Solving set constraints for greatest models

Witold Charatonik   Andreas Podelski

MPI–I–97–2–004   April 1997

Author's Address

Max-Planck Institut für Informatik
Im Stadtwald, D-66123 Saarbrücken, Germany
{witold;podelski}@mpi-sb.mpg.de

Publication Notes

At the moment of printing, the results presented in this technical report are already superseded by results in its following follow-up papers.

- Philippe Devienne, Jean-Marc Talbot and Sophie Tison. *Solving Classes of Set Constraints with Tree Automata*, University of Lille, LIFL, Technical Report IT-303. May 1997. Also, in Gert Smolka, editor, *Proceedings of the Third International Conference on Constraint Programming*, Springer LNCS 1330, pp. 62–76, 1997.
- Witold Charatonik and Andreas Podelski. *Co-definite set constraints.* Available at `www.mpi-sb.mpg.de/~podelski/papers/union.ps.`
- Andreas Podelski, Witold Charatonik and Martin Müller. *Set-based diagnosis of concurrent constraint programs.* Available at `www.mpi-sb.mpg.de/~podelski/papers/diagnosis.ps.`
- Philippe Devienne, Jean-Marc Talbot and Sophie Tison. *Co-definite set constraints with membership constraints.* University of Lille, LIFL, Technical Report. September 1997.

The paper cited in the reference [11] in this report has appeared as Witold Charatonik and Andreas Podelski, *Set constraints with intersection.* In Glynn Winskel, editor, *Proceedings of LICS'97, the 12th IEEE Annual Symposium on Logic in Computer Science*, Warsaw, Poland, June 29 - July 2, 1997, pp. 362–372.

Abstract

In set-based program analysis, one first infers a set constraint φ from a program and then, in a constraint-solving process, one transforms φ into an effective representation of sets of program values. Heintze and Jaffar have thus analyzed logic programs with respect to the least-model semantics. In this paper, we present a set-based analysis of logic programs with respect to the greatest model semantics, and we give its complexity characterization. We consider set constraints consisting of inclusions $x \subseteq \tau$ between a variable x and a term τ with intersection, union and projection. We solve such a set constraint by computing a representation of its greatest solution (essentially as a tree automaton). We obtain that the problem of the emptiness of its greatest solution is DEXPTIME-complete. The choice of the *greatest* model for set-based analysis is motivated by the verification of safety properties ("no failure") of reactive (*i.e.*, possibly non-terminating) logic programs over infinite trees. Therefore, we account also for infinite trees.

1 Introduction

Coined by Heintze and Jaffar in 1990 [17, 18], the notion of set constraints now refers to a wide range of classes of constraints over set-valued variables. These classes have been studied for their logic, algorithmic and topological properties [20, 21]. The main application domain is set-based program analysis in the imperative, functional or logic programming paradigm [3, 5, 18, 19, 26, 29, 34]. Here, the algorithmic issue is the solving of the set constraint inferred from a program. The solved form of a set constraint provides an effective representation of the set-valued solutions. It also exhibits in an effective way which variables are valued as the empty set. When a constraint can express negative information then solving it includes testing whether it has a solution at all. Thus, in the complexity-theoretic investigations of several classes of set constraints [4, 15, 8, 1, 16, 2, 9, 10], one has formalized the constraint-solving problem as the satisfiability test. If, however, all set constraints in the class are satisfiable, then the emptiness test of a characteristic solution (the least or the greatest one, depending on the application) is a suitable formalization of the constraint-solving problem.[1]

In this paper, we consider the class where a set constraint φ consists of inclusions $x \subseteq \tau$ between a variable x and a *positive* set expression, *i.e.*, a term τ built up with union, intersection and projection[2] (and, by default, with function symbols from a given signature, but without complement). We refer to the set constraints in this class by *constraints* for short. We show how one can use the class for the set-based analysis of logic programs with respect to the greatest model semantics. We define the inference of a constraint φ_P from a logic program P and prove that the greatest solution of φ_P approximates the greatest model of P. We design and prove correct an algorithm that transforms a constraint φ into one representing the greatest solution (essentially as a (non-deterministic top-down) tree automaton). We prove its exponential time complexity, and since we can easily encode the problem of the emptiness of the intersection of tree automata [13, 30] into the problem of the emptiness of the greatest solution of a constraint in our class, we obtain that this problem is DEXPTIME-complete.

NEXPTIME characterizes the constraint-solving problems in several classes of set constraints [8, 1, 32, 9, 10]. The challenge in the design of our algorithm was to overcome the doubly-exponential time barrier. Our intuition behind the proof of the existence of a simply-exponential upper time-bound has been that the states of the tree automaton that represents the greatest solution can be chosen as the intersections of variables appearing in the constraint; hence, exponentially many states should suffice. The difficulty is the construction of that automaton. The algorithm has to deal with intersections of unions of variables (take, for example, the constraint

$$\begin{aligned} x \subseteq f^{-1}_{(2)}(y) \ \wedge\ & y \subseteq f(a,\top) \cup f(\top,a) \ \wedge \\ & y \subseteq f(b,\top) \cup f(\top,a) \end{aligned}$$

whose greatest solution for x is $\{a\}$). Our algorithm works by fixpoint iteration of the operator that adds immediate consequences under a set of axioms. One important idea underlying our algorithm is to add only inclusions between a variable and a union term (without intersections), of which there are only exponentially many. The operation that computes immediate consequences may call (exponential-time) sub-algorithms which manipulate intersections of union terms and unions of intersection terms.

We can use our algorithm also for the satisfiability test of a syntactically larger class of constraints, namely with terms $f(x_1,\dots,x_n)$ on the left-hand side of an inclusion, but now interpreted over path-closed sets of trees. Regular path-closed sets correspond to deterministic top-down tree automata [14]. Since the emptiness problem also for this restricted class of tree automata is DEXPTIME-hard [30] and since it can be encoded in the satisfiability problem, we obtain the DEXPTIME-completeness for this satisfiability problem.

[1] Frühwirth, Shapiro, Vardi and Yardeni in [13] take the membership test for formalizing the problem of their type analysis (of logic programs) which is related to set-based analysis.

[2] The application of the projection operator for a function symbol f and the k-th argument position on a set S of trees is defined by $f^{-1}_{(k)}(S) = \{t \mid \exists t_1, \dots t_n :\ t_k = t,\ f(t_1,\dots,t_k,\dots,t_n) \in S\}$.

