Set Constraints with Projections

WITOLD CHARATONIK AND LESZEK PACHOLSKI

University of Wrocław, Wrocław, Poland

Abstract. Set constraints form a constraint system where variables range over the domain of sets of trees. They give a natural formalism for many problems in program analysis. Syntactically, set constraints are conjunctions of inclusions between expressions built over variables, constructors (constants and function symbols from a given signature) and a choice of set operators that defines the specific class of set constraints. In this article, we are interested in the class of *set constraints with projections*, which is the class with all Boolean operators (union, intersection and complement) and *projections* that in program analysis directly correspond to type destructors. We prove that the problem of existence of a solution of a system of set constraints with projections is in NEXPTIME, and thus that it is NEXPTIME-complete.

Categories and Subject Descriptors: F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*automata*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*computations on discrete structures*; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*Program analysis*; F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages—*Decision problems*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Computational complexity, program analysis, set constraints

ACM Reference Format:

Charatonik, W., and Pacholski, L. 2010. Set constraints with projections. J. ACM 57, 4, Article 23, (April 2010), 37 pages.
DOI = 10.1145/1734213.1734217 http://doi.acm.org/10.1145/1734213.1734217

1. *Introduction*

Set constraints form a constraint system where variables range over the domain of sets of trees. They give a natural formalism for many problems in program analysis, type inference, order-sorted unification, and constraint logic programming, but they are best known as a tool for a particular kind of static analysis called set-based [Heintze 1992; Heintze and Jaffar 1994; Aiken 1999]. Set-based analysis

This research was partially supported by Polish Ministry of Science and Education grant 3 T11C 042 30

Authors' address: W. Charatonik and L. Pacholski, Institute of Computer Science, University of Wrocław, Joliot-Curie 15, 50-383 Wrocław, Poland, e-mail: wch@ii.uni.wroc.pl; pacholski@cs.uni.wroc.pl.

DOI 10.1145/1734213.1734217 http://doi.acm.org/10.1145/1734213.1734217

is divided into two steps: the generation of set constraints from the syntax of a given program and the resolution of the constraints, that is, the computation of a particular solution (e.g., the least one). The computed solution gives then information about possible values computed by the program or assumed by variables, and can be further used, for example, in checking the type consistency or possibility of optimization.

Syntactically, set constraints are conjunctions of inclusions between expressions built over variables, constructors (constants and function symbols from a given signature) and a choice of set operators that defines the specific class of set constraints. In this article, we are interested in the satisfiability problem for *set constraints with projections*, that is, in the question whether given system of set constraints has a solution. The class of set constraints we consider is a very natural one: we have all boolean operators (union, intersection and complement) and *projections* that in program analysis directly correspond to type destructors.

1.1. HISTORY OF SET CONSTRAINTS. The history of set constraints and set-based program analysis goes back to Reynolds [1969]. He was the first to derive recursively defined sets as approximations of runtime values from first-order functional programs and to simplify these definitions by a sort of set-constraint solving algorithm. Essentially, his algorithm solves (in cubic time) the satisfiability question for the very limited case of atomic set constraints, where except constructors there are no set operators at all.

The first general results concerning the decidability of set constraints were obtained by Heintze and Jaffar [1990a], who studied the so-called definite set constraints. The name of the class comes from the fact that satisfiable constraints in this class always have the *least* solution. Definite set constraints are of the form $exp_1 \subseteq exp_2$ where exp_2 is restricted to constants, variables and function symbols, and exp_1 does not contain the complement symbol, but may contain projections. This class is used for the set-based analysis of logic programs [Heintze and Jaffar 1990b]. Later, Charatonik and Podelski [2002] showed that definite set constraints are equivalent to the class of *set constraints with intersection* where the only operators forming expressions are constructors and intersection, and that the satisfiability problem for these classes is DEXPTIME-complete. The complexity characterization continues to hold for *negative set constraints with intersection*, where also negated inclusions are allowed. In Charatonik and Podelski [1998] the same authors introduced the class of *co-definite set constraints*, and showed that its satisfiability problem is also DEXPTIME-complete. This is a natural subclass of set constraints which, when satisfiable, have the *greatest* solution. Both classes of definite and co-definite set constraints are motivated by the analysis of programs with the semantics defined by respectively least or greatest fixed points. Both these classes are further investigated by Talbot et al. [2000], where the syntax is extended by a new operation called membership expression. This extension allows to capture both classes of the definite and co-definite constraints while preserving their main properties: existence of the least or greatest solutions and DEXPTIME-completeness of the satisfiability problem.

Two years after the formulation of the satisfiability question by Heintze and Jaffar, Aiken and Wimmers [1992] proved decidability for the class of *positive* set constraints. This is a very natural class of constraints defined by choosing the boolean set operators (i.e., intersection, union and complement, but not projection).

For several years (until the publication of the conference version of Charatonik and Podelski [2002] in 1997) this class was considered to be incomparable with definite set constraints because of the missing projection operator. The algorithm of Aiken and Wimmers worked in NEXPTIME, and their decidability proof was followed by other proofs of the same result: Gilleron et al. [1993a, 1999] gave a proof based on automata theoretic techniques and Bachmair et al. [1993] gave a proof using the decision procedure for the first order theory of monadic predicates, providing also NEXPTIME-completeness of the problem. In Aiken et al. [1993], yet another algorithm has been presented and a detailed analysis of the complexity of positive set constrains depending on the number of constructors of given arity has been given.

There were several attempts to solve the problem with all Boolean operators and projections. The first partial solution was given by Heintze and Jaffar [1994] in their paper about definite set constraints. The second partial solution (for negative occurrences of projections in positive set constraints) was given by Bachmair et al. [1993]. In the same paper, the authors observed that using projections one can encode negated inclusions (we repeat this reduction in the next chapter). Following this idea, several papers [Gilleron et al. 1993b; Aiken et al. 1995; Stefansson 1994; Charatonik and Pacholski 1994a; Gilleron et al. 1999] gave the decidability of the satisfiability problem for constraints with negated inclusions. The full solution was given by Charatonik and Pacholski [1994b]. This was done by applying the techniques developed in Charatonik and Pacholski [1994a] to the idea of Bachmair, Ganzinger and Waldmann. In this article, we give another proof based on automata that run on DAG representations of (sets of) trees.

Set constraints were also studied from the logical and topological point of view [Kozen 1993, 1995; Cheng and Kozen 1996; Charatonik and Podelski 1996], and also in domains different from the Herbrand universe [Heintze 1993; McAllester et al. 1996; Charatonik 1998a; Goubault-Larrecq 2002], over infinite trees [Charatonik and Podelski 1998; Rychlikowski and Truderung 2004], with additional set operators [Bachmair et al. 1993; Charatonik and Pacholski 1994a; Talbot et al. 2000], with restricted set operators [Charatonik and Podelski 1998, 2002; Charatonik et al. 2000; Charatonik and Talbot 2002] and over nonempty sets [Müller et al. 1997; Charatonik and Podelski 1996, 2002]. It turns out that *nonempty-set constraints* have interesting algorithmic properties [Müller et al. 1997], and enjoy a fundamental property of independence for set constraints with intersection [Charatonik and Podelski 1996; Müller et al. 1999; Charatonik and Podelski 2002]. Kozen [1994, 1998] explored the use of set constraints in constraint logic programming. Uribe [1992] used set constraints in order-sorted languages. The first-order theory of set constraints is undecidable. This has been established in Seynhaeve et al. [1997, 2001], and improved in Charatonik [1998b]. The strongest known results, that is, undecidability of the $\exists\forall\forall$-fragment, has been given in Talbot [2000].

1.2. EXAMPLE PROGRAM ANALYSIS. Set constraints were successfully applied in program analysis. Below, we give a very simple example of such an application. More examples can be found in tutorials [Heintze and Jaffar 1994; Aiken 1999]. Of course, for efficiency reasons, existing toolkits for constraint-based program analyses, such as Kodumal and Aiken [2005], implement constraint-solving algorithms specialized for restricted classes of constraints. In particular, constraints below can be solved with methods much simpler then these presented in this article.

Consider a simple imperative program for list reversal

```
    y:=nil
A:  while (x≠nil) do
B:    y:=cons(head(x),y)
      x:=tail(x)            C:
    done                    D:
```

where `nil` and `cons` are list constructors and `head` and `tail` are list destructors. We are interested in four program points: entry to the loop (A:), entry to the body of the loop (B:), and exits from the loop body and from the loop (C: and D:, respectively). For each of these program points and each variable in the program we introduce a set variable denoting the set of values assumed by the given variable (i.e., its type) at the given program point. The following constraints can be inferred from this program.

$$
\begin{array}{lll}
Y_A \supseteq \texttt{nil} & & \\
Y_B \supseteq Y_A \cup Y_C & & X_B \supseteq X_A \cap \overline{\texttt{nil}} \\
Y_C \supseteq \texttt{cons}(\texttt{cons}^{-1}(X_B), Y_B) & & X_C \supseteq \texttt{cons}^{-2}(X_B) \\
Y_D \supseteq Y_A \cup Y_C & & X_D \supseteq (X_A \cap \texttt{nil}) \cup (X_C \cap \texttt{nil})
\end{array}
$$

Here $\texttt{cons}^{-1}$ and $\texttt{cons}^{-2}$ are projections to the first and the second argument, respectively, of a set of lists, that correspond to `head` and `tail` operations. The least solution of these constraints assigns empty sets to set variables X_A, X_B, X_C, X_D, which indicates a problem with uninitialized program variable `x`. If we supply the analysis with the information that at the program point A the variable `x` can be any list, then the least solution provides the information that at the program point B the program variable `x` is a nonempty list, and at point D it is the empty list.

1.3. OUR CONTRIBUTION. The main contribution of this article is the proof that the satisfiability problem for sets constraints with projections is decidable in NEXPTIME. Besides giving this proof we introduce a new technique of pumping in the context of directed acyclic graphs; we believe that this technique is of independent interest and can find other applications.

This article is an extended and revised version of Charatonik and Pacholski [1994a, 1994b]. There are two main changes when compared with the conference versions. First, we have replaced the monadic class used in Charatonik and Pacholski [1994a], and tree set automata used in Charatonik and Pacholski [1994b] by automata on directed acyclic graphs, which seem to be best suited to study the satisfiability problem of set constraints. Then, we have completely revised the combinatorial part. We hope that this part of the proof is now comprehensible. We are sorry, but we were unable to make it easy.

The article is organized as follows. In Section 3, we introduce the notions of automata to be used later. In Section 4, we translate the satisfiability problems for some classes of set constraints into the non-emptiness problems for corresponding classes of automata. In Section 5, we study the nonemptiness problem for these classes of automata, and we prove that this problem is in NP for all of these classes. The proofs in Section 5 are complete except of Lemma 5.15, whose proof is given in Section 6.

2. *Preliminaries*

2.1. SYNTAX AND SEMANTICS OF SET CONSTRAINTS. A *signature* is a finite set Σ of function symbols, each of which has assigned *arity* (the number of arguments). Symbols of arity 0 are called *constants*. The set T_Σ of all ground terms over Σ is called the *Herbrand universe* and is defined as the least set containing constant symbols in Σ and closed under the application of function symbols in Σ, which means that whenever $t_1, \ldots, t_n \in T_\Sigma$ and $f \in \Sigma$ is an n-ary function symbol, then $f(t_1, \ldots, t_n) \in T_\Sigma$.

Syntactically, *positive* and *negative* set constraints are inclusions of the form $E \subseteq E'$ and $E \not\subseteq E'$ where the expressions E and E' are given by the grammar

$$E \ ::= \ X \ \mid \ \bot \ \mid \ E \cap E \ \mid \ \overline{E} \ \mid \ f(E, \ldots, E)$$

where X stands for a variable from a given set, and f is a function symbol from a given signature Σ. A system of set constraints is a finite conjunction of such constraints; we will often identify such conjunction with the set of its conjuncts. In the case of set constraints with *projections* we allow also expressions of the form $f^{-i}(E)$, where i is a number between 1 and the arity of f. We will sometimes use $\top$ and $E_1 \cup E_2$ as an abbreviation for $\overline{\bot}$ and $\overline{\overline{E_1} \cap \overline{E_2}}$, respectively. We will also identify $\overline{\overline{E}}$ with E.

Semantically, the variables range over subsets of the Herbrand universe T_Σ. The symbol $\bot$ denotes the empty set; the Boolean connectives are interpreted in the usual way, and function symbols operate on sets as follows:

$$f(S_1, \ldots, S_n) = \{f(t_1, \ldots, t_n) \mid t_1 \in S_1, \ldots, t_n \in S_n\}$$

and

$$f^{-i}(S) = \{t_i \mid \exists t_1 \ldots \exists t_{i-1} \exists t_{i+1} \ldots \exists t_n : f(t_1, \ldots, t_n) \in S\}.$$

Note that in the case of $n = 0$, if c is a function symbol of arity 0, the set expression c denotes the set $\{c\}$.

Now, let us recall the notion of a solution of a system of set constraints SC. Let Var denotes the set of set variables that appear in SC,and let $\sigma : \mathsf{Var} \to 2^{T_\Sigma}$ be an assignment of subsets of T_Σ to variables in Var. Then σ extends in a unique way to a function from expressions to subsets of T_Σ, which abusing the notation we also denote by σ. This extension is defined as follows: $\sigma(\bot) = \emptyset$, $\sigma(E_1 \cap E_2) = \sigma(E_1) \cap \sigma(E_2)$, $\sigma(\overline{E}) = T_\Sigma \setminus \sigma(E)$,

$$\sigma(f(E_1, ..., E_k)) = f(\sigma(E_1), ..., \sigma(E_k))$$

and

$$\sigma(f^{-i}(E)) = f^{-i}(\sigma(E))$$

An assignment $\sigma : \mathsf{Var} \to 2^{T_\Sigma}$ is solution of (SC) if $\sigma(E) \subseteq \sigma(E')$, for each positive constraint $E \subseteq E'$ in SC and $\sigma(E) \not\subseteq \sigma(E')$, for each negative constraint $E \not\subseteq E'$ in SC.

A system SC of set constraints is satisfiable if there exists an assignment of subsets of T_Σ to the variables satisfying all the constraints in SC.

In the presence of projections, negative set constraints can be expressed using the positive ones, so here it suffices to consider only positive set constraints:

The constraint $a \subseteq f^{-1}(f(a, E))$ is equivalent to $E \not\subseteq \bot$ and, more generally, $a \subseteq f^{-1}(f(a, E_1 \cap \overline{E_2}))$ is equivalent to $E_1 \not\subseteq E_2$. In the following, by positive set constraints, we mean constraints without projections and without negative constraints; by positive and negative set constraints, we mean constraints without projections, and by set constraints with projections we mean constraints without negative constraints.

2.2. REDUCTION TO THE BINARY CASE. To simplify some reasonings in Section 6, we shall assume that the signature does not contain symbols of arity greater than two. The following lemma shows that we do not loose generality with this assumption. Note that this reduction works for all classes of constraints considered in this article.

LEMMA 2.1. *The problem whether a system of set constraints with projections has a solution can be reduced (in polynomial time) to such a problem over vocabularies containing no function symbols of arity greater then two.*

PROOF. Let SC be a system of set constraints over a signature Σ that contains symbols of arity greater than two. Let Σ' be a signature containing all constants and all unary symbols from Σ, and $n-1$ binary function symbols $f_1, \ldots f_{n-1}$ for each n-ary function symbol $f \in \Sigma$ with $n \geq 3$. For each term $t \in T_\Sigma$ we define its representation $t' \in T_{\Sigma'}$ as follows. Constants are represented by themselves i.e. $c' = c$ for all constant symbols $c \in \Sigma$. We put $g(t)' = g(t')$ for all unary symbols g, and finally $f(t_1, \ldots, t_n)' = f_1(t'_1, f_2(t'_2, \ldots f_{n-1}(t'_{n-1}, t'_n)\ldots)$, for all n-ary symbols $f \in \Sigma$.

