 1990. Graph Rewriting: An Algebraic and Logic Approach. In *Handbook of Theoretical Computer Science, Volume B*. Elsevier Science Publishers. 193–242.

de Kleer, J.; Mackworth, A. K.; and Reiter, R. 1992. Characterizing diagnoses and systems. *Artif. Intell.* 56(2-3):197–222.

Downey, R. G., and Fellows, M. R. 1999. *Parameterized Complexity*. New York: Springer.

Eiter, T., and Gottlob, G. 1995. The complexity of logic-based abduction. *J. ACM* 42(1):3–42.

Flum, J.; Frick, M.; and Grohe, M. 2002. Query evaluation via tree-decompositions. *J. ACM* 49(6):716–752.

Gottlob, G.; Pichler, R.; and Wei, F. 2006. Bounded treewidth as a key to tractability of knowledge representation and reasoning. In *Proc. AAAI’06*, 250–256.

Gottlob, G.; Pichler, R.; and Wei, F. 2007. Monadic Datalog over Finite Structures with Bounded Treewidth. In *Proc. PODS’07*, 165–174.

Grohe, M. 1999. Descriptive and Parameterized Complexity. In *Proc. CSL’99*, volume 1683 of *LNCS*, 14–31.

Klarlund, N.; Møller, A.; and Schwartzbach, M. I. 2002. MONA Implementation Secrets. *International Journal of Foundations of Computer Science* 13(4):571–586.

Maryns, H. 2006. On the Implementation of Tree Automata: Limitations of the Naive Approach. In *Proc. 5th Int. Treebanks and Linguistic Theories Conference (TLT 2006)*, 235–246.

Szeider, S. 2004. On Fixed-Parameter Tractable Parameterizations of SAT. In *Proc. 6th Int. Conf. SAT 2003, Selected Revised Papers*, volume 2919 of *LNCS*, 188–202. Springer.