2 Motivation and related work

The definition of our class and the framework of the approximation (with the inference of the constraint φ_P and the soundness proof) can be seen in dual analogy with the framework by Heintze and Jaffar [18, 17]. They consider the class of set constraints with inclusions of the form $\tau \subseteq x$ between a variable x and a positive set expression τ, and they use it for the set-based analysis of logic programs with respect to the least-model semantics. They give a constraint-solving algorithm which computes a representation of the least solution. Cousot and Cousot have formalized the set-based analysis within the framework of abstract interpretation in [12]. The algorithm can be used as a satisfiability test for the larger class of definite set constraints where also terms $f(x_1, \ldots, x_n)$ may appear on the right-hand side of an inclusion; this fact has led to the first decidability result for a class of set constraints [17]. Recent results [11] of the present authors imply that the satisfiability problem of definite set constraints as well as the emptiness problem of the least solution of an HJ set constraint are DEXPTIME-complete.

The above-mentioned duality with HJ set constraints does not extend to the constraint-solving algorithm for our constraints. The complement of the application of the projection is generally not the application of the projection of the complement; the algorithm given in Section 4.2 in [17] does therefore *not* compute the greatest solution.[3] The dual of the emptiness problem for a constraint φ *without* projection is the universality problem (and not the emptiness problem) for the HJ set constraint $\tilde{\varphi}$ dual to φ. That is, one can test emptiness for φ by first computing the tree automaton for $\tilde{\varphi}$ and then testing it for universality. By the bounds in [11] and the DEXPTIME-completeness of the universality of tree automata [30], this yields a doubly-exponential emptiness test for constraints φ without projection. To our knowledge, no algorithm has been given before for computing the greatest solution and testing the emptiness for the whole class of our constraints. Also, there was no proof that the greatest solution is a regular set of trees.

The class of path-closed set constraints has been considered first by Mishra in an early paper [25]. It is widely agreed that the presentation of the satisfiability test in [25] is uncomplete; in any case, it does not consider complexity issues. Mishra has used these constraints for approximating the least model; our results imply that they even approximate the greatest model.

Our work is motivated by a concrete application as well as by the general interest in constraint-solving problems for set constraints. The concrete application is the testing of safety properties that are characterized by the greatest model of a logic program. The greatest model $gm(P)$ of a logic program P is given by the greatest fixpoint of the immediate consequence operator T_P, which, in the case of a logic program over infinite trees, is obtained after ω-many iterations [22]. Related to this well-known fact is the following characterization. A goal $p(t)$ consisting of the call of the predicate p applied to the term t will never fail if and only if the set of ground instances of the goal intersected with $gm(P)$ is nonempty. The only-if direction in this characterization uses a fairness assumption (as realized in a parallel execution model); for a simple proof see [28]. If logic programs are used for specifying reactive, *i.e.*, possibly non-terminating processes, or as approximations of concurrent constraint (cc) programs (written, for example, in Oz, a cc programming language over the domain of infinite trees [31]), then non-failure is an important safety property. Other safety properties of logic programs P can be formulated and can be characterized as well with the greatest model of P, as shown in the assertion-based verification schema of [28]. Our set-based analysis provides the test of a sufficient condition for the emptiness of the denotation of a predicate p in $gm(P)$, and, by using the tree-automata representation, for the emptiness of its intersection with the set of ground instances of a goal $p(t)$.[4] This amounts to the test of a necessary condition of the corresponding safety property. In order to account for this application domain, we have to represent the greatest solution of a constraint over sets of infinite trees. We can do this by a rather straightforward construction (which is implicit in [6]) of a Büchi automaton over infinite

[3] The algorithm transforms the constraint $x = f^{-1}_{(2)}(f(a,a))$ — whose greatest solution is $\{a\}$ — into the constraint $x = \mathit{dual}(f^{-1}_{(2)}(f(a,a))) = f^{-1}_{(2)}(\mathit{dual}(f(a,a))) = f^{-1}_{(2)}(\Omega_f \cup f(\Omega_a, \top) \cup f(\top, \Omega_a)) = \top$ and then yields the constraint $x = \mathit{dual}(\top) = \bot$ — whose greatest solution is the empty set.

[4] The membership test (see Footnote 1) is not relevant for this application.

trees from the regular system of equations which is the output of our constraint-solving algorithm.

The investigation of our class of set constraints is interesting in its own right. It can be seen as part of the systematic characterization of the natural and useful classes of set constraints by their constraint-solving complexity. Aiken, Kozen, Vardi and Wimmers started with the study of the classes obtained by restricting the signature of the trees [1]. We are interested in the classes obtained by a restricted choice of set operators. In [11] we have taken intersection as the only set operator. It turns out that the choice of union as the only operator does not lead to an interesting class. We do not see any potential application; note that neither the least nor the greatest solution generally exists (to see this, take the inclusions $a \subseteq x \cup y$ and $f(x, y) \subseteq f(a, a) \cup f(b, b)$, respectively). The class considered in this paper accommodates union in a useful way.

3 Preliminaries

We assume given a ranked alphabet Σ fixing the arity $n \geq 0$ of its function symbols $f, g, a, b, \ldots$ and an infinite set $\mathcal{V}$ of variables $x, y, z, u, v, w, \ldots$. We use $t, s, \ldots$ as meta-variables for finite or infinite trees, $\tau, \tau_1, \tau_2, \ldots$ for unions of terms of depth ≤ 1, and φ for constraints .

$$\begin{array}{lcl} \tau & ::= & x \mid f(\bar{u}) \mid \tau_1 \cup \tau_2 \\ \varphi & ::= & x \subseteq \tau \mid x \subseteq f^{-1}_{(k)}(u) \mid \varphi_1 \wedge \varphi_2 \end{array}$$

We write $\bar{u}$ for the tuple $(u_1, \ldots, u_n)$ of variables and $\bar{t}$ for the tuple $(t_1, \ldots, t_n)$ of trees, where $n \geq 0$ is given implicitly (*e.g.*, in $x \subseteq f(\bar{u})$ by the arity of the function symbol f). We write $\bar{u} \subseteq \bar{v}$ for $\{u_1 \subseteq v_1, \ldots, u_n \subseteq v_n\}$. As is usual, we identify a conjunction of constraints with the set of all conjuncts.

We use $\mathcal{V}(E)$ for the set of variables contained in the expression E, and $\mathsf{Terms}(\varphi)$ for the set of all *flat* terms τ (*i.e.*, without union) occurring in φ.

Note that the terms τ in the constraints φ do not contain intersection and are unions of flat terms. This does not present a proper restriction since we can replace intersection by conjunction and we can remove nested expressions by introducing new variables. This transformation does not change the complexity measure. The number of new variables is linear.