Now, we shall construct a system SC' of set constraints over the signature Σ'. First, to make sure that we work only with terms representing original terms, we replace each occurrence of $\top$ by a new variable $X_\top$, add a constraint $X_\top = \bigcup_{f\in\Sigma} f(X_\top, \ldots, X_\top)$, and add a constraint $X \subseteq X_\top$ for each variable X occurring in SC. Then, we replace each occurrence of an expression $f(E_1, \ldots, E_n)$ (including $f(X_\top, \ldots, X_\top)$) by $f_1(E_1, f_2(E_2, \ldots f_{n-1}(E_{n-1}, E_n)\ldots)$ and each occurrence of a projection $f^{-i}(E)$ by the expression $f_i^{-1}(f_{i-1}^{-2}(\ldots f_1^{-2}(E)\ldots))$.

Now, it is easy to see that the system so obtained has a solution if and only if the original one has. For one direction, if σ is a solution of SC then defining $\sigma'(X) = \{t' \mid t \in \sigma(X)\}$ we obtain a solution of SC'. Conversely, if σ' is a solution of SC' then for each variable X, since $\sigma'(X) \subseteq \sigma'(X_\top)$, we have that $\sigma'(X)$ contains only terms that are representations of terms in T_Σ and thus defining $\sigma(X) = \{t \mid t' \in \sigma'(X)\}$ we obtain a solution of SC. □

2.3. THE MAIN RESULT. The main contribution of this article is a decision procedure for solving systems of set constraints with projections. An instance of the problem is a conjunction of inclusions

$$E_1 \subseteq E'_1 \wedge \cdots \wedge E_n \subseteq E'_n$$

where the expressions E_j and E'_j, for $j = 1, \ldots, n$, are given by the grammar

$$E \ ::= \ X \ \mid \ \bot \ \mid \ E \cap E \ \mid \ \overline{E} \ \mid \ f(E, \ldots, E) \ \mid \ f^{-i}(E).$$

The following theorem is proved at the end of Chapter 5. Together with NEXPTIME-hardness proved in Bachmair et al. [1993] it gives NEXPTIME-completeness of the problem.

THEOREM 2.2. *The satisfiability problem for set constraints with projections is decidable in* NEXPTIME.

3. *Automata for Set Constraints*

In this Section, we introduce automata theoretic tools that provide for a translation of the problem of finding solutions to set constraints to the problem of finding accepting runs of automata. To fix the terminology, we recall some standard definitions.

Definition 3.1 (*DAG*). Let $G = \langle V, E\rangle$ be a directed acyclic graph.

—If $u, v \in V$ and there is an edge from v to u (i.e., if $\langle v, u\rangle \in E$, then we call v a predecessor of u, and u a successor of v.

—$G = \langle V, E\rangle$ is ordered if for each $v \in V$ the set of edges outgoing from v is linearly ordered.

—If u is a successor of v and there are $i-1$ edges outgoing from v that are smaller than $\langle v, u\rangle$ in the linear ordering of edges outgoing from v, then we call u the ith successor of v.

—A subgraph $G' = \langle V', E'\rangle$ of G is *closed* if, for every node v of G', G' contains all successors of v in G.

—A *root* in G is any node without predecessors; a leaf is any node without successors.

Definition 3.2 (*t-dag*).

(1) A DAG representation of a set of ground terms (s-dag in short) over a signature Σ is a labeled directed acyclic ordered graph $G = \langle V, E, l\rangle$ such that
   —$l : V \to \Sigma$,
   —if a node v is labeled with a function symbol of arity n, then there are exactly n edges outgoing from v in the graph, and
   —G does not contain two different isomorphic closed labeled subgraphs.

(2) A representation of a term (t-dag in short) is an s-dag with exactly one root.

To obtain a DAG representation of a set of ground terms, it is enough to take the usual tree representation and to identify every two isomorphic subtrees.

Definition 3.3. The term $t(v)$ represented by the node $v \in V$ of a s-dag $G = \langle V, E, l\rangle$ is defined inductively as follows.

—if v is a leaf, then $t(v)$ is the constant symbol $l(v)$,

—if v has successors $v_1, \ldots, v_n$ (in this order) and $l(v) = f$, then $t(v)$ is the term $f(t(v_1), \ldots, t(v_n))$.

The set of terms represented by G is the set $\{t(v) \mid v \in V\}$. If G is a t-dag , then the term represented by G is the term represented by the root of G.

DAG representations of terms have been used in the study of unification (see, e.g., Paterson and Wegman [1978]). The last condition in the definition of an s-dag above is known as *maximal sharing of structure*. The assumption that an s-dag is ordered is needed to assure that the t-dags representing $f(a, b)$ and $f(b, a)$ are

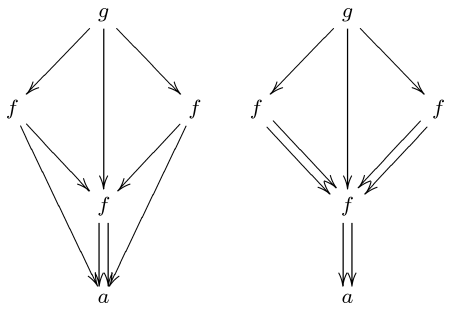

FIG. 1. A t-dag representing the term $g(f(a, f(a, a)), f(a, a), f(f(a, a), a))$ and a graph not being a t-dag.

not isomorphic. Note that we consider isomorphisms of labeled graphs, that is, the isomorphisms preserving labelings.

The representation of a set T of terms is the smallest s-dag G such that, for every term $t \in T$, there exists a node v in G such that the closed subgraph of G rooted at v is the t-dag representing t. In the following, we will often refer to the s-dag G_Σ representing the Herbrand universe T_Σ: this is the graph $\langle T_\Sigma, E, l\rangle$ where for every term t of the form $f(t_1, \ldots, t_n)$ the edge relation E contains $\langle t, t_1\rangle, \ldots, \langle t, t_n\rangle$ (in this order) and $l(t) = f$. Note that s-dags may be infinite and not connected. Note that every t-dag has exactly one root while s-dags may have an arbitrary number of roots (including zero - in which case the s-dag is either empty or infinite). Moreover, each t-dag represents the set of its subterms.

Figure 1 gives an example of a t-dag, and a graph which is not a t-dag because it contains two isomorphic copies of the graph representing $f(f(a, a), f(a, a))$.

Definition 3.4 (*Automaton*).

(1) An automaton is a tuple $\langle \Sigma, Q, \Delta\rangle$ where Σ is a finite signature, Q is a finite set of states, and Δ is a set of transitions of the form $f(q_1, \ldots, q_n) \to q$ with $q, q_1, \ldots, q_n \in Q$, $f \in \Sigma$, and n being the arity of f.
(2) An automaton is called *complete* if for each $f \in \Sigma$ and each sequence $q_1, \ldots, q_n$ where n is the arity of f, there exists $q \in Q$ such that $f(q_1, \ldots, q_n) \to q$ belongs to Δ.
(3) An automaton is called *deterministic* if for all f and $q_1, \ldots, q_n$ there exists at most one q with $f(q_1, \ldots, q_n) \to q \in \Delta$, and *nondeterministic* otherwise.

Definition 3.5 (*Run*). A run of an automaton $\langle \Sigma, Q, \Delta\rangle$ on a given s-dag G is a mapping ρ from the set of nodes of G to the set of states Q such that for each node v and each $f \in \Sigma$, if v is labeled with f and $v_1, \ldots, v_n$ are the successors of v, then Δ contains a transition $f(\rho(v_1), \ldots, \rho(v_n)) \to \rho(v)$.

Definition 3.6 (*t-dag Automaton*).

(1) A t-dag automaton is a tuple $\langle \Sigma, Q, \Delta, F\rangle$ such that $\langle \Sigma, Q, \Delta\rangle$ is an automaton and $F \subseteq Q$ is the set of final states.
(2) A run ρ on a t-dag G is *successful* if $\rho(v) \in F$, where v is the root of G.
(3) An automaton $\mathcal{A}$ accepts a t-dag G if there exists a successful run of $\mathcal{A}$ on G.

The main and only difference between tree and t-dag representations of terms is that if a term has several appearances of a subterm, then in the t-dag (s-dag) representation all these appearances are represented by the same element, while in the tree representation each appearance is represented by a distinct copy of the same tree. This has, however, consequences for respective automata. Tree automata may assign different states to different trees representing different appearances of the same subterm, which is not possible for t-dags. As a consequence t-dag automata accept less terms than tree automata.

Example 3.7. As a tree automaton, the automaton $\langle \{a, f(\cdot,\cdot)\}, \{q_1, q_2, q\}, a \to q_1,\ a \to q_2, f(q_1, q_2) \to q\}, \{q\}\rangle$ accepts $f(a, a)$ but as a t-dag automaton it does not accept any t-dag representation of any tree.

The following definition of a request is motivated by application of automata to solving set constraints with projections. Consider an inclusion $X \subseteq f^{-2}(Y)$. For all solutions σ and all terms $t \in \sigma(X)$ there must exist a term s (a witness for t and projection $f^{-2}(Y)$) such that $f(s, t) \in \sigma(Y)$. Intuitively, an automaton while processing t sends a request for existence of such a witness; this request should be granted while processing $f(s, t)$.

Definition 3.8 (*Request*). Let $\langle \Sigma, Q, \Delta\rangle$ be an automaton and ρ a run on an s-dag G.

(1) A triple $\langle f, i, R\rangle$ is a *request* if f is a function symbol in Σ of arity $n \geq 2$, $i \in \{1, \ldots, n\}$, and $R \subseteq Q$. We denote the set of requests by $\mathcal{R}$.
(2) We say that the request $\langle f, i, R\rangle$ at a node t in G is *granted* by a node u in G if u is labeled with the symbol f, $\rho(u) \in R$ and t is the ith successor of u.

Definition 3.9 (*Automaton with Projections*). An automaton with projections is a tuple $\langle \Sigma, Q, \Delta, \pi\rangle$ where

(1) $\langle \Sigma, Q, \Delta\rangle$ is an automaton
(2) $\pi : Q \to 2^{\mathcal{R}}$ is a function assigning a set of requests to each state of Q.

Example 3.10. Consider the automaton $\langle \{a, b, f(\cdot,\cdot)\}, \{q_1, q_2\}, \{a \to q_1,\ b \to q_1, f(q_1, q_1) \to q_2\}, \pi\rangle$ where $\pi(q_1) = \{\langle f, 2, \{q_2\}\rangle\}$, and a run of this automaton on the s-dag representing the set $\{f(a, a), f(a, b)\}$. The request $\langle f, 2, \{q_2\}\rangle$ is granted at node a by the node $f(a, a)$ and at node b by the node $f(a, b)$. On the other hand, if we restrict the run to the t-dag representing $f(a, b)$ then the same request at the node b is granted by the node $f(a, b)$, but it is not granted at the node a.

Note that since the granting node is a predecessor, no request can be granted at any root of a s-dag.

Definition 3.11 (*Faithful Run*). Let G be an s-dag with set of nodes V and let V_r be a (possibly empty) set of roots of G. We say that a run ρ of an automaton $\mathcal{A} = \langle \Sigma, Q, \Delta, \pi\rangle$ on the s-dag G is faithful up to V_r with respect to π if for each node $t \in V - V_r$ and each request in $\pi(\rho(t))$, this request is granted at the node t. We say that ρ is faithful if it is faithful up to the empty set.

Example 3.12. Consider the automaton and the run from Example 3.10. If we have $\pi(q_2) = \emptyset$, then this run is faithful. However, if we add any request to $\pi(q_2)$, then this run becomes only faithful up to $\{f(a, a), f(a, b)\}$.

4. *Solving Set Constraints*

The basic idea how to define an automaton representing a given system SC of set constraints is quite simple. Automata run on the s-dag representation G_Σ of the Herbrand universe. If σ is a solution of SC, then σ assigns to each expression E a subset $\sigma(E)$ of the Herbrand universe. A run of an automaton on the Herbrand universe assigns to each element t a state which represents a collection of expressions E such that $t \in \sigma(E)$. Since the number of relevant expressions is finite, the number of states is finite. Intuitively, a state φ is assigned to the t-dags that represent ground terms belonging to the intersection of all sets $\sigma(E)$ where $E \in \varphi$.

For a given system $SC = \bigwedge_{i\in I} E_i \subseteq E'_i \wedge \bigwedge_{j\in J} E_j \not\subseteq E'_j$ of set constraints, by $E(SC)$, we mean the smallest set containing the expressions E_i, E'_i, E_j, E'_j for all $i \in I$ and $j \in J$ and closed under subexpressions and complementation. For example, if SC consists of the constraint $f(\overline{X}, Y) \subseteq \overline{g(\top \cap X)}$, then

$$\begin{aligned} E(SC) = {} & \{f(\overline{X}, Y), X, Y, g(\top \cap X), \top \cap X, \top\} \\ & \cup \{\overline{f(\overline{X}, Y)}, \overline{X}, \overline{Y}, \overline{g(\top \cap X)}, \overline{\top \cap X}, \bot\}. \end{aligned}$$

Definition 4.1 (*Automaton Coresponding to SC*). Let SC be a system of set constraints. $\mathcal{A}(SC) = \langle \Sigma, Q, \Delta \rangle$ is the automaton corresponding to SC if

(1) Q is the set of all subsets q of $E(SC)$ such that
   —$\bot \notin q$,
   —$E \in q$ iff $\overline{E} \notin q$,
   —if $(E_1 \cap E_2) \in E(SC)$, then $(E_1 \cap E_2) \in q$ iff $E_1, E_2 \in q$, and
   —if $E \subseteq E' \in SC$ and $E \in q$, then $E' \in q$.

   and

(2) Δ is the set of transitions of the form $f(q_1, \ldots, q_n) \to q$ such that
   —$f \in \Sigma$, the arity of f is $n \geq 0$, and $q_1, \ldots, q_n, q \in Q$,
   —q does not contain any expression of the form $g(E'_1, \ldots, E'_m)$ with $g \neq f$, and
   —if $f(E_1, \ldots, E_n) \in E(SC)$, then $f(E_1, \ldots, E_n) \in q$ iff $E_i \in q_i$ for all $i = 1, \ldots, n$.

As we will see in the next section, the runs of the automaton $\mathcal{A}(SC)$ correspond to solutions of SC. In this correspondence, the definition of states of $\mathcal{A}(SC)$ is used to make sure that the solutions σ satisfy the input constraints and $\sigma(\bot) = \emptyset$, $\sigma(E_1 \cap E_2) = \sigma(E_1) \cap \sigma(E_2)$ and $\sigma(\overline{E}) = T(\Sigma) \setminus \sigma(E)$, while the definition of transitions is used to make sure that $\sigma(f(E_1, ..., E_k)) = f(\sigma(E_1), ..., \sigma(E_k))$.

Example 4.2. Consider the system of set constraints $\{\mathsf{nil} \subseteq \mathit{list},\ \mathsf{cons}(\top, \mathit{list}) \subseteq \mathit{list}\}$ where $\Sigma = \{\mathsf{nil}, \mathsf{cons}(\cdot, \cdot)\}$ and list is the only set variable. Then $E(SC) = \{\top, \bot, \mathsf{nil}, \overline{\mathsf{nil}}, \mathsf{cons}(\top, \mathit{list}), \overline{\mathsf{cons}(\top, \mathit{list})}, \mathit{list}, \overline{\mathit{list}}\}$,

the set of states of the automaton $\mathcal{A}(SC)$ is $\{q_1, q_2, q_3, q_4, q_5\}$ where

$$\begin{aligned} q_1 &= \{\top, \overline{\mathsf{nil}}, \overline{\mathsf{cons}(\top, \overline{list})}, \overline{list}\},\\ q_2 &= \{\top, \overline{\mathsf{nil}}, \overline{\mathsf{cons}(\top, list)}, list\},\\ q_3 &= \{\top, \mathsf{nil}, \overline{\mathsf{cons}(\top, list)}, list\},\\ q_4 &= \{\top, \overline{\mathsf{nil}}, \mathsf{cons}(\top, list), list\},\\ q_5 &= \{\top, \mathsf{nil}, \mathsf{cons}(\top, list), list\}, \end{aligned}$$

and the set of transitions Δ consists of 20 transitions $\mathsf{cons}(q, q') \to q_4$ for all $q \in \{q_1, q_2, q_3, q_4, q_5\}$ and $q' \in \{q_2, q_3, q_4, q_5\}$ and one transition $\mathsf{nil} \to q_3$. Note that the states q_1, q_2, q_5 are not reachable as there is no transition leading there, and they may be removed.

The automaton $\mathcal{A}(SC)$ may be nondeterministic as the following example shows.

Example 4.3. Let $\Sigma = \{\mathsf{nil}, \mathsf{cons}(\cdot, \cdot)\}$ and let SC consists of just one constraint $X \subseteq Y$. Then, the states of the automaton are $Q = \{\{X, Y\}, \{\overline{X}, Y\}, \{\overline{X}, \overline{Y}\}\}$. The transition relation contains $\mathsf{nil} \to q$ and $\mathsf{cons}(q_1, q_2) \to q$ for all $q, q_1, q_2 \in Q$.

The construction given above is a slight reformulation of a definition given in Gilleron et al. [1993a, 1993b], and has been influenced by monadic formulas of Bachmair et al. [1993], and by hypergraphs of Aiken et al. [1995].

4.1. POSITIVE SET CONSTRAINTS. An automaton $\langle \Sigma, Q, \Delta \rangle$ is a subautomaton of $\langle \Sigma, Q', \Delta' \rangle$ if $Q \subseteq Q'$ and $\Delta \subseteq \Delta'$; if the two automata come together with sets of final states F and F', respectively (that is, if they are t-dag automata), then we also require $F \subseteq F'$.

For any run ρ of the automaton defined above and any expression $E \in E(SC)$, we define

$$[\![E]\!]_\rho = \{t \in T_\Sigma \mid E \in \rho(t)\}.$$

THEOREM 4.4. *Let SC be a system of positive set constraints, $\mathcal{A}(SC)$ be the automaton corresponding to SC, and G_Σ be the s-dag representing the Herbrand universe. The following conditions are equivalent*

(1) *SC is satisfiable*
(2) *there exists a run of $\mathcal{A}(SC)$ on the s-dag G_Σ*
(3) *the automaton $\mathcal{A}(SC)$ has a complete sub-automaton*

PROOF. We start with the implication $(1 \Rightarrow 2)$. Assume that σ is a solution of SC. For any node t of G_Σ define $\rho(t) = \{E \in E(SC) \mid t \in \sigma(E)\} \cup \{\overline{E} \mid E \in E(SC),\ t \notin \sigma(E)\}$. Note that $\rho(t)$ satisfies the conditions defined in point 1 of Definition 4.1 and thus $\rho(t)$ is a state of $\mathcal{A}(SC)$. Moreover, if t is labeled with a function symbol f of arity n, then $t = f(t_1, \ldots, t_n)$ for some terms $t_1, \ldots, t_n$ and then for any expression $f(E_1, \ldots, E_n) \in E(SC)$ the definition of σ implies that $t \in \sigma(f(E_1, \ldots, E_n))$ iff $t_i \in \sigma(E_i)$ for all $i = 1, \ldots, n$, which means that $f(\rho(t_1), \ldots, \rho(t_n)) \to \rho(t)$ is a transition of $\mathcal{A}(SC)$. Hence ρ is a run of $\mathcal{A}(SC)$ on G_Σ.

For the implication $(2 \Rightarrow 3)$ assume that ρ is a run of $\mathcal{A}(SC)$ on G_Σ and define $\mathcal{A}$ to be a subautomaton of $\mathcal{A}(SC)$ restricted to the set of states $\{\rho(t) \mid t \in T_\Sigma\}$.

Obviously $\mathcal{A}$ is complete: if $f \in \Sigma$ and $q_1, \ldots, q_n$ are states of $\mathcal{A}$ then there exist terms $t_1, \ldots, t_n$ such that $\rho(t_1) = q_1, \ldots, \rho(t_n) = q_n$. Then $f(q_1, \ldots, q_n) \to \rho(f(t_1, \ldots, t_n))$ is a transition of $\mathcal{A}$.

We finish the proof with the implication $(3 \Rightarrow 1)$. Assume that $\mathcal{A} = \langle \Sigma, Q, \Delta \rangle$ is a minimal complete subautomaton of $\mathcal{A}(SC)$, that is, that for any $f \in \Sigma$ and any sequence of states of $\mathcal{A}$ (where n is the arity of f) there is exactly one state q such that $f(q_1, \ldots, q_n)$ is a transition of $\mathcal{A}$ (which is also a transition of $\mathcal{A}(SC)$). By induction on terms we define a run ρ on G_Σ: for $f(t_1, \ldots, t_n) \in T_\Sigma$, $\rho(f(t_1, \ldots, t_n))$ is the state q such that $f(\rho(t_1), \ldots, \rho(t_n)) \to q \in \Delta$. Next we define a valuation: $\sigma(X) = [\![X]\!]_\rho$ for all variables X. An easy induction on structure of expressions (using Definition 4.1 and the definition of the extension of a valuation to expressions from Chapter 2) shows that $\sigma(E) = [\![E]\!]_\rho$ for all $E \in E(SC)$. Here we show only the induction step for $f(E_1, \ldots, E_n)$, which is the most interesting one.

Assume that $t \in \sigma(f(E_1, \ldots, E_n))$. Then, by the definition of σ, we have that $t = f(t_1, \ldots, t_n)$ and $t_1 \in \sigma(E_1), \ldots, t_n \in \sigma(E_n)$. By induction hypothesis we have $t_1 \in [\![E_1]\!]_\rho, \ldots, t_n \in [\![E_n]\!]_\rho$, therefore $E_1 \in \rho(t_1), \ldots, E_n \in \rho(t_n)$. By the completeness of the automaton and the definition of Δ there is a transition $f(\rho(t_1), \ldots, \rho(t_n)) \to q$ such that $q = \rho(f(t_1, \ldots, t_n))$ and $f(E_1, \ldots, E_n) \in q$. Hence, $t \in [\![f(E_1, \ldots, E_n)]\!]_\rho$. Conversely, assume that $t \in [\![f(E_1, \ldots, E_n)]\!]_\rho$. Then, there exists a state q such that $\rho(t) = q$ and $f(E_1, \ldots, E_n) \in q$. By the definition of Δ the state q is reachable only with transitions starting with the symbol f, so we have $t = f(t_1, \ldots, t_n)$ and for all $i = 1, \ldots, n$, $\rho(t_i) = q_i$ such that $E_i \in q_i$. Hence $t_i \in [\![E_i]\!]_\rho$. By induction hypothesis we have $t_i \in \sigma(E_i)$, which implies $t \in \sigma(f(E_1, \ldots, E_n))$ and ends the proof of the equality $\sigma(f(E_1, \ldots, E_n)) = [\![f(E_1, \ldots, E_n)]\!]_\rho$.

Finally, the last condition of point 1 of Definition 4.1 gives that $[\![E]\!]_\rho \subseteq [\![E']\!]_\rho$ for all $E \subseteq E' \in SC$, which means that σ is a solution of SC. $\square$

Relations to the Other Approaches. A t-dag automaton can be seen as a hypergraph, where states correspond to nodes, and transitions correspond to hyperedges of the hypergraph. Up to minor details, the construction of the t-dag automaton corresponding to a system of set constraints coincides with the corresponding construction of the hypergraph corresponding to the same system of set constraints in Aiken et al. [1993, 1995]. Under this correspondence, the equivalence between conditions 1 and 3 in Theorem 4.4 coincides with Theorem 1 in Aiken et al. [1993] and Theorem 3 in Aiken et al. [1995].

A t-dag automaton is essentially equivalent to a tree set automaton as defined in Gilleron et al. [1993a, 1993b]. The equivalence between conditions 1 and 2 directly corresponds to Theorem 15 in Gilleron et al. [1993a], the equivalence between 2 and 3 to Theorem 4 in Gilleron et al. [1993a].

Bachmair et al. [1993] prove that a system SC of positive set constraints is satisfiable iff the corresponding monadic formula $\Phi(SC)$ is satisfiable. The formula $\Phi(SC)$ is built using predicates named P_E for $E \in E(SC)$. Roughly, one obtains this formula by taking the conjunction of all conditions from Definition 4.1 where every occurrence of $E \in q$ is replaced by $P_E(q)$. Additionally, one has to add the quantifiers $\forall q$ in Condition 1 of Definition 4.1 and $\forall q_1 \cdots q_n \exists q$ in Condition 2 of Definition 4.1. Now a complete sub-automaton $\mathcal{A}'$ of $\mathcal{A}$ can be seen as the model of this formula: states of the automaton are elements of the model and the interpretation of predicates are given by: $P_E(q)$ holds iff E occurs in q. The

transitions $f(q_1, \ldots, q_n) \to q$ give the interpretation of the Skolem functions for the formula in the model (i.e., the assignment of such element q that makes the formula true for a given instance of elements $q_1, \ldots, q_n$). The subautomaton must be complete to ensure that f is defined *for all* elements of the model. Thus, the theorem reducing satisfiability of set constraints to satisfiability of monadic formulas can be seen as the equivalence between conditions 1 and 3, together with a standard result on monadic logic (see Ackermann [1954, page 34]) saying that a monadic formula with predicates from a given set P is satisfiable iff it has a model consisting of elements of 2^P.

4.2. POSITIVE AND NEGATIVE SET CONSTRAINTS. First, note that a system of set constraints with n negative constraints can be reduced to an equivalent system with one negative constraint only. Simply note that $E \not\subseteq E'$ is equivalent to $E \cap \overline{E'} \not\subseteq \bot$ and replace $E_1 \not\subseteq E'_1, \ldots, E_n \not\subseteq E'_n$ with

$$f(E_1 \cap \overline{E'_1}, \ldots, f(E_{n-1} \cap \overline{E'_{n-1}}, E_n \cap \overline{E'_n}) \cdots) \not\subseteq \bot,$$

where f is a binary function symbol in Σ.

Let SC be a system of set constraints with one negative constraint $E \not\subseteq \bot$. Define the set of final states of the automaton corresponding to SC as

$$F = \{q \in Q \mid E \text{ occurs positively in } q\}.$$

THEOREM 4.5. *Let $\mathcal{A} = \langle \Sigma, Q, \Delta, F \rangle$ be the automaton corresponding to SC as defined above. The following conditions are equivalent*

(1) *SC is satisfiable*
(2) *There exists a complete sub-automaton $\mathcal{A}'$ of the automaton $\mathcal{A}$ and a finite t-dag G accepted by $\mathcal{A}'$*

PROOF. It is enough to note that in the proof of Theorem 4.4 above, the correspondence between a run of the automaton and a solution of set constraint is such that a final state is reachable iff the set assigned to E is nonempty. □

Again there are strong relations between this theorem and the main theorems in the three papers showing decidability of the satisfiability problem for negative set constraints [Aiken et al. 1995; Charatonik and Pacholski 1994a; Gilleron et al. 1993b]. In Aiken et al. [1995], satisfiability for negative set constraints is reduced to another problem (Problem 4 in Aiken et al. [1995]), which translates (if SC contains one negative constraint) directly to the emptiness problem for a closed sub-automaton. The relation with Gilleron et al. [1993b] is also direct, since the satisfiability problem is there reduced to the emptiness problem of the corresponding tree set automaton. In case of Charatonik and Pacholski [1994a], the emptiness problem is reduced to existence of a model of the corresponding formula (which translates to a complete sub-automaton) satisfying some property (essentially, existence of an accepted t-dag). Moreover, the technique we used here for testing emptiness is the same as in Charatonik and Pacholski [1994a].

4.3. SET CONSTRAINTS WITH PROJECTIONS. Let SC be a system of set constraints with projections. We shall define an automaton with projections $\mathcal{A}(SC) = \langle \Sigma, Q, \Delta, \pi \rangle$ corresponding to SC. The construction is an extension of Definition 4.1.

Since we will have to reason interchangeably about s-dags and sets of terms, we will identify every node in an s-dag with the term it represents.

Definition 4.6 (*Automaton Corresponding to SC*). Let SC be a system of set constraints with projections and let $E(SC)$ denote the set of subexpressions of SC. By $\mathcal{A}(SC)$, we denote the automaton $\langle \Sigma, Q, \Delta, \pi \rangle$, where Σ, Q, Δ, π are defined below.

(1) $Q \subseteq 2^{E(SC)}$ is the family of subsets of $E(SC)$ that satisfy Condition 1 of Definition 4.1

(2) Δ is the set of transitions of the form $f(q_1, \ldots, q_n) \to q$ that satisfy Condition 2 of Definition 4.1 in conjunction with
—if $E \in q$ and $f^{-i}(E) \in E(SC)$, then $f^{-i}(E) \in q_i$

(3) $\pi(q) = \{\langle f, i, R\rangle \mid \exists E.\ f^{-i}(E) \in q, R = \{q' \mid E \in q'\}\}$.

LEMMA 4.7. *Let $\mathcal{A}(SC)$ be the automaton corresponding to a system SC of set constraints with projections, and let ρ be any run of this automaton on G_Σ. Then, for each expression $f^{-i}(E)$ occurring in $E(SC)$ we have*

$$f^{-i}([\![E]\!]_\rho) \subseteq [\![f^{-i}(E)]\!]_\rho. \tag{1}$$

Moreover, for unary symbols $f \in \Sigma$ we have

$$f^{-1}([\![E]\!]_\rho) = [\![f^{-1}(E)]\!]_\rho. \tag{2}$$

PROOF. Assume that $t \in f^{-i}([\![E]\!]_\rho)$ and let n be the arity of f. By the definition of the semantics of projection there exist a sequence of terms $t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n$ such that $f(t_1, \ldots, t_{i-1}, t, t_{i+1}, \ldots, t_n) \in [\![E]\!]_\rho$. From the definition of $[\![E]\!]_\rho$ we obtain that E occurs in $\rho(f(t_1, \ldots, t_{i-1}, t, t_{i+1}, \ldots, t_n))$; by the definition of ρ, $f(\rho(t_1), \ldots, \rho(t), \ldots, \rho(t_n)) \to \rho(f(t_1, \ldots, t_{i-1}, t, t_{i+1}, n \ldots, t_n)) \in \Delta$; by the definition of Δ, $f^{-i}(E)$ occurs in $\rho(t)$. Hence, $t \in [\![f^{-1}(E)]\!]_\rho$ which proves the inclusion (1).

For the equality (2), note that $t \in f^{-1}([\![E]\!]_\rho)$ if and only if $f(t) \in [\![E]\!]_\rho$, which, by the definition of Δ, is equivalent to $t \in [\![f^{-1}(E)]\!]_\rho$. □

Several authors [Bachmair et al. 1993; Charatonik and Podelski 2002] noticed that the algorithms for solving positive set constraints can be easily extended to set constraints with projections occurring only on the left-hand side of inclusions as well as constraints with projections for unary function symbols; a simple explanation is that $f^{-i}(E) \subseteq E'$ is equivalent to $E \cap f(\top, \ldots, \top) \subseteq f(\top, \ldots, E', \ldots, \top)$. In our setting this is done by the syntactic restrictions on transitions in Δ in the definition of the automaton corresponding to SC (see item 2 in Definition 4.6). This is why requests are not used in the proof of Lemma 4.7 above. The difficulty in solving constraints with projections is to determine whether there exists a run such that $[\![f^{-i}(E)]\!]_\rho \subseteq f^{-i}([\![E]\!]_\rho)$ for n-ary symbols $f \in \Sigma$, which we reduce to the question of existence of a faithful run, and in the proof of correctness of this reduction we exploit requests from item 3 of Definition 4.6. This corresponds to projections occurring on the right-hand side of inclusions.