4 Constraint inference

We start with an example. The program P below contains two predicates e and o defining even and odd numbers (where the natural number n is represented by the tree $s^n(zero)$).

$$\begin{array}{l} e(s^2(X_e)) \leftarrow e(X_e). \\ e(zero). \\ o(X_o) \leftarrow e(s(X_o)). \end{array}$$

For the logical semantics of P we take Clark's completion, *i.e.*, the following formula.

$$\begin{array}{l} e(X_e) \leftrightarrow \exists Y (X_e = s^2(Y) \wedge e(Y) \vee X_e = zero) \\ o(X_o) \leftrightarrow \exists Y (Y = s(X_o) \wedge e(Y)) \end{array}$$

The conjunction of the following formulas characterizes the models of the regular approximation of the program P (which, in this case, is an accurate approximation); the variables x_e and x_o now range over sets of trees.

$$\begin{array}{l} x_e = \{s^2(t) \in T_\Sigma \mid t \in x_e\} \cup \{zero\} \\ x_o = \{t \in T_\Sigma \mid s(t) \in x_e\} \end{array}$$

Using the projection operator and the function symbols in Σ as functions over sets of trees, we transform the above formula into the following equivalent one.

$$\begin{array}{l} x_e = s^2(x_e) \cup zero \\ x_o = s^{-1}_{(1)}(x_e) \end{array} \qquad (1)$$

If one is interested in the least solution of the formula (1), one may replace the equality by an inclusion $\subseteq$ after having exchanged left- and right-hand sides. Then, occurrences of the union operator may be eliminated by adding conjunctions of inclusions. This explains why HJ set constraints may be restricted to inclusions $\tau \subseteq x$ where τ contains intersection and projection but no union (the intersection translates the conjunction of atoms in the body of a clause). The HJ set constraint derived from P is

$$zero \subseteq x_e \wedge s^2(x_e) \subseteq x_e \wedge s^{-1}_{(1)}(x_e) \subseteq x_o.$$

Its solved form is

$$zero \subseteq x_e \wedge s^2(x_e) \subseteq x_e \wedge s(x_e) \subseteq x_o.$$

Its least solution (over sets of finite trees or over sets of possibly infinite trees) assigns the sets of (finite) trees representing even and, respectively, odd numbers to x_e and x_o.

Since we want to approximate the greatest model of a logic program, we are interested in the greatest solution of the formula (1). We replace the equality by an inclusion $\subseteq$ (leaving left- and right-hand sides as before). Since now we can eliminate occurrences of the intersection operator, we can restrict our constraints to inclusions $x \subseteq \tau$ where τ contains union and projection but no intersection. We derive the following constraint from P.

$$x_e \subseteq zero \cup s^2(x_e) \wedge x_o \subseteq s^{-1}_{(1)}(x_e)$$

Its solved form is

$$x_e \subseteq zero \cup s^2(x_e) \wedge x_o \subseteq s(x_e).$$

Its greatest solution over sets of finite trees is the assignment mentioned above. Its greatest solution over sets of possibly infinite trees, however, adds the infinite tree $s^\omega = s(s(s\ldots))$ to both sets valuing x_e and x_o. For illustrating the characterization of "non-failure" in Section 2, we note that the predicate call $p(x)$ of the predicate p defined by $p(x) \leftarrow e(x) \wedge o(x)$ will never fail, in accordance with the fact that the set of its ground instances intersected with the denotation of p in the greatest model is nonempty (since it contains $p(s^\omega)$).

We will now describe the general inference of a constraint φ_P from a logic program P. We assume that P defines the predicates $p \in \mathsf{Pred}$ by the formula below. We use capital letters X, Y, ... for variables ranging over trees. In order to simplify the presentation, we assume that all predicates are unary (and that the signature of function symbols contains symbols forming tuples). We use $\exists_{-X}$ to denote the existential quantification of all variables in the formula but X.

$$P \;\equiv\; \bigwedge_{p \in \mathsf{Pred}} p(X_p) \leftrightarrow \bigvee_i \exists_{-X_p}(X_p = t_i \wedge \bigwedge_j p_{ij}(t_{ij}))$$

For the inference of the constraint φ_P, we assume that the different clauses are renamed apart (if not, we apply α-renaming of existentially quantified variables). We introduce a fresh variable z_t for each subterm t appearing in the formula and then define the constraint $\Phi(t \subseteq x)$ for a term t and a variable x inductively on the depth of t.

$$\begin{array}{rcl}
\Phi(y \subseteq x) & = & y \subseteq x \\
\Phi(a \subseteq x) & = & true \\
\Phi(f(t_1,\ldots,t_n) \subseteq x) & = & z_{f(t_1,\ldots,t_n)} \subseteq x \wedge \;\; z_{t_1} \subseteq f^{-1}_{(1)}(z_{f(t_1,\ldots,t_n)}) \wedge \Phi(t_1 \subseteq z_{t_1}) \wedge \ldots \wedge \\
 & & \qquad\qquad\qquad\quad\; z_{t_n} \subseteq f^{-1}_{(n)}(z_{f(t_1,\ldots,t_n)}) \wedge \Phi(t_n \subseteq z_{t_n})
\end{array}$$

The constraint inferred from P has the following form.

$$\varphi_P \;\equiv\; \bigwedge_{p \in \mathsf{Pred}} x_p = \bigcup_i t_i \;\wedge\; \bigwedge_i \bigwedge_j \; \Phi(t_{ij} \subseteq x_{p_{ij}})$$

1. $x \subseteq x$ for all $x \in \mathcal{V}(\varphi)$
2. $x \subseteq \top$ for all $x \in \mathcal{V}(\varphi)$
3. $\top \subseteq \bigcup_{f\in\Sigma} f(\top, \ldots, \top)$
4. $\bot \subseteq f(\bot, \ldots, \bot)$ for all $f \in \Sigma$
5. $x \subseteq y,\ y \subseteq z \rightarrow x \subseteq z$
6. $x \subseteq \tau_1 \cup y,\ y \subseteq \tau_2 \rightarrow x \subseteq \tau_1 \cup \tau_2$
7. $u \subseteq f^{-1}_{(k)}(x) \rightarrow \bigwedge_i u \subseteq \bigcup_j u_{ij}$ where $\bigcap_i \bigcup_j u_{ij} = FCNF(f^{-1}_{(k)}(FDNF_x))$
8. $u \subseteq f^{-1}_{(k)}(x) \rightarrow u \subseteq \bot$ if $f^{-1}_{(k)}(FDNF_x) = \bot$

Table 1: Axioms for constraints φ.

1. $x \subseteq FDNF_x$
2. $x_1 \cap \ldots \cap x_n \subseteq FDNF_{x_1 \cap \ldots \cap x_n}$

Table 2: Axioms for computing the tree automaton corresponding to a constraint φ.