THEOREM 4.8. *A system SC of set constraints with projections has a solution if and only if $\mathcal{A}(SC)$ admits a faithful run on the s-dag G_Σ.*

PROOF. The correspondence between solutions of SC and faithful runs of $\mathcal{A}(SC)$ is the same as in the proof of Theorem 4.4 in Section 4.1. To see that the run ρ corresponding to a solution σ is faithful, consider any node t in G_Σ and any request $\langle f, i, R\rangle \in \pi(\rho(t))$. By the definition of the automaton, there exists an expression E such that $f^{-i}(E) \in \rho(t)$, and $R = \{\psi \mid E \in \psi\}$. Since $t \in \sigma(f^{-i}(E))$, there exist nodes $t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n$, where n is the arity of f, such that $u = f(t_1, \ldots, t_{i-1}, t, t_{i+1}, \ldots, t_n) \in \sigma(E)$. Then, t is the ith successor of u where $\rho(u) \in R$, hence the request is granted.

Conversely, to see that the valuation σ corresponding to a faithful run ρ is a solution of SC, we have to show the sets $\sigma(f^{-i}(E))$ are properly defined, that is, that for each expression E such that $f^{-i}(E) \in E(SC)$ we have

$$f^{-i}([\![E]\!]_\rho) = [\![f^{-i}(E)]\!]_\rho. \tag{3}$$

In the case of unary function symbols f, the equality (3) follows from the equality (2) in Lemma 4.7. For the function symbols of greater arity, the inclusion $\subseteq$ is just inclusion (1) in Lemma 4.7.

To prove the $\supseteq$ part of the equality (3), assume that $t \in [\![f^{-i}(E)]\!]_\rho$. Then, by the definition of $[\![f^{-i}(E)]\!]_\rho$ we have $f^{-i}(E) \in \rho(t)$. By the definition of π, there is a request $\langle f, i, \{\psi \mid E \in \psi\}\rangle \in \pi(\rho(t))$. Since the run ρ is faithful, this request is granted at t, that is, there exists a predecessor s of t of the form $s = f(t_1, \ldots, t_{i-1}, t, t_{i+1}, \ldots, t_n)$ such that $\rho(s) \in \{\psi \mid E \in \psi\}$. This implies that $E \in \rho(s)$ which means that $s \in [\![E]\!]_\rho$. Since $s = f(t_1, \ldots, t_{i-1}, t, t_{i+1}, \ldots, t_n)$, we have $t \in f^{-i}([\![E]\!]_\rho)$. □

5. *Emptiness Problem for t-dag Automata*

In this Section, we prove that the emptiness problem for t-dag automata, that is, the problem of answering the question whether there exists a t-dag accepted by a given automaton, is decidable in NP. We use here a sort of pumping lemma technique; the main idea comes from Charatonik and Pacholski [1994a]. We present it as natural extension of the analogous proof for tree automata. One can prove NP-completeness of the problem by a rather easy encoding of the SAT problem.

5.1. INTUITION: TREE AUTOMATA.

To make the understanding of our approach easier, we first show our view on the emptiness problem for bottom-up tree automata. We show that if the language recognized by an automaton $\mathcal{A} = \langle \Sigma, Q, \Delta, F\rangle$ is nonempty, then there is a tree of depth at most $|Q|$, accepted by $\mathcal{A}$.

Consider a tree t accepted by $\mathcal{A}$, and a path of maximal length in this tree. If this path is longer then $|Q|$, there must be a state in $|Q|$ assigned to two different nodes v and v' in t, and one can remove all nodes between v and v'. One should not, however, forget here about the paths to other states that are needed to reach the final state and do not lie on the chosen path. The formalization of this method leads to the following notion of a skeleton of a run of an automaton on a tree. For the sake of simplicity, we identify here a tree with its graph representation, and a node in this graph with a tree rooted at this node.

If $f(q_1, \ldots, q_n) \to q$ is a transition of $\mathcal{A}$, then we say that this transition *uses* states $q_1, \ldots, q_n$ and *produces* the state q. When a run of an automaton is fixed and assigns a state q to a node v then we say that v *produces* q. A node representing

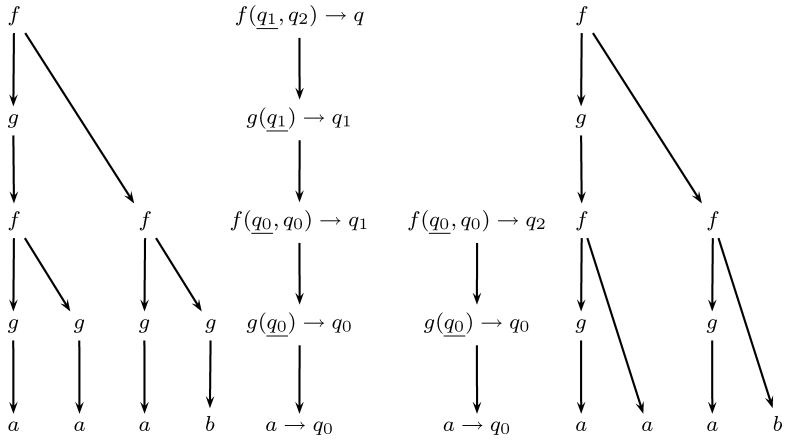

FIG. 2. An accepted tree, a skeleton and the tree reconstructed from the skeleton. Underlined states indicate the position of the leftmost maximal strict subterm.

a term t is *below* a node representing a term t' if the depth of t is smaller than the depth of t'. In the definition below, a maximal strict subterm of a term v is a strict subterm of v with maximal depth among all strict subterms of v.

Definition 5.1. A skeleton of a run ρ of a tree automaton $\mathcal{A}$ on a tree t is a subgraph G of t such that

—G contains the root of t, and
—for each node v in G, the leftmost maximal strict subterm of v is in G, and
—each node v in G is labeled with the transition used by $\mathcal{A}$ to reach v and the position of the maximal strict subterm of v, and
—if a state q is used in a transition labeling a node v, then it is produced by a transition labeling some other node below v.

If any path in a skeleton contains twice the same state, we can remove the appropriate part of the path, just as one does it in the pumping lemma for string automata. It is easy to see how one can reconstruct from a skeleton a (possibly different from the initial one) tree accepted by the automaton.

Example 5.2. Consider an automaton $\mathcal{A}$ with signature $\Sigma = \{a, b, g(\cdot), f(\cdot, \cdot)\}$, states $Q = \{q_0, q_1, q_2, q\}$, final states $F = \{q\}$, and transitions

$$\Delta = \{a \to q_0,\ \ b \to q_0,\ \ g(q_0) \to q_0,\ \ g(q_1) \to q_1,\\ f(q_0, q_0) \to q_1,\ \ f(q_0, q_0) \to q_2,\ \ f(q_1, q_2) \to q\}.$$

Figure 2 an example of a tree accepted by $\mathcal{A}$, a skeleton of a run of $\mathcal{A}$ on this tree and the tree induced from this skeleton. Figure 3 shows the "pumped-out" skeleton and the accepted tree induced by this skeleton.

This method does not work for t-dag automata. The reason is that the skeleton may contain two nodes with the same left-hand sides of the labeling transitions, the same paths below them, but different right-hand sides of the transitions (like the nodes labeled $f(\underline{q_0}, q_0) \to q_1$ and $f(\underline{q_0}, q_0) \to q_2$ on Figures 2 and 3). Such two nodes induce the same tree and thus must be identified in a DAG representation.

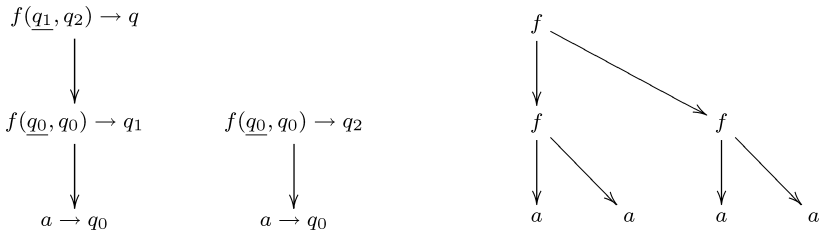

FIG. 3. An pumped-out skeleton and a smaller accepted tree.

On the other hand, they must be different in order to produce two different states in the run. To overcome this problem we extend the skeleton in order to obtain different graphs that are induced in such nodes. In case of trees, in both nodes labeled with a transitions starting with $f(\underline{q_0}, q_0)$, we used the same tree, with the smallest possible depth, producing the state q_0. In case of t-dags, we will use two different t-dags (the least and the second least in some order) producing q_0, which we note $q_0[1]$ and $q_0[2]$. The technical problem here is to make sure that we have enough nodes producing given state and to estimate the size of the skeleton.

5.2. NOTATIONS. A *leaf* in an s-dag G is a node without successors, labeled with a constant symbol. The *depth* of a node v in G is the length of the longest path leading from v to a leaf in G. The depth of a t-dag is the depth of its root. We say that a node v *lies below* a node v' if the depth of v is smaller than the depth of v'. We fix some linear order $\prec$ on nodes of the s-dag G_Σ representing the Herbrand universe T_Σ that extends the "lies below" partial order, and we use the same notation $\prec$ for the restriction of $\prec$ to the nodes of any s-dag G.

The *main successor* of a node v in G is the biggest according to $\prec$ successor of v. The *main path* for v in G is the sequence $v_0, v_1, \ldots, v_k$ of nodes in G such that $v_0 = v$, for all $0 \leq i < k$, v_{i+1} is the main successor of v_i, and v_k is a leaf. Note that the length of the main path for v coincides with the depth of v. The *position* of the main successor of v is the number identifying this node in the ordered sequence of successors of v, and is called the main position. If the main successor occurs more than once in this sequence, then the main position is the smallest (leftmost) position.

States occurring on the left-hand side of the arrow in a transition are called *used* by this transition; the state on the right-hand side is called *produced* by this transition. We say that a state q *accepts* a sub-t-dag rooted at a node v (equivalently, that the node v *produces* the state q) for a given run ρ if $\rho(v) = q$.

A *pointer* is a pair (q, i), where q is a state and i is a number. We write $q[i]$ instead of (q, i). For a given s-dag G and a run ρ, we say that the pointer $q[i]$ points to the i-th (according to the order $\prec$) node producing the state q. A *transition with pointers* is an expression of the form $f(q_1[i_1], \ldots, q_{k-1}[i_{k-1}], \underline{q_k}, q_{k+1}[i_{k+1}], \ldots, q_n[i_n]) \to q$. We say that such a transition is compatible with the transition $f(q_1, \ldots, q_n) \to q$ and the kth position.

For a given node v, its kth successor v_k and a transition with pointers $\tau = f(q_1[i_1], \ldots, \underline{q_k}, \ldots, q_n[i_n]) \to q$, we will often follow the pointers to access the nodes they point to. To do this, we introduce the notion of *dereference* of τ with respect to v, which is the sequence $\langle f, v_1, \ldots, v_n \rangle$ where for $j \neq k$ the pointer $q_j[i_j]$ points to v_j. If v is a leaf, then the dereference of a transition $a \to q$ wrt. v is $\langle a \rangle$.

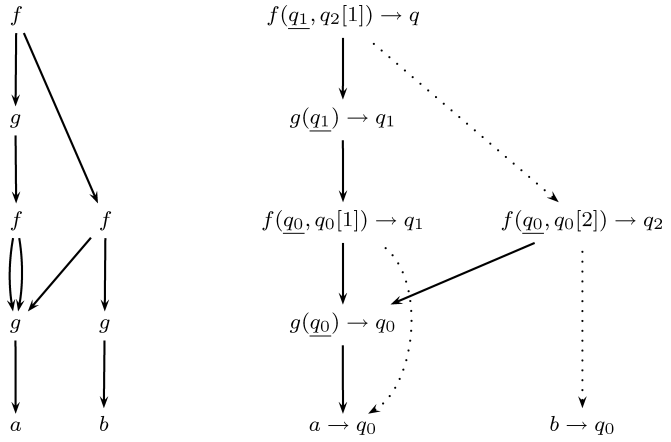

FIG. 4. A t-dag and a skeleton.

Definition 5.3 (*Skeleton*). A skeleton of a run ρ of an automaton $\mathcal{A}$ on an s-dag G is a subgraph G' of G such that

(1) G' contains all roots of G, and
(2) each node of G' that is not a leaf in G has exactly one successor in G', and
(3) each node v in G' is labeled with a transition with pointers, compatible with the transition used by $\mathcal{A}$ to reach v and the position of the successor of v in G', and
(4) if a pointer $q[i]$ is used in a transition labeling a node v then $q[i]$ points to some node v' such that there is no path from v' to v in G', and
(5) G' does not contain two different nodes v, v' labeled with respectively τ, τ' such that the dereference of τ with respect to v is the same as dereference of τ' with respect to v'.

The *width* of a skeleton G' is the maximal number of nodes of the same depth in G'.

Note that a skeleton of a given run is not uniquely defined—there may be more than one transition with pointers that is compatible with a given transition. In the following, we will be interested in skeletons that induce small graphs, so we want to use as small as possible numbers in pointers.

Example 5.4. Figure 4 shows an example of a t-dag G accepted by the automaton from Example 5.2 (this time we view it as a t-dag automaton) and a skeleton of a successful run on G. The skeleton has width 2 and is built only of solid edges; the dotted edges represent pointers which are not included in the skeleton. Item 5.3 in Definition 5.3 is a requirement that the graph consisting of both solid and dotted edges is acyclic. Item 5.3 of this definition requires that the corresponding t-dag (formalized in Definition 5.5 below) does not contain different isomorphic subgraphs. Note that a skeleton does not need to be a connected graph (the node labeled $b \to q_0$ is isolated).

Definition 5.5 (*Induced Graph*). A graph induced from a skeleton S is a directed graph G such that

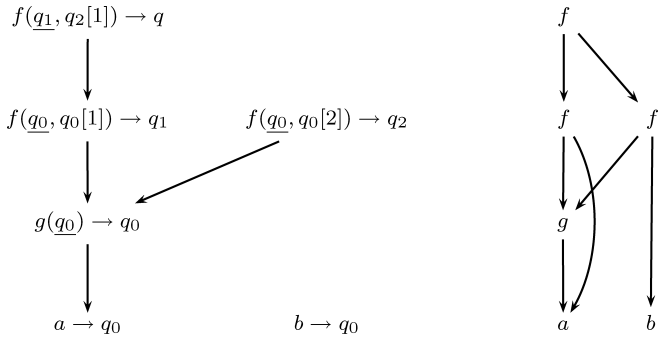

FIG. 5. A pumped-out skeleton and the induced t-dag .

—the set of nodes in G is the same as the set of nodes in S, and
—if a node v is labeled with $f(q_1[i_1], \ldots, \underline{q_k}, \ldots, q_n[i_n]) \to q$ in S, then
  —v is labeled with f in G, and
  —the kth successor of v in G is the successor of v in S, and
  —for $j = 1, \ldots, k-1, k+1, \ldots, n$, the jth successor of v in G is the i_j-th node (according to the ordering $\prec$) of S producing the state q_j.

Figure 5 shows an example of a graph induced from a skeleton.

LEMMA 5.6. *A graph induced from a skeleton is a s-dag.*

PROOF. Let G be a graph induced from a skeleton S. By condition 4 in Definition 5.3, the induced graph is always acyclic.