Theorem 1 The greatest model $gm(P)$ of the program P is approximated by the greatest solution σ of φ_P; more precisely, for all predicates $p \in \mathsf{Pred}$ holds:

$$\{t \in T_\Sigma \mid p(t) \in gm(P)\} \subseteq \sigma(x_p).$$

Proof. The statement is a consequence of the framework of Heintze and Jaffar in [18] or the one given by Cousot and Cousot in [12]. In a more direct proof one defines an abstract program $P^\#$ without "non-regular" subtree dependencies and then expresses (an approximation of) the fixpoint equation of $T_{P^\#}$ by φ_P. The definition of the abstract program $P^\#$ below is inspired by McAllester [23]. We here assume that no variable in t_i occurs twice (if not, we replace t_i by $\sigma(t_i)$ where σ renames apart the different occurrences of the same variable and repeat the body of the clause for each occurrence).

$$P^\# \equiv \bigwedge_{p\in\mathsf{Pred}} p(X_p) \leftrightarrow \bigvee_i \exists \mathsf{Var}(t_i)\ (X_p = t_i \wedge \bigwedge_{X\in\mathsf{Var}(t_i)} \exists_{-X} \bigwedge_j p_{ij}(t_{ij}))$$

□

5 Algorithm

Before we present the algorithm, we need some basic definitions. The following rather technical definition is an operational variant of the standard concept of disjunctive normal form, whose computation would here require doubly-exponential time.

Definition 1 (FDNF) The *full disjunctive normal form* of an expression $E = \bigcap_{i\in I} \bigcup_{j\in J} f_{ij}(\bar{u}_{ij})$

is the expression

$$FDNF(E) = \bigcup\{f(\bigcap S_1, \ldots, \bigcap S_k) \mid \begin{array}{l} f \in \Sigma,\ k = \text{arity}(f), \\ S_1 \subseteq \mathcal{V}(E), \ldots, \bigcap S_k \subseteq \mathcal{V}(E), \\ \forall i \in I\ \exists j \in J:\ \ f = f_{ij} \wedge u_{ij,1} \in S_1 \wedge \ldots \wedge u_{ij,k} \in S_k\}. \end{array}$$

Here, we define the union over the empty set as $\bot$.

Example 1 $FDNF((f(u, v_1) \cup f(u, v_2)) \cap f(u, u))$ is the expression $f(u, u \cap v_1) \cup f(u, u \cap v_2) \cup \ldots \cup f(u \cap v_1 \cap v_2, u \cap v_1 \cap v_2)$ which contains redundant disjuncts. To illustrate the remark after the definition, we note that $FDNF(a \cap b) = \bot$.

Definition 2 ($\mathbf{FDNF}_q$) *Given a constraint φ, we define the following expression for each intersection $q = x_1 \cap \ldots \cap x_n$ of variables x_i in φ.*

$$FDNF_q = FDNF(\bigcap_{i=1}^{n} \bigcap_{x \subseteq \bigcup_j f_j(\bar{u}_j) \in \varphi} \bigcup_j f_j(\bar{u}_j))$$

Definition 3 (FCNF) The *full conjunctive normal form* of an expression $E = \bigcup_{i \in I} \bigcap_{j \in J} u_{ij}$, written $FCNF(E)$, is the expression

$$FCNF(E) = \bigcap\{\bigcup S \mid S \subseteq \mathcal{V}(E),\ \forall i \in I\ \exists j \in J:\ \ u_{ij} \in S\}.$$

We define the algorithm by fixed point iteration with respect to the axioms in Table 1. We start with the constraint φ, and at each iteration step we add the direct consequences under the axioms 1–6 in Table 1 of the set of constraints derived so far. When the fixed point is reached, we forget for a moment about all constraints involving projections and constraints of the form $x \subseteq \tau \cup y$ where y is a variable. For each variable x, we transform all constraints of the form $x \subseteq \bigcup_j f_{ij}(\bar{u}_{ij})$ to one inclusion $x \subseteq FDNF_x$ between x and an expression in disjunctive normal form (Axiom 1 in Table 2). For complexity reasons, we use here the FDNF. Since the FDNF involves intersections of variables, we compute again full disjunctive normal forms for these intersections (Axiom 2 in Table 2).

The formulas in Table 2 are inclusions $q \subseteq \bigcup_{j \in J} f_j(\bar{q}_j)$ between intersection variables q (intersections of variables in $\mathcal{V}(\varphi) \uplus \{\top, \bot\}$) and unions of flat terms over intersection variables. These inclusions define a top-down tree automaton $\mathcal{A}$ in the following way. The states of the automaton are intersection variables; the constraint $q \subseteq \bigcup_{j \in J} f_j(\bar{q}_j)$ defines a set of transitions for the state q, namely $\{(q, f_j, \bar{q}_j) \mid j \in J\}$. Axioms 2 and 3 in Table 1 are used to define transitions for unconstrained variables (the greatest solution assigns the set of all trees to the variable $\top$). Axioms 3 uses the assumption that the signature Σ is finite; this assumption can easily be circumvented.

Let $\mathcal{L}(\mathcal{A}(x))$ denote the set of trees recognized by the automaton, with x as initial state, and $\mathcal{L}(\mathcal{A}(f(x_1, \ldots, x_n))) = f(\mathcal{L}(\mathcal{A}(x_1)), \ldots, \mathcal{L}(\mathcal{A}(x_1)))$. Using a standard algorithm for tree automata, we check for each state x if the set $\mathcal{L}(\mathcal{A}(x))$ is empty. The result of the emptiness tests are used when the direct consequences under Axioms 7 and 8 in Table 1 are added. These two axioms use the following notation.

$$f_{(k)}^{-1}(\bigcup_i f_i(q_{i1}, \ldots, q_{in})) = \bigcup_{\{i \mid f = f_i, \mathcal{L}(\mathcal{A}(f_i(q_{i1}, \ldots, q_{in}))) \neq \emptyset\}} q_{ik}$$

First, we remove all empty states (that is, all states q satisfying $\mathcal{L}(\mathcal{A}(q)) = \emptyset$). Given a conjunct of the form $u \subseteq f_{(k)}^{-1}(x)$, we thus obtain a formula (over intersection variables q_i) constraining x which is of the form $x \subseteq \bigcup_i f_i(\bar{q}_i)$ where all $f_i(\bar{q}_i)$ are nonempty. From this we infer $u \subseteq \bigcup_{\{i \mid f_i = f\}} q_{i,k}$. Here, $q_{i,k}$ is the k-th component of the tuple $\bar{q}_i$. The variables $q_{i,k}$ are intersection variables. Therefore, we transform the previous formula into a *full conjunctive normal form* and then express it as a conjunction of constraints over original variables. When the fixed point with respect to all axioms in Table 1 is reached, the valuation α defined by $\alpha(x) = \mathcal{L}(\mathcal{A}(x))$ is the greatest solution of φ.

Example 2 Let $\Sigma = \{zero, s\}$, where $zero$ is a constant and s is unary. Consider the constraint $x_e \subseteq zero \cup s(x_o), x_o \subseteq s(x_e), x_e \subseteq x$. The application of axioms in Table 2 gives as the normal form for x_e the constraint $x_e \subseteq zero \cup s(x_o \cap \top)$ and for $x_o \cap \top$ the constraint $x_o \cap \top \subseteq s(x_e \cap \top)$. The automaton then has the transitions $(x_e, zero)$, $(x_e, s, x_o \cap \top)$, and others. Thanks to the introduced variable $\top$, the transitions from x, namely $(x, zero)$ and $(x, s, \top)$, are defined.

Example 3 Consider the constraint $x_e \subseteq zero \cup s(x_o)$, $x_o \subseteq s(x_e)$, $x \subseteq zero$, $x \subseteq x_o$, $\mathcal{Y} \subseteq f(x_e, x) \cup f(x_o, x_o), u \subseteq f_1^{-1}(\mathcal{Y})$. After the first iteration (that is, the closure under Axioms 1–6 in Table 1), we have the constraint $u \subseteq zero \cup s(\top) \cup f(\top, \top)$, $\mathcal{L}(\mathcal{A}(u))$ is the set of all trees and the set $\mathcal{L}(\mathcal{A}(x))$ is empty. In the second iteration we infer $u \subseteq x_o$, and $\mathcal{L}(\mathcal{A}(u))$ is the set of odd numbers, which is the greatest solution for u.

6 Correctness

The next two lemmas simply express that both full normal forms preserve the meaning of an expression.

Lemma 1 (FDNF) For any expression E of the form $\bigcap_{i \in I} \bigcup_{j \in J} f_{ij}(\bar{u}_{ij})$, the equality $\alpha(E) = \alpha(FDNF(E))$ holds for every valuation α.

Proof. To see that $\alpha(E) \subseteq \alpha(FDNF(E))$, transform E into a disjunctive normal form. Now, using the equality $\alpha(f(\bar{u}) \cap g(\bar{v})) = \emptyset$ for $f \neq g$ and the equality $\alpha(f(u_1, \dots, u_n) \cap f(v_1, \dots, v_n)) = \alpha(f(u_1 \cap v_1, \dots, u_n \cap v_n))$, we can transform the result to an expression such that it is in disjunctive normal form and each disjunct satisfies the condition from the definition of $FDNF(E)$.

To see that $\alpha(FDNF(E)) \subseteq \alpha(E)$, take the partial ordering on tuples of intersections defined by $(\bigcap S_1, \dots, \bigcap S_n) \prec (\bigcap S'_1, \dots, \bigcap S'_n)$ (which we abbreviate by $\overline{\bigcap S} \prec \overline{\bigcap S'}$) if $S_i \subseteq S'_i$ holds for all $i = 1, \dots, n$. We observe that, if $\overline{\bigcap S} \prec \overline{\bigcap S'}$, then $\alpha(f(\overline{\bigcap S}) \cup f(\overline{\bigcap S'})) \subseteq \alpha(f(\overline{\bigcap S}))$. Discard from $FDNF(E)$ all disjuncts that are not minimal in this ordering, and call the result F. By the observation above, $\alpha(FDNF(E)) = \alpha(F)$. We have to show that $\alpha(F) \subseteq \alpha(E)$. Take any disjunct $f(\overline{\bigcap S})$ from F. We will show that the value of this disjunct under α is equal to the value of some disjunct from the disjunctive normal form of E. We know that for all $i \in I$ there exists a $j_i \in J$ such that $f_{ij_i} = f$ and $\bar{u}_{ij_i} \in \overline{S}$. Hence, for all $k = 1, \dots, a(f)$, it holds that $\bigcup_{i \in I} \{u_{ij_i,k}\} \subseteq S_k$, and by the minimality of $\overline{\bigcap S}$ these two sets are equal. Therefore, $\alpha(f(\overline{\bigcap S})) = \alpha(\bigcap_{i \in I} f(\bar{u}_{ij_i}))$. □

Lemma 2 For any expression E of the form $\bigcup_{i \in I} \bigcap_{j \in J} u_{ij}$, the equality $\alpha(E) = \alpha(FCNF(E))$ holds for every valuation α.

Proof. The proof is similar to the proof of Lemma 1; we can take the expression dual to E (replace unions with intersections and vice versa), compute the full disjunctive normal form (this time over variables, not terms $f(\bar{u})$) and then take once more the dual, which is in conjunctive normal form. □

The next lemma is folklore in tree automata theory. It expresses essentially the construction of a tree automaton $\mathcal{A}$ from a regular system of equations such that $\mathcal{L}(\mathcal{A})$ is its greatest solution. In the case of of finite trees, the least and the greatest solutions coincide. In the case of infinite trees, the least and greatest solution differ by trees with possibly infinite branches. In that case, the constructed automaton is a Büchi automaton where all states are final states. The construction is implicit in [6]. The tree automaton $\mathcal{A}$ accepts a tree t iff there exists a run of $\mathcal{A}$ on t; this acceptance condition works for finite as well as for infinite trees.

Lemma 3 The valuation $\alpha : x \mapsto \mathcal{L}(\mathcal{A}(x))$ is the greatest solution of the constraint $\bigwedge_{x \in \mathcal{V}(\varphi)} x \subseteq FDNF_x$.

Proof. We will prove the lemma not only for the variables from $\mathcal{V}(\varphi)$, but also for intersections of these variables. We recall that $\mathcal{A}(x)$ is the tree automaton defined in Section 5 which has the intersection variable x as its initial state.

For one direction of the proof, let β be any solution of $\bigwedge_x x \subseteq FDNF_x$ (in particular, it can be the greatest one). We will show that $\beta(x) \subseteq \mathcal{L}(\mathcal{A}(x))$. If $\beta(x)$ is empty, then the inclusion is trivially satisfied; otherwise, take any tree $t \in \beta(x)$. By induction of the depth of the positions p in t, we will construct a run of $\mathcal{A}(x)$ on t that satisfies the following invariant: If $\mathcal{A}(x)$ is in state q at position p, then the subtree $t|_p$ of t rooted at p belongs to $\beta(q)$.

For the root position, the initial state is x and $t \in \beta(x)$. Let $\mathcal{A}(x)$ be in state q at position p such that $t|_p \in \beta(q)$. If $t|_p$ is a tree of the form $f(t_1, \ldots, t_n)$, then we will continue the construction of the run to the positions $p.1, \ldots, p.n$. Since β is a solution, there must be a disjunct $f(q_1, \ldots, q_n)$ in $FDNF_q$ such that $t_i \in \beta(q_i)$ for $i = 1, \ldots, n$. But then, by the definition of $\mathcal{A}(x)$, there exists a transition $(q, f, q_1, \ldots, q_n)$.