To prove that G is a s-dag , we have to show that it does not contain two closed isomorphic subgraphs. If G violates this condition, then there exist two different nodes v, v' in G such that the subgraphs of G rooted at these nodes are isomorphic. Assume that v and v' are nodes of minimal depth with this property. Then, they are labeled in G with the same function symbol, and their respective successors are isomorphic (and thus equal, since v and v' are of minimal depth). This means that the dereferences of the labels of v and v' in S wrt. these nodes are equal, which violates the condition 5.3 from Definition 5.3 and leads to contradiction. □

5.3. EMPTINESS. Let us fix an automaton $\mathcal{A} = \langle \Sigma, Q, \Delta, F \rangle$, a t-dag G and a successful run ρ of $\mathcal{A}$ on G. To avoid unnecessary technical details, we assume here that Σ does not contain symbols of arity greater than 2. The extension to arbitrary function symbols is straightforward and presented in Charatonik [1999], in particular the bounds from Theorem 5.7 and Claim 5.10 remain the same.

THEOREM 5.7. *If an automaton $\mathcal{A} = \langle \Sigma, Q, \Delta, F \rangle$ accepts a t-dag , then there exists another t-dag with at most $2|Q|^3$ nodes, accepted by $\mathcal{A}$.*

Let ρ be a successful run of $\mathcal{A}$ on a t-dag G. The idea of the proof is quite simple: first we prove that there exists a skeleton of ρ. If this skeleton is too big, then we "pump it out". We obtain the other t-dag as the graph induced from the pumped-out skeleton. Figures 4 and 5 give an example of this procedure. Figure 4 shows an example of a t-dag accepted by the automaton from Example 5.2 (this time as an automaton over t-dags, not trees) and a skeleton of a successful run ρ on

this t-dag. Figure 5 shows the same skeleton after pumping it out, and the induced t-dag, accepted by the same automaton.

Construction of the skeleton S. We define S as the smallest graph satisfying the conditions

—S contains the root of G,
—for each node v in S, the main successor of v in G is in S,
—for each fork of degree m and color q (see Definition 5.8 below) the nodes pointed by $q[1], \ldots, q[m]$ are in S, and
—S is labeled according to the fork labeling below.

Note that this is a fixed point construction: we start with the root of G and successively add main successors and pointed nodes; the fixed point is reached in finite number of steps because S is a subgraph of G.

A *fork* of degree $m \geq 1$ in a skeleton S is a subgraph of S consisting of a node and its m different predecessors such that the left-hand sides of the corresponding transitions of the automaton (i.e., the transitions compatible with the labels) and the positions of the main successor are the same for all predecessors. The color of the fork is the state used on the left-hand side of the transitions, but not on the main position. On Figure 4, there is one fork of degree 2 and color q_0 (the color used in pointers $q_0[1]$ and $q_0[2]$).

Definition 5.8 (*Fork, Fork Labeling*). We say that $\langle v, v_1, \ldots, v_m \rangle$ is a fork of degree $m \geq 1$ and color q' in a skeleton S of a run ρ on a t-dag G if $v_1, \ldots, v_m$ are predecessors of v in S with the following properties:

—$v_1 \prec \ldots \prec v_m$, and
—for each $i = 1, \ldots, m$, v is the main successor of v_i, and
—the main position for $v_1, \ldots, v_m$ is the same
—all nodes v_i, for $i = 1, \ldots, m$ are labeled in G with the same binary function symbol f, and have two successors $\langle v, v_i' \rangle$ (if the main position is 1) or respectively, $\langle v_i', v \rangle$ (if the main position is 2), and
—all nodes v_i', for $i = 1, \ldots, m$, produce the same state q', that is, $\rho(v_1') = \cdots = \rho(v_m') = q'$

The *fork labeling* is that the predecessor v_i is labeled with $f(\underline{q}, q'[i]) \to \rho(v_i)$ (respectively, $f(q'[i], \underline{q}) \to \rho(v_i)$), where q is the state produced by v.

CLAIM 5.9. *S is a skeleton.*

PROOF. We have to prove that pointers do not introduce cycles and that there are no two nodes with the same dereferences (the last two conditions in Definition 5.3). For the first of them, note that in the definition of S a pointer $q'[i]$ used in a label of v_i points to the ith (according to the order $\prec$) node producing the state q'. This node precedes (or is equal to) the node v_i' which lies below v_i. Therefore, the node pointed by $q'[i]$ lies strictly below v_i; since all edges in S go downwards, there is no path from this node to v_i.

For the second condition, suppose that there is a pair v_1, v_2 of nodes with the same dereferences. Since the main successors of v_1 and v_2 are the maximal according to $\prec$ nodes in these dereferences, v_1 and v_2 must have the same main successor

v. If the main successor occurs on the same position, then we have a fork and by construction the labels used at v_1 and v_2 in S are different and the dereferences cannot be equal. So v occurs on two different positions in the two dereferences. But then the considered dereference is $\langle f, v, v\rangle$ and the nodes v_1 and v_2 are labeled in S with transitions of the form $f(\underline{q_1}, q_2[i]) \to q$ and $f(q_1[j], \underline{q_2}) \to q'$. Suppose v_2 is labeled with the second one. By the definition of the main position, the first successor of v_2 strictly precedes the node v in the ordering $\prec$. Hence, also the node pointed by $q_1[j]$ strictly precedes v and thus it is different from v, which is a contradiction. □

A node v in S is called a *milestone* if it is the root of G or there exist a pointer $q[i]$ used in a label in S, pointing to v. The index of a milestone pointed by a pointer $q[i]$ is the number i; the index of the root is 1. For example, on Figure 4, we have four milestones: the root and the three nodes on ends of dotted edges. Three of them have index 1 and one (the one labeled $b \to q_0$) has index 2.

CLAIM 5.10. *S contains at most $|Q|^2$ milestones.*

PROOF. First, note that each node in S is either a milestone of index 1 (this includes the root of G), or a milestone of index greater than 1, or lies on a main path for some milestone.

To obtain different labels for all predecessors in a fork of degree m, we need at most m milestones producing given state. Therefore, a fork of degree m requires at most $m - 1$ milestones of index greater than 1.

Observe that all predecessors of a fork are nodes at the same depth, so the number of such nodes is bounded by the width of S. Therefore, the degree of a fork cannot exceed the width of S.

Now starting at the root of S and moving towards the leaves observe the number of nodes at the same depth. There are at most $|Q|$ milestones of index 1 (one per state). Hence, for each d there are at most $|Q|$ nodes of depth d lying on main paths for milestones of index 1. A milestone of index greater than 1 is introduced only to satisfy the condition of the fork labeling for some fork. A fork of degree m at depth d gives exactly one successor of m nodes of depth $d + 1$ in S, and thus it decreases the width of S at depth d and below by $m - 1$; on the other hand it may introduce at most $m - 1$ milestones of index greater than 1 (and below them the nodes on main paths for them) at depth d or below, and thus it may increase the width of S at depth d or below by at most $m - 1$. Hence, the width at each depth is bounded by $|Q|$.

Since the degree of every fork is bounded by the width of S (which is bounded by $|Q|$), each fork requires at most $|Q| - 1$ milestones of index greater than 1. Thus the maximal index of a milestone does not exceed $|Q|$ and there are at most $|Q|^2$ milestones. □

A node that is neither milestone nor have more than one predecessor in S is called *ordinary*. Ordinary nodes have only one predecessor in the graph induced from S, which is already present in S. This is why we do not get the problem of "hanging edges" in the pumping lemma.

CLAIM 5.11 (PUMPING LEMMA). *Let the skeleton S contain a path $v_0 \to v_1 \to \cdots \to v_m$ such that*

—the states produced by v_1 and v_m are the same, and

—all the nodes $v_1, \ldots, v_m$ are ordinary.

If S' is a graph obtained from S by removing the nodes $v_1, \ldots, v_{m-1}$ and defining v_m as the successor of v_0, then the graph induced from S' is a t-dag accepted by the automaton $\mathcal{A}$.

PROOF. We have to show that G' is a t-dag and that $\mathcal{A}$ accepts it. The latter thing is quite simple: the mapping assigning to each node of G' the state produced by this node in S' is a successful run. Hence, it is enough to show that G' is a t-dag.

Suppose G' contains two different closed isomorphic subgraphs G'_1 and G'_2. If none of them contain the node v_m, then both G'_1 and G'_2 are closed subgraphs of G, which contradicts the fact that G is a t-dag. Suppose G'_1 contains v_m. Then, G'_2 contains an isomorphic copy of the subgraph rooted at v_m, which must be a subgraph of G (the differences between G and G' start on the level above v_m). Since the only closed subgraph of G isomorphic to the subgraph rooted at v_m is the subgraph rooted at v_m, G'_2 also contains v_m. Hence the difference between G'_1 and G'_2 must occur somewhere above v_m. Since v_m is a node with only one predecessor v_0 in S' (and thus in G', too), both G'_1 and G'_2 contain v_0. Now let G_1 and G_2 be the two graphs obtained from G'_1 and G'_2 by replacing the edge $v_0 \to v_m$ with the path $v_0 \to v_1 \to \cdots \to v_m$ together with all induced edges Now the isomorphism between G'_1 and G'_2 can be extended to an isomorphism between G_1 and G_2, which contradicts the fact that G is a t-dag. □

PROOF OF THEOREM 5.7. By repeated applications of the procedure above, we can construct a t-dag H accepted by the automaton $\mathcal{A}$ such that the skeleton S_H constructed for H does not contain any path whose nodes are ordinary and two of them produce the same state. Thus, every path consisting of ordinary nodes has the length bounded by $|Q|$. Every maximal path of this form has a unique predecessor, which is not ordinary.

Now we estimate the number of non-ordinary nodes in S. It is bounded by the number of milestones plus the number of nodes with more than one predecessor in S_H. The nodes with more than one predecessor are nodes in S_H with indegree (the number of incoming edges) greater than their outdegree (the number of outgoing edges). On the other hand, the only nodes with outdegree greater than indegree are the roots of S_H (i.e., the nodes without predecessors in S_H), which are milestones. Since in every directed graph the sum of indegrees of all nodes equals to the sum of outdegrees of all nodes, the number of nodes that have more than one predecessor in S_H is bounded by the number of milestones. Therefore, there are at most $2|Q|^2$ nodes which are not ordinary.

Finally, the number of nodes in S_H (equal to the number of nodes in H) is bounded by the number of nonordinary nodes times the maximal length of a path of ordinary nodes, which is bounded by $2|Q|^3$. □

COROLLARY 5.12. *The emptiness problem for t-dag automata is decidable in NP.*

PROOF. This is a direct consequence of Theorem 5.7: it is enough to guess a t-dag of size at most $2|Q|^3$ where Q is the set of states, and a successful run of the automaton on this t-dag. □

The emptiness problem for t-dag automata is NP-complete. The lower bound is proved in Charatonik [1999]; we do not present it here because it has no direct consequences in solving set constraints.

5.4. AUTOMATA WITH PROJECTIONS: EMPTINESS. From now on, to simplify notations, we assume that all function symbols occurring in requests $\langle f, i, R\rangle$ are binary (and thus $i \in \{1, 2\}$). In light of Lemma 2.1, Theorem 4.8, and Lemma 4.7, this restriction does not influence the correctness of the satisfiability test for set constraints.

Recall that, in Theorem 4.8, we reduced the problem of satisfiability of set constraints with projections to the problem of existence of a faithful run of some automaton on the (infinite) graph representing Herbrand universe. Below, we show that such infinite run can be reconstructed from a finite piece of information that we call a germ.

Definition 5.13 (*Germ*). A germ for an automaton $\mathcal{A} = \langle \Sigma, Q, \Delta, \pi\rangle$ is a tuple $\langle G, \rho\rangle$ such that

—G is a finite s-dag with the set V of nodes and $V_r \subseteq V$ of roots, and
—ρ is a faithful up to V_r run of $\mathcal{A}$ on G, and
—$\mathcal{A}$ restricted to the states in $\rho(V - V_r)$ is complete, and
—$\rho(V - V_r) = \rho(V)$.

LEMMA 5.14. *If there exists a germ for the automaton $\mathcal{A}$, then $\mathcal{A}$ admits a faithful run on the s-dag representation G_Σ of the Herbrand universe.*

PROOF. Let $\langle G, \rho\rangle$ be a germ for the automaton $\mathcal{A}$. For each state $q \in \rho(V - V_r)$ we fix a term t_q represented by a node in $V - V_r$ such that $\rho(t_q) = q$ (recall that we identify nodes with the terms they represent). For each state $q \in \rho(V - V_r)$ and each request $\langle f, i, R\rangle \in \pi(q)$ we fix a term $w_{q,f,i,R}$ (a witness for the term t_q and the request $\langle f, i, R\rangle$) such that $\rho(f(t_q, w_{q,f,i,R})) \in R$ (if $i = 1$; otherwise, if $i = 2$, we require $\rho(f(w_{q,f,i,R}, t_q)) \in R$). Now we inductively define an extension of ρ to a faithful run. The idea is to reuse the chosen witnesses to grant all requests, and to proceed in any fixed way with all other terms.

Since constant and unary symbols do not contribute to faithfulness of a run, we can use any transition from Δ to pass across these symbols. Suppose that $\rho(t)$ is defined (otherwise consider strict subterms of t first) and f is a binary symbol in Σ. Now we define ρ on all terms of the form $f(t, t')$ and $f(t', t)$, where t' is a term such that $\rho(t')$ is already defined but $\rho(f(t, t'))$ (or, respectively, $\rho(f(t', t))$) is not yet defined. We do it as follows.

—for $t' = w_{\rho(t),f,1,R}$ we put $\rho(f(t, t')) = \rho(f(t_{\rho(t)}, t'))$
—for $t' = w_{\rho(t),f,2,R}$ and $t \neq w_{\rho(t),,f,1,R}$ we put $\rho(f(t', t)) = \rho(f(t', t_{\rho(t)}))$
—for all other terms t', we put $\rho(f(t, t')) = \rho(f(t_{\rho(t)}, t_{\rho(t')}))$ and $\rho(f(t', t)) = \rho(f(t_{\rho(t')}, t_{\rho(t)}))$.

To see that ρ is a faithful run on G_Σ, note that for all terms $t \in T_\Sigma - (V - V_r)$ and any request $\langle f, i, R\rangle \in \pi(\rho(t))$, there exists a term t', namely $t' = f(t, w_{\rho(t),f,i,R})$ or respectively $t' = f(w_{\rho(t),f,i,R}, t)$, such that $\rho(t') \in R$ and t is the ith successor of t'. For $t \in V - V_r$, the definition of a germ implies that the requests are granted. □

LEMMA 5.15. *If $\mathcal{A}$ admits a faithful run, then there exists a germ for $\mathcal{A}$ containing $O(|Q|^5 \cdot P^3)$ nodes, where P is the number of requests occurring in $\pi(Q)$.*

PROOF. This technical lemma is proved in Section 6. The proof is a refinement of the proof of Theorem 5.7. □

Now we are ready to prove the main result.

PROOF OF THEOREM 2.2. In light of Theorem 4.8 and Lemmas 4.7, 5.14, and 5.15, it is enough to guess a germ for the automaton corresponding to the input system of set constraints. The size of the germ is polynomial in the size of the automaton, which is exponential in the size of the input system. □

6. *Proof of Lemma 5.15*

We assume that the automaton $\mathcal{A}$ and its faithful run ρ on G_Σ are fixed.

6.1. NOTATIONS. The general idea of the proof is first to construct a big enough skeleton without labels, second to label it in a consistent way, and third to make the skeleton small enough by pumping it out. Then, such a skeleton induces a desired germ.

By a color of a term t, we mean the state $\rho(t)$ of the automaton $\mathcal{A}$. Note that, in Definition 5.8, the notion of a fork does not depend on the labeling of the skeleton, so it may be applied to graphs without such labeling.