For the other direction of the proof, we will show that α satisfies $\bigwedge_{x \in \mathcal{V}(\varphi)} x \subseteq FDNF_x$. Take any tree $t = f(t_1, \ldots, t_n)$ (n can be equal to 0) from $\mathcal{L}(\mathcal{A}(x))$. By the definition of $\mathcal{L}(\mathcal{A}(x))$, there exists a computation of $\mathcal{A}(x)$ on t, starting from x. Let $(x, f, x_1, \ldots, x_n)$ be the first transition used in this computation. By the definition of a run , there are runs on t_i starting from x_i, hence $t_i \in \mathcal{L}(\mathcal{A}(x_i))$. By the definition of $\mathcal{A}(x)$, there is a disjunct $f(x_1, \ldots, x_n)$ in $FDNF_x$. Therefore, $\mathcal{L}(\mathcal{A}(x)) \subseteq \bigcup_{(x,f,x_1,\ldots,x_n)\in\delta} f(\mathcal{L}(\mathcal{A}(x_1)), \ldots, \mathcal{L}(\mathcal{A}(x_n)))$ where δ is the set of all transitions of $\mathcal{A}(x)$. This means that α satisfies $x \subseteq FDNF_x$. □

Proposition 1 (Soundness) The axioms in Table 1 are valid under all valuations assigning the set of all trees to the variable $\top$ and the empty set to the variable $\bot$.

Proof. The proof is done by inspection of each axiom. The validity of Axioms 7 and 8 follows from consecutive applications of Lemma 1, Lemma 3, and Lemma 2. □

Proposition 2 (Completeness) If φ contains all of its consequences under the axioms in Table 1, then φ is implied by its *solved form* φ^S, where

$$\varphi^S = \{x \subseteq FDNF_x \mid x \in \mathsf{Var}(\varphi)\}.$$

Proof. Let α be a valuation satisfying φ^S. We show that α satisfies each conjunct in φ. The conjuncts of the form $x \subseteq \bigcup_i f_i(\bar{u}_i)$ are trivially satisfied, since $FDNF_x$ is equivalent to an intersection of the expressions $\bigcup_i f_i(\bar{u}_i)$).

We will show the satisfaction of the constraints $x \subseteq \bigcup_i f_i(\bar{u}_i) \cup \bigcup_j y_j$ (this includes the case $x \subseteq y$) indirectly. Suppose $t \notin \alpha(\bigcup_i f_i(\bar{u}_i) \cup \bigcup_j y_j)$; we will show $t \notin \alpha(x)$. By Axioms 2, 3 and 5, every variable y_j occurs in a constraint of the form $y_j \subseteq \bigcup_k f_{jk}(\bar{u}_{jk})$, and $FDNF_{y_j}$ is of the form $FDNF(E)$ where E is a nonempty intersection of union terms. Since $t \notin \alpha(FDNF_{y_j})$, there is a constraint of the above form such that $t \notin \alpha(\bigcup_k f_{jk}(\bar{u}_{jk}))$. By Axiom 6, φ contains a constraint $x \subseteq \bigcup_i f_i(\bar{u}_i) \cup \bigcup_j \bigcup_k f_{jk}(\bar{u}_{jk})$ such that t does not belong to the value of the right-hand side of the inclusion under α. Hence, $t \notin \alpha(x)$.

The proof for the constraints $u \subseteq f^{-1}_{(k)}(x)$ is similar. If $t \notin \alpha(f^{-1}_{(k)}(x))$, then, by the definition of projection, for all trees $t_1, \ldots, t_n$ such that $t_k = t$ and n is the arity of f, $f(t_1, \ldots, t_n) \notin \alpha(x)$. Let $FDNF_x = \bigcup_i f_i(\bar{q}_i)$. Then, for all $t_1, \ldots, t_n$ as above, $f(t_1, \ldots, t_n) \notin \alpha(\bigcup_{\{i \mid f=f_i, \mathcal{L}(\mathcal{A}(f_i(\bar{q}_i)))\neq\emptyset\}} f(\bar{q}_i))$. Hence, $t \notin \alpha(\bigcup_{\{i \mid f=f_i, \mathcal{L}(\mathcal{A}(f_i(\bar{q}_i)))\neq\emptyset\}} q_{i,k})$. By Axiom 7 and Lemma 2, φ contains a sequence of constraints equivalent to $u \subseteq \bigcup_{\{i \mid f=f_i, \mathcal{L}(\mathcal{A}(f_i(\bar{q}_i)))\neq\emptyset\}} q_{i,k}$. Hence, $t \notin \alpha(u)$. □

Theorem 2 The greatest solution of a constraint φ is the greatest solution of the solved form $(\varphi^C)^S$ of the closed form φ^C of φ, where φ^C is the smallest constraint containing all consequences of φ under the axioms in Table 1.

Proof. By Proposition 1 and by the fact that the greatest solution assigns the set of all trees T_Σ to $\top$ and the empty set $\emptyset$ to $\bot$, φ implies φ^C. By Lemma 1, φ^C implies $(\varphi^C)^S$. By Proposition 2, $(\varphi^C)^S$ implies φ^C. Since φ^C contains φ, it implies φ. □

7 Complexity

For an input φ of size n, the terms that occur in the derived consequences are either variables in $\mathcal{V}(\varphi) \cup \{\top, \bot\}$ or flat terms built up from these variables and function symbols in Σ. The number of such terms is bounded by $|\Sigma| \cdot (n+2)^k + n + 2$ where k is the maximal arity of a function symbol in Σ. Each inclusion occurring in φ^C involves one variable on the left-hand side and a union of flat terms. The size of such an inclusion is polynomial in n. Thus, the number of these inclusions is at most exponential (precisely $O(c_1^n)$ for some constant c_1 which depends on Σ). At each iteration of the fixpoint algorithm, the consequences of all (pairwise combinations of) inclusions under Axioms 1–6 are computed. This amounts to a cost of $O((c_1^n)^3)$. Adding consequences of Axioms 7 and 8 is done in exponential time (say, $O(c_2^n)$) by the lemma below and by the polynomial time complexity of the emptiness test for tree automata (also in the case of Büchi tree automata [33]); it may be repeated $O(c_1^n)$ times. Hence, the whole algorithm runs in $O(c_1^{3n} + (c_1 c_2)^n)$.

Lemma 4 For any intersection x, $FDNF_x$ can be computed in exponential time.

Proof. To compute $FDNF(E)$, we check, for all $f \in \Sigma$ and all sequences $(S_1, \ldots, S_{a(f)})$ such that $S_i \subseteq \mathcal{V}(E)$ for $i = 1, \ldots, a(f)$, if the condition from the definition of $FDNF$ is satisfied. The size of $\mathcal{V}(E)$ is at most n, so the number of terms $f(\bigcap S_1, \ldots, \bigcap S_{a(f)})$ is bounded by $|\Sigma|(2^n)^k|\Sigma|$, where k is the maximal arity of a symbol in Σ. Hence, the number of these terms is bounded by c_4^n for some constant c_4. To check the condition, we have to run through the constraints $x \subseteq \bigcup_i f_i(\bar{u}_i)$. The number of such constraints is bounded by c_1^n, and for each such constraint we can do the job in polynomial time. Therefore, the whole procedure is exponential. □

Theorem 3 The problem of the emptiness of the greatest solution of a constraint φ is DEXPTIME-complete.

Proof. The upper bound is shown above. The lower bound follows by the reduction of the problem of the emptiness of the intersection of tree automata [13]. □

Since intersection corresponds to conjunction, one can expect the DEXPTIME lower bound for every formalism of set constraints that can express regular sets of trees.

8 Satisfiability over path-closed sets

Due to reasons of space, we will give only an outline of our algorithm for testing the satisfiability of set constraints in a different class of set constraints (which we call *path-closed set constraints*). The class is syntactically larger; terms $f(x_1, \ldots, x_n)$ may occur also on the left-hand side of an inclusion (union on the left-hand side is trivial). The interpretation is now over path-closed sets of trees. More precisely, a valuation α satisfies the inclusion $E \subseteq E'$ between two expressions E and E' if and only if $PC(\alpha(E)) \subseteq PC(\alpha(E'))$. Here, $PC(X)$ denotes the *path-closure* of the set X of trees, *i.e.*, the smallest path-closed set of trees containing X. A set X is path-closed if it is recognized by a deterministic top-down tree automaton (infinite state if X is not regular) [14].

We can use the class of path-closed set constraints for our set-based analysis if we exclude the empty set from the interpretation domain.[5] Then, the greatest solution always exists (which is not true if we admit the empty set; take $f(x, y) \subseteq \bot$). In the framework presented in Section 4, we then define simply $\Phi(\tau \subseteq x) = \tau \subseteq x$. Then, the emptiness test is replaced by the satisfiability test in its role for the analysis. The new analysis is less accurate, but has the same complexity characterization (Theorem 4).

[5]The complexity of the satisfiability test does not change if we admit the empty set. Applying the equivalence:

$$f(x_1, \ldots, x_n) \subseteq \tau \Leftrightarrow (x_1 \subseteq \bot) \vee \ldots \vee (x_n \subseteq \bot) \vee (x_1 \not\subseteq \bot \wedge \ldots \wedge x_n \not\subseteq \bot \wedge f(x_1, \ldots, x_n) \subseteq \tau)$$

to all constraints of the form $f(x_1, \ldots, x_n) \subseteq \tau$ gives an exponential number of constraints, each of which can be solved in exponential time; thus, the whole algorithm is single exponential.

The algorithm for solving these constraints runs in three steps: First, the constraints of the form $f(x_1, \ldots, x_n) \subseteq \tau$ with $n > 0$ are replaced by $x_1 \subseteq f^{-1}_{(1)}(\tau) \wedge \ldots \wedge x_n \subseteq f^{-1}_{(n)}(\tau)$ and the constraints $a \subseteq \tau$ are forgotten. Second, the greatest solution of the obtained constraint is computed and non-emptiness is tested. Here, we modify our previous algorithm. We apply the rule

$$\bigcup_i f(\bar{u}_i) = f(\bigcup_i u_{i,1}, \ldots, \bigcup_i u_{i,n})$$

for function symbols of arity n. Thus we obtain always a deterministic top-down tree automata. Third, for each constraint of the form $a \subseteq \tau$ the membership of a in the value of τ is tested. The correctness of the algorithm follows from the following lemma:

Lemma 5 In the interpretation over path-closed sets defined above, the formula $f(x_1, \ldots, x_n) \subseteq \tau$ and the formula $x_1 \subseteq f^{-1}_{(1)}(\tau) \wedge \ldots \wedge x_n \subseteq f^{-1}_{(n)}(\tau)$ are equivalent.

Proof. For the one direction, we assume $\alpha(f(x_1, \ldots, x_n)) \subseteq \alpha(\tau)$. We prove that for any $i = 1, \ldots, n$, $\alpha(x_i) \subseteq \alpha(f^{-1}_{(i)}(\tau))$. Take any tree $t_i \in \alpha(x_i)$. We need to find trees $t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n$ such that $f(t_1, \ldots, t_n) \in \alpha(\tau)$. This is trivial from the non-emptiness of $\alpha(x_1), \ldots, \alpha(x_n)$.

For the other direction, we assume $\alpha(x_i) \subseteq \alpha(f^{-1}_{(i)}(\tau))$ for all i. We prove $\alpha(f(x_1, \ldots, x_n)) \subseteq PC(\alpha(\tau))$. Take any $f(t_1, \ldots, t_n) \in \alpha(f(x_1, \ldots, x_n))$. We know that $t_i \in \alpha(x_i)$, so $t_i \in \alpha(f^{-1}_{(i)}(\tau))$. By the definition of projection, there exist trees $t_i^1, \ldots, t_i^{i-1}, t_i^{i+1}, \ldots, t_i^n$ for all i such that $f(t_i^1, \ldots, t_i, \ldots, t_i^n) \in \alpha(\tau)$. Then, $f(t_1, \ldots, t_n) \in PC(\alpha(\tau))$ holds. $\square$

Theorem 4 The satisfiability problem of path-closed set constraints is DEXPTIME-complete.

Proof. We have shown the upper bound. The lower bound follows from Seidl's characterization of the problem of the emptiness of the intersection for deterministic top-down tree automata [30]. $\square$

9 Conclusion

Motivated by a concrete application as well as by the general interest in constraint-solving problems for set constraints, we have formulated a set-based analysis of logic programs with respect to their greatest model semantics, and we have given the complexity-theoretic characterization of the constraint-solving problem of the corresponding class of set constraints.

One question arising from this work and the work by Cousot and Cousot in [12] is whether this set-based analysis is again an instance of an abstract interpretation, *i.e.*, whether our constraint-solving process can be modeled as the iteration of an abstraction of the T_P fixpoint operator.

Kozen has given an equational axiomatization of the algebra of sets of trees in [20]. It would be useful to modify this axiomatization in order to account for the projection operator (or for path-closeness) and thus fix the algebraic laws underlying our algorithm.