Definition 6.1 (*Naked Skeleton*). A naked skeleton $S(T)$ (of the run ρ of the automaton $\mathcal{A}$) generated by a finite set T of terms is the smallest graph such that

—$S(T)$ is a subgraph of the s-dag G representing T

—$S(T)$ contains all the roots of G

—for each node v in $S(T)$, the main successor of v in G is in $S(T)$, and there is an edge connecting v with its main successor

—for each fork of color q and degree m, $S(T)$ contains the m nodes pointed by $q[1], \ldots, q[m]$.

Terms in the set T together with milestones of index 1 are called *generators* of $S(T)$. All nodes representing terms in T are declared to be milestones.

The main difference between a naked skeleton and a skeleton defined in Chapter 5 is that naked skeleton is not labeled with transitions.

In a fork $\langle t, f(t, s_1), \ldots, f(t, s_m)\rangle$ (or, respectively, $\langle t, f(s_1, t), \ldots, f(s_m, t)\rangle$), the terms $s_1, \ldots, s_m$ are called *used* by this fork (we do not call t used). Sometimes, slightly abusing the notation, we will call the node t itself a fork. Note that in such a case the color of the fork (which is the color of the nodes s_i) is usually different from the color of the node t. By $\mathsf{maxfork}(c, S)$, we denote the maximal degree of a fork of color c in a naked skeleton S.

We call a node t a *small milestone* in S if t is pointed by a pointer $c[i]$ with $i \leq \mathsf{maxfork}(c, S)$. In Section 5, all milestones except the root were small. A *big milestone* is a node that is declared to be a milestone and is not small.

Color c is called *saturated* in a naked skeleton S if all nodes of color c are nodes in S and are milestones.

Let us fix a function ω assigning to every term t a set of representatives of all witnesses for the requests in $\pi(\rho(t))$. That is, if $\pi(\rho(t)) = \{\langle f_1, i_1, R_1\rangle, \ldots, \langle f_k, i_k, R_k\rangle\}$ then $\omega(t) = \{s_1, \ldots, s_k\}$ such that for all $j = 1, \ldots, k$ we have $\rho(f_j(t, s_j)) \in R_j$ or $\rho(f_j(s_j, t)) \in R_j$ depending on whether i_j is 1 or 2. Note that the witnesses s_j do exist since ρ is a faithful run; there may be many different witnesses for the term t and the request $\langle f_j, i_j, R_j\rangle$, and we choose just one representative (it can be, for example, the least witness in accordance with the ordering $\prec$).

Let $K = 7$, $K_1 = (|Q| + 1) \cdot (|Q| + P) + (|Q| + P + 2) \cdot (1 + P)$ and

$$N = \max\left(2K^2 \cdot P \cdot (|Q| + 1) \cdot (|Q| + P) + K \cdot |Q|, \ \frac{K_1 \cdot 4K^2 + P \cdot 2K}{K - 6}\right),$$

where P is the number of requests occurring in $\pi(Q)$ (which, in terms of set constraints, is the number of projections expressions occurring in the initial system) be fixed. We assume here that $P \geq 1$. The numbers K, K_1 and N are chosen in such a way that K, K_1 are constants and N is big enough to make some inequalities of the form $c_1 \cdot N^2 \leq c_2 \cdot N$ impossible, where c_1 and c_2 are some constants that will appear in the construction below. Note that N is bounded by a polynomial in $P, |Q|$.

Observation 6.2. If P and $|Q|$ range over natural numbers and N is defined as above, then $N \in O(P^2 \cdot |Q|^2)$.

We say that a *witness* s is *busy* in S if the color $\rho(s)$ is saturated in S and $s \in \omega(t)$ for at least $\frac{N}{K}$ milestones t. We say that a milestone s is *often used* in S if it is used by at least $\frac{N}{K}$ different forks of the same color; otherwise, it is *seldom used*. We say that a *milestone* is *busy* if it is often used or it is a busy witness.

A fork $\langle t, f(t, s_1), \ldots, f(t, s_m)\rangle$ (or, respectively, $\langle t, f(s_1, t), \ldots, f(s_m, t)\rangle$) of color c is called *tight* in a naked skeleton S if its degree is greater or equal to

$$|\{s \in S \mid \rho(s) = c, \ s \prec t\}| - \frac{N}{2}.$$

Intuitively, a fork of degree m is tight if there is $\leq m + \frac{N}{2}$ terms that could replace the terms $s_1, \ldots, s_m$ in their role of the terms used by the fork.

6.1.1. *Some Properties of Naked Skeletons.* Recall that a width of a skeleton S is the maximal number of nodes of the same depth in S.

LEMMA 6.3. *The maximum degree of a fork in a naked skeleton is bounded by the width of this skeleton.*

PROOF. A fork of degree m has m predecessors, and all of them are at the same depth. □

LEMMA 6.4. *The width of a naked skeleton $S(T)$ is bounded by $|T| + |Q|$.*

PROOF. The reasoning here is the same as in Claim 5.10 in Section 5.

First note that each node in $S(T)$ is either a generator, or a milestone of index greater than 1, or lies on a main path for some milestone.

To obtain in $S(T)$ the nodes pointed by $q[1], \ldots, q[m]$ for a fork of degree m and color q, we need at most m milestones producing given color. Therefore a fork of degree m requires at most $m - 1$ milestones of index greater than 1.

There are at most $|Q| + |T|$ generators. Hence, for each d, there are at most $|Q|$ nodes of depth d lying on main paths for generators. To estimate the width of S, note that a milestone of index greater than 1 is introduced only to satisfy the last condition of Definition 6.1.

A fork of degree m at depth d gives exactly one successor of m nodes of depth $d + 1$ in S, and thus it decreases the width of S at depth d and below by $m - 1$; on the other hand, it may introduce at most $m - 1$ milestones of index greater than 1 (and below them the nodes on main paths for them) at depth d or below, and thus it may increase the width of S at depth d or below by at most $m - 1$. Hence, the width at each depth is bounded by the number of generators. □

LEMMA 6.5. *The number of milestones in a naked skeleton $S(T)$ is bounded by $|Q| \cdot width(S(T)) + |T|$.*

PROOF. Since the degree of every fork is bounded by the width of $S(T)$, each fork requires at most $width(S(T))$ small milestones of a given color. Thus, the number of small milestones in $S(T)$ is bounded by $|Q| \cdot width(S(T))$. All big milestones are members of T. □

LEMMA 6.6. *If s is a small milestone in $S(T)$, then the sets $S(T)$, $S(T \setminus \{s\})$ and $S(T \cup \{s\})$ are equal.*

PROOF. If s is a small milestone of color c, then s is pointed by a pointer $c[i]$ for some $i \leq \mathsf{maxfork}(c, S(T))$. Consider a fork $\langle t, f(t, s_1), \ldots, f(t, s_n)\rangle$ (or respectively $\langle t, f(s_1, t), \ldots, f(s_n, t)\rangle$) of color c with maximal degree, that is, with $n = \mathsf{maxfork}(c, S(T))$. By the definition of the fork, the terms $s_1, \ldots, s_i$ are all of the same color and are smaller than t in the ordering $\prec$, therefore they are on the same depth as t or below t. Moreover, s is smaller or equal to s_i in the ordering $\prec$, hence it is also at the same depth or below t, so for sure it is below the terms $f(t, s_1), \ldots, f(t, s_n)$ (or, respectively $f(s_1, t), \ldots, f(s_n, t)$). Therefore, all these terms, being above s, are in $S(T \setminus \{s\})$. Now the last item of Definition 6.1 implies that s is in $S(T \setminus \{s\})$ so neither adding it to the set of generators nor removing it from this set changes the final skeleton. □

6.2. CONSTRUCTION OF A NAKED SKELETON S. In this section, we construct in an iterative way a naked skeleton S. For all $i \in \mathbb{N}$ we define the set T_i and the naked skeleton $S_i = S(T_i)$ generated by T_i, such that $S_i \subseteq S_{i+1}$. Then, S is a fixpoint of this construction. We say that a color c is reachable in ρ if there exist t such that $\rho(t) = c$.

Let T_0 be the set consisting of (at most) $N \cdot |Q|$ terms, namely for every reachable color $c \in Q$ the first (in accordance with the ordering $\prec$) N terms colored by c.

For $i \geq 0$, let $S_i = S(T_i)$. Let

$$T_i' = T_i - \{t \mid t \text{ is a small milestone in } S_i\} \cup \bigcup \{\omega(t) \mid t \text{ is a busy milestone in } S_i\},$$

Finally, let T_{i+1} be the union of T_i' with so many minimal (not occurring in S_i) terms of each color that T_{i+1} contains at least N elements of each color (or the color becomes saturated in $S(T_{i+1})$). More precisely, for each reachable color c, if there are at least $\mathsf{maxfork}(c, S_i) + N$ terms of color c then we add (if they are not there already) to T_{i+1} all the terms pointed by $c[\mathsf{maxfork}(c, S_i) + 1], \ldots, c[\mathsf{maxfork}(c, S_i) + N]$; otherwise, we add to T_{i+1} all terms of color c.

To prove that this construction reaches a fixpoint and to estimate its size, we will need a couple of properties that we prove by induction on i.

CLAIM 6.7. *The number of elements in T_i is bounded by $N \cdot (|Q| + P) - |Q|$.*

CLAIM 6.8. *The width of S_i is bounded by $N \cdot (|Q| + P)$.*

CLAIM 6.9. *The number of milestones in S_i is bounded by $N \cdot (|Q|+1) \cdot (|Q|+P)$.*

CLAIM 6.10.

$$\sum_v (\deg(v) - 1) \leq N \cdot (|Q| + 1) \cdot (|Q| + P).$$

where v ranges over all forks in S_i.

CLAIM 6.11. *The number of often used milestones in S_i is bounded by $\frac{N}{K}$.*

CLAIM 6.12. *The number of busy witnesses is bounded by $\frac{N}{K}$.*

6.2.1. *Proof of Claims* 6.7–6.12. We first prove the induction basis, which is Claim 6.7 for $i = 0$. Then, in the induction step, we prove that Claim 6.7 implies Claims 6.8–6.12, and these claims together imply Claim 6.7 with $i + 1$ substituted for i.

Induction Basis. The size of T_0 is bounded by $N \cdot |Q|$, which is less than $N \cdot (|Q| + P) - |Q|$ since $N \cdot P > |Q|$.

Induction Step. Suppose that $|T_i| \leq N \cdot (|Q| + P) - |Q|$.

PROOF OF CLAIM 6.8. This claim is a direct consequence of Claim 6.7 and Lemma 6.4. □

PROOF OF CLAIM 6.9. By Lemma 6.5 and Claim 6.8, the number of milestones in S_i is bounded by $|Q| \cdot N \cdot (|Q| + P) + N \cdot (|Q| + P) - |Q|$ which is less than $N \cdot (|Q| + 1) \cdot (|Q| + P)$. □

PROOF OF CLAIM 6.10. Recall that a fork of degree m in S_i at depth d gives exactly one successor of m nodes at depth $d+1$ in S_i, it "consumes" $m-1$ paths in S_i. Thus, the sum $\sum_v (\deg(v) - 1)$ is bounded by the number of roots in S_i. Since every root in a naked skeleton is a milestone, the claim follows from Claim 6.9 □

PROOF OF CLAIM 6.11. First, note that the number of uses of terms in forks is bounded by the sum of degrees of these forks. Then, from Claim 6.10 and the observation that the number of forks of degree at least 2 is also bounded by the number of roots in S_i, we have

$$\sum_v \deg(v) \leq 2N \cdot (|Q| + 1) \cdot (|Q| + P)$$

where v ranges over forks of degree at least 2.

To estimate the number of often-used milestones, suppose that this number exceeds $\frac{N}{K}$. Since there are at most $|Q|$ milestones of index 1, there are at least $\frac{N}{K} - |Q|$ often used milestones of index greater then 1. Every such milestone is used in at least $\frac{N}{K}$ forks of degree at least 2. Therefore, the number of uses of terms

in forks exceeds $(\frac{N}{K} - |Q|) \cdot \frac{N}{K}$, which should be bounded by the sum of degrees of forks given above. We have chosen N big enough to make $(\frac{N}{K} - |Q|) \cdot \frac{N}{K} \leq 2N \cdot (|Q| + 1) \cdot (|Q| + P)$ a contradiction $\square$

PROOF OF CLAIM 6.12. Consider the set

$$\{\langle w, t\rangle \mid w \in \omega(t),\ t \text{ is a milestone}\}.$$

Since $|\omega(t)| \leq P$, the cardinality of this set is bounded by P times the number of milestones in S_i (bounded by $N \cdot (|Q| + 1) \cdot (|Q| + P)$). On the other hand, if we denote by B the number of busy witnesses, then this set has at least B times $\frac{N}{K}$ elements. Therefore, $B \cdot \frac{N}{K} \leq P \cdot N \cdot (|Q| + 1) \cdot (|Q| + P)$ and

$$B \leq P \cdot N \cdot (|Q|+1) \cdot (|Q|+P) \cdot \frac{K}{N} \leq \frac{K^2 \cdot P \cdot (|Q| + 1) \cdot (|Q| + P)}{K} < \frac{N}{K}. \quad \square$$

PROOF OF CLAIM 6.7. The set T_{i+1} contains at most $|Q| \cdot N$ elements plus all the elements in $\bigcup\{\omega(t) \mid t \text{ is a busy milestone in } S_i\}$. By Claims 6.11 and 6.12, there are at most $\frac{2N}{K}$ busy milestones in S_i. Therefore, $|T_{i+1}| \leq |Q| \cdot N + P \cdot \frac{2N}{K} \leq N \cdot (|Q| + P) - \frac{K-2}{K} \cdot P \cdot N < N \cdot (|Q| + P) - |Q|$ since $N > K \cdot |Q|$ and $K > 3$ and $P \geq 1$. $\square$

PROPOSITION 6.13. *There exists an index i such that $S_i = S_{i+1} = S$.*

PROOF. Let M_i for $i > 0$ be the set of small milestones in S_{i-1}. By Lemma 6.6, we have that $S_i = S(M_i \cup T_i)$. Observe that for all $i > 0$, the set $M_i \cup T_i$ is a subset of $M_{i+1} \cup T_{i+1}$. By Claims 6.9 and 6.7, the size of $M_i \cup T_i$ is bounded, so the construction must reach a fixed point before the size of $M_i \cup T_i$ reaches its bound. $\square$

6.3. PROPERTIES OF THE NAKED SKELETON S.

LEMMA 6.14. *Let S be the naked skeleton constructed above. Then*

(1) *For every color c in S, either c is saturated or all nodes pointed by $c[1], \ldots, c[\mathsf{maxfork}(c, S) + N]$ are milestones in S,*
(2) *for every busy milestone t, every member of the set $\omega(t)$ is a milestone in S,*
(3) *for every color c the number of milestones of color c is bounded by the number $\mathsf{maxfork}(c, S) + N + \frac{2N}{K}$,*
(4) *for every nonsaturated color d, there are at least $\mathsf{maxfork}(d, S) + N - \frac{2N}{K}$ nonbusy milestones of color d*

PROOF. Let T be the set T_i such that $S = S(T_i)$. For the proof of 1, suppose that c is not saturated. Then the nodes pointed by $c[1], \ldots, c[\mathsf{maxfork}(c, S)]$ are small milestones and the nodes pointed by $c[\mathsf{maxfork}(c, S) + 1], \ldots, c[\mathsf{maxfork}(c, S) + N]$ are members of T and are declared to be milestones. Point 2 follows from the observation that members of $\omega(t)$ are in T and thus are declared to be milestones. For point 3, every milestone of color c is either pointed by $c[i]$ for some $i \leq \mathsf{maxfork}(c, S) + N$ or is a busy milestone; the number of busy milestones by Claims 6.11 and 6.12, is bounded by $\frac{2N}{K}$. The last point also follows directly from point 6.14 and the bound on the number of busy milestones. $\square$

LEMMA 6.15. *Let C be the set of all milestones of a given color c, and let K_1 be the constant defined in Section* 6.1. *Then*

$$\sum_{v \in C} (\deg(v) + P) \leq K_1 \cdot N.$$

PROOF. From Lemma 6.14 (point 3), Lemma 6.3, and Claim 6.8, we know that

$$|C| \leq N \cdot (|Q| + P) + N + \frac{2N}{K}.$$

From Claim 6.10, we know

$$\sum_{v \in C} (\deg(v) - 1) \leq N \cdot (|Q| + 1) \cdot (|Q| + P).$$

Therefore

$$\begin{array}{ll} & \sum_{v \in C} (\deg(v) + P) \\ \leq & \sum_{v \in C} (\deg(v) - 1) + (P + 1) \cdot |C| \\ \leq & N \cdot (|Q| + 1) \cdot (|Q| + P) + (P + 1) \cdot (N \cdot (|Q| + P) + N + \frac{2N}{K}) \\ \leq & K_1 \cdot N \end{array}$$

□

LEMMA 6.16. *Let $\langle t, t_1, \ldots, t_m \rangle$ be a tight fork of color c in S. Then, every node s of color c such that $s \prec t$ is a milestone.*

PROOF. If the color c is saturated, then every node of color c is a milestone and there is nothing to be proved. So suppose c is not saturated. By Lemma 6.14 (point 1), all minimal $N + \mathsf{maxfork}(c, S)$ nodes of color c are milestones. By the definition of a tight fork, the number n of nodes of color c preceding t in the ordering $\prec$ is bounded by

$$n \leq m + N/2 \leq \mathsf{maxfork}(c, S) + N/2 \leq \mathsf{maxfork}(c, S) + N,$$

hence, every such node is a milestone. □

6.4. A LITTLE BIT OF COMBINATORICS ON BIPARTITE GRAPHS. For the labeling construction in the next section, we will need some combinatorics on bipartite graphs. The finite sets $C = \{c_1, \ldots\}$ and $D = \{d_1, \ldots\}$ are sets of all nonbusy milestones of color c and d respectively. We will also assume that the color d is not saturated and that f is a fixed function symbol in our signature.

Consider a directed bipartite graph $\langle C, D, E \rangle$ with the two sets of nodes C and D and the edge relation $E \subseteq C \times D \cup D \times C$. Both edges $\langle c_i, d_j \rangle$ and $\langle d_j, c_i \rangle$ in E represent the term $f(t, s)$ where t, s are respectively terms represented by the nodes c_i and d_j in the s-dag representation of the set T (in a symmetric context both these edges represent $f(s, t)$). Imagine that all edges in E are labeled with some color, intuitively it should be the color of the term represented by the edge: $\rho(f(t, s))$. We will say that there is a conflict in the graph if there exists a pair of edges $\langle c_i, d_j \rangle$ and $\langle d_j, c_i \rangle$ in E labeled with two different colors.

In the rest of this section, we will prove that in some case (the precise definition of this case will be given in the next section) it is possible to define the graph $\langle C, D, E \rangle$ in such a way that every node has a desired outdegree (which is necessary to induce the labeling of the skeleton from the graph) and there are no conflicts in the graph.

The intuition behind the proof is very simple: we need only linearly many in N edges in the graph while there is quadratically many possible edges, so there

is enough room to avoid conflicts. The details are however more complicated, we present them below.

For a given edge relation $E \subseteq C \times D \cup D \times C$, we define $indeg(v, E) = |\{u \mid \langle u, v\rangle \in E\}|$ and $outdeg(v, E) = |\{u \mid \langle v, u\rangle \in E\}|$. By $indeg(D, E)$, we mean $\max\{indeg(v, E) \mid v \in D\}$.

PROPOSITION 6.17. *Let $\langle C, D, E_0, A, m, p\rangle$ be such that*

—*C and D are the sets of non-busy nodes of color c and d, respectively*
—$|D| \geq \mathsf{maxfork}(d, S) + N - \frac{2N}{K}$
—$E_0 \subseteq C \times D \cup D \times C$
—$indeg(D, E_0) \leq \frac{2N}{K}$
—$A : C \to 2^D$ *is a function assigning to every member of C a set of preferred nodes in D*
—$m : C \to \mathbb{N}$ *and* $p : C \to \mathbb{N}$ *are functions assigning numbers to elements of C such that*
  —$\sum_{v \in C} m(v) + p(v) \leq K_1 \cdot N$
  —*for all* $v \in C$, $m(v) \leq \mathsf{maxfork}(d, S)$
  —*for all* $v \in C$, $p(v) \leq P$
—*for all* $v \in C$, $outdeg(v, E_0) = m(v)$ *or* $outdeg(v, E_0) = 0$
—*for all* $v \in C$, *if* $outdeg(v, E_0) = m(v)$ *then* $A(v) \supseteq \{u \mid \langle v, u\rangle \in E_0\}$,
—*for all* $v \in C$, *if* $outdeg(v, E_0) = 0$ *then* $|A(v)| \geq m(v) + \frac{N}{2} - \frac{N}{K}$.

Then there exists $E \supseteq E_0$ such that

—$indeg(D, E) \leq indeg(D, E_0) + \frac{N}{K}$
—*for all v in C,* $outdeg(v, E) = m(v) + p(v)$
—*for all v in C, v is connected with at least $m(v)$ preferred nodes:*

$$|\{u \ : \ \langle v, u\rangle \in E\} \cap A(v)| \geq m(v),$$

no conflicts are created in E: if $\langle u, v\rangle \notin E_0$ then either $\langle u, v\rangle \notin E$ or $\langle v, u\rangle \notin E$

PROOF. We construct iteratively for $i = 1, \ldots, |C|$ the edge relation by adding to E_{i-1} edges outgoing from the node c_i. Then we define $E = E_{|C|}$.

To define E_i, we first take the $m(c_i)$ nodes (if outdegree of c_i in E_0 is 0; otherwise, we skip this step) with the smallest indegree from the set $A(c_i) - \{v \mid \langle v, c_i\rangle \in E_0\}$. Then, we take $p(c_i)$ nodes with the smallest indegree from $D - \{v \mid \langle v, c_i\rangle \in E_0\}$ that were not chosen in the first step. Finally, we connect c_i to all chosen nodes.

In the first step, due to the conditions on the function A and $indeg(D, E_0)$, we have choice of at least $m(c_i) + \frac{N}{2} - \frac{N}{K} - \frac{2N}{K}$ elements. In the second step, due to the condition on the cardinality of D, we still have choice of at least $N - \frac{2N}{K}$ elements, which, by the definition of N, is more than P and thus more than $p(c_i)$. Hence E_i is well defined.

By construction, all conditions required by the proposition, except $indeg(D, E) \leq indeg(D, E_0) + \frac{N}{K}$, are trivially satisfied. To see the inequality above, suppose that there exists a node $v \in D$ such that $indeg(v, E) > indeg(D, E_0) + \frac{N}{K}$. While choosing v to be connected to some node in C, we left at least $\frac{N}{2} - \frac{N}{K} - \frac{2N}{K} - P$ elements with indegree at least $indeg(D, E_0) + \frac{N}{K}$ which is more than $\frac{N}{K}$. Therefore,

the number of edges in E exceeds $\frac{N}{K} \cdot (\frac{N}{2} - \frac{N}{K} - \frac{2N}{K} - P)$ which is impossible since the number of edges is bounded by $K_1 \cdot N$: the second argument of the function max in the definition of N was chosen to make this inequality a contradiction. □

6.5. CONSTRUCTION OF A PRE-GERM. In this section, we extend the naked skeleton from Section 6.2 by defining a labeling that will allow us reconstructing a faithful run of the automaton.

Definition 6.18 (*Pre-Germ*). A pre-germ G is a naked skeleton with additional labeling

(1) every node v in G is labeled with a transition with pointers, such that G is a skeleton according to Definition 5.3,
(2) for every node v in G and every request $\langle f, 1, R\rangle \in \pi(\rho(v))$, v is labeled with $f(*, q[i]) : q'$ for some q, q' and i such that $q' \in R$, and
(3) for every node v in G and every projection expression $\langle f, 2, R\rangle \in \pi(\rho(v))$, v is labeled with $f(q[i], *) : q'$ for some q, q' and i such that $q' \in R$.

We refer to labels in point (1) as to *transition* labels and to labels in points (2) and (3) as *request* labels.

Definition 6.19 (*Consistent Labeling*).

(1) We say that a labeling in a pre-germ G is *locally consistent* if for every node t in G, if t is labeled $f(*, d[i]) : q$ and some predecessor of t is labeled $f(\underline{c}, d[i]) \to q'$ (respectively, if t is labeled $f(d[i], *) : q$ and its predecessor labeled $f(d[i], \underline{c}) \to q'$), then $q = q'$.
(2) The labeling of G is *globally consistent* if for all pairs s, t of nodes, if
—s has a predecessor labeled $f(\underline{c}, d[i]) \to q$ or s is labeled $f(*, d[i]) : q$,
—$d[i]$ points to t,
—t has a predecessor labeled $f(c[j], \underline{d}) \to q'$ or t is labeled $f(c[j], *) : q'$, and
—$c[j]$ points to s
then $q = q'$.
(3) We call a pre-germ G consistent if its labeling is both locally and globally consistent.

Now we start constructing a consistent labeling of the naked skeleton S defined in Section 6.2. Every transition label for a node t is of the form $f(\underline{c}, d[i]) \to \rho(t)$ or $f(d[i], \underline{c}) \to \rho(t)$ where f is the function symbol occurring in the node t in the s-dag representing the Herbrand universe, and c is the color of the main successor of t. In particular, there are labels $a \to \rho(a)$ for all leaves a in S.

Now, for every node t in S we define the transition labels of the form $f(\ldots) \to c$ for all predecessors of t and the request labels of the form $f(\ldots) : c$ for t. We do it depending on the type of the node t and the color d.

Case 1. *t is busy or d is saturated.* Let t be a node, with $\rho(t) = c$. If t a busy milestone, or it is a fork of color d (note that every node that is not a root is a fork of degree ≥ 1) with d being saturated, or some of the nodes in $\omega(t)$ are of such color, we induce the labels from the original run: The predecessor $f(t, s)$, (or respectively,

$f(s,t)$) (note that every node in S is also a node in the s-dag G_Σ representing the Herbrand universe) where $\rho(s) = d$ is labeled $f(\underline{c}, d[j]) \to \rho(f(t,s))$ (or respectively, $f(d[j], \underline{c}) \to \rho(f(s,t))$), where $d[j]$ is a pointer pointing to s. Similarly, for a request $\langle f, i, R\rangle \in \pi(c)$, if the witness s for t and this request is of saturated color d, then t is labeled $f(*, d[j]) : \rho(f(t,s))$ or $f(d[j], *) : \rho(f(s,t))$ where $d[j]$ is the pointer pointing to s.

Case 2. *t is not a milestone, d is not saturated.* For colors that are not saturated, the labeling is easier: in a fork $\langle t, f(t,s_1), \ldots, f(t,s_m)\rangle$ of color d the predecessor $f(t,s_j)$ is labeled $f(\underline{c}, d[j]) \to \rho(f(t,s_j))$ (in a fork where t is the right successor it is done symmetrically). If $\omega(t)$ contains k terms $w_1, \ldots, w_k$ of color d (we repeat this for all such colors d), then t is labeled with the k labels $f(*, d[\mathsf{maxfork}(c,S)+j]) : \rho(f(t,w_j))$ (or symmetrically if the respective request is $\langle f, 2, R\rangle$). Note that in a color that is not saturated we have N big milestones; since $N > P$, we can use milestones pointed by $d[\mathsf{maxfork}(c,S)+j]$ with $j \le P$.

Case 3. *t is a nonbusy milestone, d not saturated.* Let $\rho(t) = c$. We define the labeling for all non-busy milestones of color c at the same time, and if c is not saturated, we define also the labeling for nonbusy milestones of color d. The labeling is constructed for each kind of fork separately, where the kind of a fork $\langle t, f(t,s_1), \ldots, f(t,s_m)\rangle$ is the tuple $\langle f, \mathit{left}, \rho(s_1)\rangle$ and the kind of a fork $\langle t, f(s_1,t), \ldots, f(s_m,t)\rangle$ is the tuple $\langle f, \mathit{right}, \rho(s_1)\rangle$.

Let us fix the kind $\langle f, \mathit{left}, d\rangle$ of a fork that we process; the construction in case of $\langle f, \mathit{right}, d\rangle$ is symmetric. We will define all labels of the form $f(\underline{c}, d[j]) \to q$ and $f(*, d[j]) : q$ for (the respective predecessors of) nodes of color c. If c is not saturated, we process at the same time forks of type $\langle f, \mathit{right}, c\rangle$ in the other color, that is we define labels $f(c[i], \underline{d}) \to q$ and $f(c[i], \underline{d}) : q$ for (the respective predecessors of) nodes of color d.

The transition labels for tight forks are induced from the original run. Note that all nodes $s_1, \ldots, s_m$ that occur in a tight fork $\langle t, f(t,s_1), \ldots, f(t,s_m)\rangle$ are milestones by Lemma 6.16, so we can use pointers to them in the label of t. If c is not saturated, we induce also transition labels for tight forks of kind $\langle f, \mathit{right}, c\rangle$ in color d (if c is saturated, then all labels for forks of this type are already induced from the original run by Case 1 above).

Let $C = \{c_1, \ldots\}$ and $D = \{d_1, \ldots\}$ be all non-busy milestones in colors c and d respectively. Define E_0 as the set of edges $\langle c_i, d_j\rangle$ such that the label $f(\underline{c}, d[\cdots]) \to \cdots$ is already defined for some predecessor of c_i where $d[\cdots]$ is the pointer to d_j, plus the set of edges $\langle d_j, c_i\rangle$ such that some predecessor of d_j is already labeled $f(c[\cdots], \underline{d}) \to \cdots$ where $c[\cdots]$ is the pointer to c_i, plus (if c is saturated) the set of edges $\langle d_j, c_i\rangle$ such that d_j is labeled $f(c[\cdots], \underline{d}) : \cdots$ where $c[\cdots]$ is the pointer to c_i. Let $A(c_i) = \{d_j \mid d_j \prec c_i\}$. Let $m(c_i)$ be the degree of the fork $\langle c_i, f(c_i, \ldots), \ldots\rangle$, let $p(c_i) = |\{s \in \omega(c_i) \mid \rho(s) = d\}|$.

It is easy to see that $\langle C, D, E_0, A, m, p\rangle$ satisfy the assumptions of Proposition 6.17: the bound on the cardinality of D follows from Lemma 6.14 (point 4); the bound on $\mathit{indeg}(D)$ follows from the definition of non-busy milestone; the bound on $\sum_{v\in C} m(v) + p(v)$ from Lemma 6.15. The last two conditions follow from the observation that outgoing edges in E_0 are defined only for tight forks; for non-tight forks the number of elements in $A(c_i)$ is at least $m(c_i) + \frac{m}{2}$ minus the number of busy milestones (which are not in D).

Let E_1 be the outcome of Proposition 6.17 applied on $\langle C, D, E_0, A, m, p\rangle$.

If c is not saturated, then let $A'(d_j) = \{c_i \mid c_i \prec d_j\}$. Let $m'(d_j)$ be the degree of the fork $\langle d_j, f(\ldots, d_j), \ldots\rangle$, let $p'(d_j) = |\{s \in \omega(d_j) \mid \rho(s) = c\}|$. Then, it is easy to see that $\langle D, C, E_1, A', m', p'\rangle$ also satisfies the assumptions of Proposition 6.17. Let E_2 be the outcome of Proposition 6.17 applied on $\langle D, C, E_1, A', m', p'\rangle$.