As required by our application, we account also for sets of infinite trees. For the first time, automata over infinite trees have been used to represent solutions of set constraints. The represented sets of infinite trees appear in the μ-level in the hierarchy of the fixpoint calculus of Niwiński [27]. The essential difference between the fixpoint expressions on the μ-level and our set constraints formalisms seems to be the projection operator; for the addition of intersection to the fixpoint expressions see [7]. The question arises whether the formalism of set constraints can be extended to have solutions in all levels, *i.e.*, to be able to express all Rabin-recognizable sets. This is related to the addition of fixpoint operators as in [24] (there, however, not over infinite trees but arbitrary first-order domains).

References

[1] A. Aiken, D. Kozen, M. Vardi, and E. L. Wimmers. The complexity of set constraints. In *1993 Conference on Computer Science Logic*, LNCS 832, pages 1–17. Springer-Verlag, Sept. 1993.

[2] A. Aiken, D. Kozen, and E. L. Wimmers. Decidability of systems of set constraints with negative constraints. *Information and Computation*, 122(1):30–44, Oct. 1995.

[3] A. Aiken and B. Murphy. Static type inference in a dynamically typed language. In *Eighteenth Annual ACM Symposium on Principles of Programming Languages*, pages 279–290, January 1991.

[4] A. Aiken and E. L. Wimmers. Solving systems of set constraints (extended abstract). In *Seventh Annual IEEE Symposium on Logic in Computer Science*, pages 329–340, 1992.

[5] A. Aiken, E. L. Wimmers, and T. Lakshman. Soft typing with conditional types. In *Twenty-First Annual ACM Symposium on Principles of Programming Languages*, Portland, Oregon, Jan. 1994.

[6] A. Arnold and M. Nivat. Formal computations of non deterministic recursive program schemes. *Mathematical Systems Theory*, 13:219–236, 1980.

[7] A. Arnold and D. Niwiński. Fixed point characterization of weak monadic logic definable sets of trees. In M. Nivat and A. Podelski, editors, *Tree Automata and Languages*, pages 159–188. North Holland, 1992.

[8] L. Bachmair, H. Ganzinger, and U. Waldmann. Set constraints are the monadic class. In *Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 75–83, 1993.

[9] W. Charatonik and L. Pacholski. Negative set constraints with equality. In *Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 128–136, 1994.

[10] W. Charatonik and L. Pacholski. Set constraints with projections are in NEXPTIME. In *Proceedings of the* 35^{th} *Symposium on Foundations of Computer Science*, pages 642–653, 1994.

[11] W. Charatonik and A. Podelski. Set constraints with intersection. submitted for publication. `http://www.mpi-sb.mpg.de/~podelski/papers/intersection.ps`.

[12] P. Cousot and R. Cousot. Formal language, grammar and set-constraint-based program analysis by abstract interpretation. In *Record of FPCA '95 - Conference on Functional Programming and Computer Architecture*, pages 170–181, La Jolla, California, USA, 25-28 June 1995. SIGPLAN/SIGARCH/WG2.8, ACM Press, New York, USA.

[13] T. Frühwirth, E. Shapiro, M. Vardi, and E. Yardeni. Logic programs as types for logic programs. In *Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 300–309, July 1991.

[14] F. Gécseg and M. Steinby. *Tree Automata*. Akademiai Kiado, 1984.

[15] R. Gilleron, S. Tison, and M. Tommasi. Solving systems of set constraints using tree automata. In *10th Annual Symposium on Theoretical Aspects of Computer Science*, LNCS 665, pages 505–514. Springer-Verlag, 1993.

[16] R. Gilleron, S. Tison, and M. Tommasi. Solving systems of set constraints with negated subset relationships. In *Proceedings of the* 34^{th} *Symp. on Foundations of Computer Science*, pages 372–380, 1993. A full version *Technical report IT 247, Laboratoire d'Informatique Fondamentale de Lille*.

[17] N. Heintze and J. Jaffar. A decision procedure for a class of set constraints (extended abstract). In *Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 42–51, 1990.

[18] N. Heintze and J. Jaffar. A finite presentation theorem for approximating logic programs. In *Seventeenth Annual ACM Symposium on Principles of Programming Languages*, pages 197–209, January 1990.

[19] N. D. Jones and S. S. Muchnick. Flow analysis and optimization of lisp-like structures. In *Sixth Annual ACM Symposium on Principles of Programming Languages*, pages 244–256, January 1979.

[20] D. Kozen. Logical aspects of set constraints. In *1993 Conference on Computer Science Logic*, LNCS 832, pages 175–188. Springer-Verlag, Sept. 1993.

[21] D. Kozen. Rational spaces and set constraints. *Theoretical Computer Science*, 167(1–2):73–94, 30 Oct. 1996.

[22] J. W. Lloyd. *Foundations of Logic Programming.* Symbolic Computation. Springer-Verlag, Berlin, Germany, second, extended edition, 1987.

[23] D. McAllester. Personal communication, August 1996.

[24] D. A. McAllester, R. Givan, C. Witty, and D. Kozen. Tarskian set constraints. In *Proceedings, 11*$^{\text{th}}$ *Annual IEEE Symposium on Logic in Computer Science*, pages 138–147, New Brunswick, New Jersey, July 1996. IEEE Computer Society Press.

[25] P. Mishra. Towards a theory of types in Prolog. In *IEEE International Symposium on Logic Programming*, pages 289–298, 1984.

[26] P. Mishra and U. Reddy. Declaration-free type checking. In *Twelfth Annual ACM Symposium on the Principles of Programming Languages*, pages 7–21, 1985.

[27] D. Niwiński. On fixed-point clones. In L. Kott, editor, *Proceedings of the 13th International Conference on Automata, Languages and Programming*, volume 226 of *Lecture Notes in Computer Science*, pages 464–473. Springer-Verlag, 1986.

[28] A. Podelski and W. Charatonik. Temporal properties of concurrent constraint programs. `http://www.mpi-sb.mpg.de/~podelski/papers/temporal.ps`, 1996.

[29] J. C. Reynolds. Automatic computation of data set definitions. *Information Processing*, 68:456–461, 1969.

[30] H. Seidl. Haskell overloading is DEXPTIME-complete. *Information Processing Letters*, 52:57–60, 1994.

[31] G. Smolka. The Oz Programming Model. In *Volume 1000 of Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[32] K. Stefansson. Systems of set constraints with negative constraints are *NEXPTIME*-complete. In *Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 137–141, 1994.

[33] M. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Sciences*, 32, 1986.

[34] J. Young and P. O'Keefe. Experience with a type evaluator. In D. Bjørner, A. P. Ershov, and N. D. Jones, editors, *Partial Evaluation and Mixed Computation*, pages 573–581. North-Holland, 1988.