Let E be E_1 (if c is saturated) or E_2 (if c is not saturated). Then E induces a labeling of the respective nodes in S: For a fork $c_i, v_1, \ldots, v_m$ of the kind $\langle f, \mathit{left}, d\rangle$ the node v_j is labeled with $f(\underline{c}, d[j']) \to \rho(v_j)$ where j' is the index of the jth smallest with respect to $\prec$ successor of c_i in E. Moreover, c_i is labeled with $f(*, d[k']) : \rho(f(c_i, t))$ where t is the kth element of $\omega(c_i)$ of color d and k' is the index of the $(m + k)$-th smallest with respect to $\prec$ successor of c_i in E. Finally, if c is not saturated, we induce in the same way the labeling for forks $d_j, v_1, \ldots, v_m$ of the kind $\langle f, \mathit{right}, c\rangle$ and nodes d_j having witnesses of color c in the set $\omega(d_j)$.

This ends the construction of the labeling of S.

6.6. PROPERTIES OF THE CONSTRUCTED PRE-GERM.

PROPOSITION 6.20. *Let G be the labeled naked skeleton as constructed above. Then G is a consistent pre-germ.*

PROOF. First, we show that G is a skeleton according to Definition 5.3. In the construction above, every node is labeled with a transition label compatible with the corresponding transition of the automaton. Every pointer points to some milestone, which is a node in G. Moreover, every pointer used in a (transition) label of a node v points to a node v' that lies below v; thus by the same argument as in Section 5, the graph induced from G is an s-dag.

To finish the proof, we have to show that the labeling of G is consistent. Suppose that the labeling is inconsistent. In accordance with Definition 6.19, there must exist two nodes t and s and a function symbol f such that the labeling of G induces two different colors q and q' of the node representing $f(s, t)$. Since there exist pointers $c[i]$ and $d[j]$ pointing respectively to s and t, both s and t must be milestones. Hence, we can exclude Case 2 of the construction. If both s and t are busy or their colors $\rho(s)$, $\rho(t)$ are saturated (Case 1 in the construction), the labels are induced from the original run and then $q = q' = \rho(f(s, t))$, and there is no inconsistency. Therefore, for at least one of s and t, the labels must be constructed by Case 3 of the construction. But also here no inconsistencies are possible due to the construction from Section 6.4: the last condition of Proposition 6.17 ("no conflicts are created") avoids these inconsistencies. □

Recall that an *ordinary* node is a node that is neither a milestone nor have more than one predecessor.

PROPOSITION 6.21 (PUMPING LEMMA). *Suppose that a consistent pre-germ G contains a path $v_0 \to v_1 \to \cdots \to v_m$ such that*

—the colors of v_1 and v_m are the same, and
—all the nodes $v_1, \ldots, v_m$ are ordinary.

Let G' be a graph obtained from G by

—removing the nodes $v_1, \ldots, v_{m-1}$,

—defining v_m as the successor of v_0, and
—copying all request labels of v_1 to v_m.

Then G' is a consistent pre-germ.

PROOF. First, observe that G' is consistent: Indeed, local consistency follows from the fact that for all nodes t and their predecessor t' in G', the request labels of t and transition label of t' coincide with respective labels of some node and its predecessor in G; global consistency is a condition concerning request labels of milestones and transition labels of predecessors of milestones, and pumping does not change these labels at all.

Now, to prove the proposition it is enough to show that after pumping G' remains a skeleton. To do this, we have to prove two things: that the induced graph is acyclic, and that it does not contain two isomorphic subgraphs.

Since G is a skeleton, we have that for every node v, every pointer of the form $q[i]$ used in the transition label of v points to a node v' such that there is no path from v' to v in G. Now, by removing the nodes $v_1, \ldots, v_{m-1}$, we do not create new paths, so after pumping there is still no path from v' to v in G' and thus the graph induced from G' is acyclic. The proof that this graph does not contain two isomorphic subgraphs follows the lines of the proof of Claim 5.11 in Section 5. □

COROLLARY 6.22. *There exists a consistent pre-germ of size at most $|Q| \cdot 2N \cdot (|Q| + 1) \cdot (|Q| + P)$.*

PROOF. The proof follows the lines of the proof of Theorem 5.7 in Section 6.2, having in mind Claim 6.9. □

6.7. CONSTRUCTION OF A GERM. For a node t labeled with a request label of the form $f(*, d[j]) : q$ or $f(d[j], *) : q$, the *image* of this label is the node representing the term $f(t, s)$ (respectively, $f(s, t)$) such that the pointer $d[j]$ points to the node representing s. By the definition of consistency, if an image of such a request label is a node in a consistent pre-germ G, then this node is of color q. The only problem here is that this image need not be a node in G, in which case we simply add it.

LEMMA 6.23. *Let G' be a consistent pre-germ as constructed above. Let G'' be the graph obtained from G' by adding (if they were not yet in G') for all nodes t and all their request labels $f(*, d[j]) : q$ or $f(d[j], *) : q$ their images, as the predecessors of t labeled with respectively $f(\underline{\rho(t)}, d[j]) \to q$ or $f(d[j], \underline{\rho(t)}) \to q$. Finally, let G be the graph induced from G'' and let ρ' be the mapping assigning to every node t in G the right-hand side of the transition label labeling the node t. Then, $\langle G, \rho' \rangle$ is a germ.*

PROOF. First, observe that by the consistency of G', the mapping ρ' is well-defined. Let V be the set of nodes of G and V_r be the set of nodes that are in G'' but not in G'; by construction $V_r \subseteq V$, and every node in V_r is a root in G.

The faithfulness up to V_r of the run ρ' on G follows from the fact that for any node $t \in V - V_r$ (i.e., for any node t in G') and any request in $\pi(\rho'(t))$, the request is granted by the image of the respective label.

The completeness of the automaton $\mathcal{A}$ restricted to $\rho'(V - V_r)$ follows from the completeness of $\mathcal{A}$ restricted to states reachable in ρ (which follows from the fact that ρ is a run on the whole s-dag G_Σ) and an observation that by the construction

of the naked skeleton in Section 6.2, $V - V_r$ contains nodes of all reachable colors. This last observation implies also the last condition in the definition of a germ, namely that $\rho'(V - V_r) = \rho'(V)$. □

PROOF OF LEMMA 5.15. By Corollary 6.22 there exists a pre-germ G' of size at most $|Q| \cdot 2N \cdot (|Q| + 1) \cdot (|Q| + P)$. Since every node in G' is labeled with at most P request labels, the germ G constructed in Lemma 6.23 is of size at most P times bigger. In the light of Observation 6.2, the size of G is in $O(P^3 \cdot |Q|^5)$. □

REFERENCES

ACKERMANN, W. 1954. *Solvable Cases of the Decision Problem*. North-Holland, Amsterdam, The Netherlands.

AIKEN, A. 1999. Introduction to set constraint-based program analysis. *Sci. Comput. Prog. 35*, 2, 79–111.

AIKEN, A., KOZEN, D., VARDI, M., AND WIMMERS, E. L. 1993. The complexity of set constraints. In *Proceedings of the Symposium on Computer Science Logic'93*. Lecture Notes in Computer Science, vol. 832. Springer-Verlag, Berlin, Germany, 1–17.

AIKEN, A., KOZEN, D., AND WIMMERS, E. L. 1995. Decidability of systems of set constraints with negative constraints. *Information and Computation 122*, 30–44. (Preliminary version: *Technical Report 93-1362, Computer Science Department, Cornell University*, June 1993).

AIKEN, A., AND WIMMERS, E. L. 1992. Solving systems of set constraints (extended abstract). In *Proceedings of the 7th Seventh Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, Los Alamitos, CA, 329–340.

BACHMAIR, L., GANZINGER, H., AND WALDMANN, U. 1993. Set constraints are the monadic class. In *Proceedings of the 8th Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, Los Alamitos, CA, 75–83.

CHARATONIK, W. 1998a. Set constraints in some equational theories. *Inf. Computat. 142*, 40–75.

CHARATONIK, W. 1998b. An undecidable fragment of the theory of set constraints. *Inf. Proc. Lett. 68*, 3, 147–151.

CHARATONIK, W. 1999. Automata on DAG representations of finite trees. Tech. Rep. MPI-I-1999-2-001, Max-Planck-Institut für Informatik. www.ii.uni.wroc.pl/∼wch/papers/dag.ps.

CHARATONIK, W., AND PACHOLSKI, L. 1994a. Negative set constraints with equality. In *Proceedings of the 9th Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, Los Alamitos, CA, 128–136.

CHARATONIK, W., AND PACHOLSKI, L. 1994b. Set constraints with projections are in NEXPTIME. In *Proceedings of the 35th Symposium on Foundations of Computer Science*. IEEE Computer Society Press, Los Alamitos, CA, 642–653.

CHARATONIK, W., AND PODELSKI, A. 1996. The independence property of a class of set constraints. In *Proceedings of the Conference on Principles and Practice of Constraint Programming*. Lecture Notes in Computer Science, vol. 1118. Springer-Verlag, Berlin, Germany, 76–90.

CHARATONIK, W., AND PODELSKI, A. 1998. Co-definite set constraints. In *Proceedings of the 9th International Conference on Rewriting Techniques and Applications*. Lecture Notes in Computer Science, vol. 1379. Springer-Verlag, Berlin, Germany, 211–225.

CHARATONIK, W., AND PODELSKI, A. 2002. Set constraints with intersection. *Inf. Computat. 179,* 2, 213–229. (Extended abstract appeared in proceedings of LICS'97.)

CHARATONIK, W., PODELSKI, A., AND TALBOT, J.-M. 2000. Paths vs. trees in set-based program analysis. In *Proceedings of the 27th Annual ACM Symposium on Principles of Programming Languages*. ACM, New York, 330–338.

CHARATONIK, W., AND TALBOT, J.-M. 2002. Atomic set constraints with projection. In *Rewriting Techniques and Applications. 13th International Conference (RTA 2002)*. Lecture Notes in Computer Science, vol. 2378. Springer-Verlag, Berlin, Germany, 311–325.

CHENG, A., AND KOZEN, D. 1996. A complete Gentzen-style axiomatization for set constraints. In *Annual International Colloquium on Automata, Languages and Programming*. Lecture Notes in Computer Science, vol. 1099. Springer-Verlag, Berlin, Germany, 134–145.

GILLERON, R., TISON, S., AND TOMMASI, M. 1993a. Solving systems of set constraints using tree automata. In *Proceedings of the 10th Annual Symposium on Theoretical Aspects of Computer Science*. Lecture Notes in Computer Science, vol. 665. Springer-Verlag, Berlin, Germany, 505–514.

GILLERON, R., TISON, S., AND TOMMASI, M. 1993b. Solving systems of set constraints with negated subset relationships. In *Proceedings of the 34th Symposium on Foundations of Computer Science*. IEEE Computer Society Press, Los Alamitos, CA, 372–380.

GILLERON, R., TISON, S., AND TOMMASI, M. 1999. Set constraints and automata. *Inf. Computat. 149,* 1, 1–41.

GOUBAULT-LARRECQ, J. 2002. Higher-order positive set constraints. In *Proceedings of the 16th Annual Conference of the European Association for Computer Science Logic*. Lecture Notes in Computer Science, vol. 2471. Springer-Verlag, Berlin, Germany, 473–489.

HEINTZE, N. 1992. Set based program analysis. Ph.D. dissertation, School of Computer Science, Carnegie Mellon Univ. Pittsburgh, PA.

HEINTZE, N. 1993. Set based analysis of arithmetic. Tech. Rep. CS–93–221, School of Computer Science, Carnegie Mellon Univ.

HEINTZE, N., AND JAFFAR, J. 1990a. A decision procedure for a class of set constraints. In *Proceedings of the 5th Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, Los Alamitos, CA, 42–51.

HEINTZE, N., AND JAFFAR, J. 1990b. A finite presentation theorem for approximating logic programs. In *Proceedings of the 17th Annual ACM Symposium on Principles of Programming Languages*. ACM, New York, 197–209.

HEINTZE, N., AND JAFFAR, J. 1994. Set constraints and set-based analysis. In *Proceedings of the Workshop on Principles and Practice of Constraint Programming*. Lecture Notes in Computer Science, vol. 874. Springer-Verlag, Berlin, Germany, 281–298.

KODUMAL, J., AND AIKEN, A. 2005. Banshee: A scalable constraint-based analysis toolkit. In *Proceedings of the 12th International Static Analysis Symposium (SAS)*. Lecture Notes in Computer Science, vol. 3672. Springer-Verlag, Berlin, Germany, 218–234.

KOZEN, D. 1993. Logical aspects of set constraints. In *Proceedings of the 1993 Conference on Computer Science Logic*. Lecture Notes in Computer Science, vol. 832. Springer-Verlin, Berlin, Germany, 175–188.

KOZEN, D. 1994. Set constraints and logic programming (abstract). In *Proceedingss of the 1st International Conference Constraints in Computational Logics*. Lecture Notes in Computer Science, vol. 845. Springer, Berlin, Germany, 302–303.

KOZEN, D. 1995. Rational spaces and set constraints. In *TAPSOFT: Proceedings of the 6th International Joint Conference on Theory and Practice of Software Development*. Lecture Notes in Computer Science, vol. 915. Springer-Verlag, Berlin, Germany, 42–61.

KOZEN, D. 1998. Set constraints and logic programming. *Inf. Computat. 142,* 1, 2–25.

MCALLESTER, D. A., GIVAN, R., WITTY, C., AND KOZEN, D. 1996. Tarskian set constraints. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, Los Alamitos, CA, 138–147.

MÜLLER, M., NIEHREN, J., AND PODELSKI, A. 1997. Ordering constraints over feature trees. In *Proceedings of the 3rd International Conference on Principles and Practice of Constraint Programming (CP97)*. Lecture Notes in Computer Science, vol. 1330. Springer, Berlin, Germany, 297–311.

MÜLLER, M., NIEHREN, J., AND TALBOT, J.-M. 1999. Entailment of atomic set constraints is PSPACE-complete. In *14th Annual IEEE Symposium on Logic in Computer Science (LICS)*. IEEE Computer Society Press, Los Alamitos, CA, 285–294.

PATERSON, M. S., AND WEGMAN, M. N. 1978. Linear unification. *Journal of Computer and System Sciences 16*, 158–167.

REYNOLDS, J. C. 1969. Automatic computation of data set definitions. *Information Processing 68*, 456–461.

RYCHLIKOWSKI, P., AND TRUDERUNG, T. 2004. Set constraints on regular trees. In *Computer Science Logic*. Lecture Notes in Computer Science, vol. 3210. Springer-Verlag, Berlin, Germany, 458–472.

SEYNHAEVE, F., TISON, S., TOMMASI, M., AND TREINEN, R. 2001. Grid structures and undecidable constraint theories. *Theoret. Comput. Sci. 258*, 453–490.

SEYNHAEVE, F., TOMMASI, M., AND TREINEN, R. 1997. Grid structures and undecidable constraint theories. In *Proceedings of the 9th International Joint Conference on Theory and Practice of Software Development*. Lecture Notes in Computer Science, vol. 1214. Springer-Verlag, Berlin, Germany, 357–368.

STEFANSSON, K. 1994. Systems of set constraints with negative constraints are *NEXPTIME*-complete. In *Proceedings of the 9th Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, Los Alamitos, CA, 137–141.

TALBOT, J.-M. 2000. The $\exists\forall^2$ fragment of the first-order theory of atomic set constraints is Π_1^0-hard. *Inf. Proc. Lett. 74*, 27–33.

TALBOT, J.-M., DEVIENNE, P., AND TISON, S. 2000. Generalized definite set constraints. *Constraints: Int. J. 5,* 1-2, 161–202.

URIBE, T. E. 1992. Sorted unification using set constraints. In *Proceedings of the 11th International Conference on Automated Deduction*. Lecture Notes in Artificial Intelligence, vol. 607. Springer-Verlag, Berlin, Germany, 163–177.

RECEIVED MAY 2006; REVISED AUGUST 2009 AND JANUARY 2010; ACCEPTED FEBRUARY 2010