Set constraints with intersection

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

Abstract

Set constraints are inclusions between expressions denoting sets of trees. The efficiency of their satisfiability test is a central issue in set-based program analysis, their main application domain. We introduce the class of set constraints with intersection *(the only operators forming the expressions are constructors and intersection) and show that its satisfiability problem is* DEXPTIME*-complete. The complexity characterization continues to hold for* negative set constraints with intersection *(which have positive and negated inclusions). We reduce the satisfiability problem for these constraints to one over the interpretation domain of nonempty sets of trees. Set constraints with intersection over the domain of nonempty sets of trees enjoy the fundamental property of* independence of negated conjuncts. *This allows us to handle each negated inclusion separately by the entailment algorithm that we devise. We furthermore prove that set constraints with intersection are equivalent to the class of* definite set constraints *and thereby settle the complexity question of the historically first class for which the decidability question was solved.*

1 Introduction

Set constraints. Set constraints denote relations between sets of trees. Syntactically, they are conjunctions of inclusions between expressions built over variables, constructors (constants and function symbols from a given alphabet) and a choice of set operators that defines the specific class of set constraints. The main application domain is set-based program analysis and type inference for functional, imperative and logic programming languages [3, 5, 21, 25, 35, 37, 42], but they are also used in order-sorted languages [41] and in constraint logic programming [28].

The satisfiability problems for the various classes of set constraints have been widely studied, in part for their significance for applications, in part for their own sake. Four different decidability proofs [4, 16, 8, 1] were given for the class of *positive set constraints*, three [17, 2, 10] for the class of *negative set constraints*, and one [11] for the class of *set constraints with projection* (which includes all above-mentioned classes). Set constraints were studied from the logical and topological point of view [27, 13, 29] and also in domains different from the Herbrand universe [19, 9, 34, 36, 12].

Definite set constraints. This was the first class of set constraints for which decidability was shown [20, 22]. It was introduced by Heintze and Jaffar and is used for the type analysis of Prolog programs [21, 18, 23]. The satisfiable constraints in this class have a least solution (this fact is at the origin of the attribute "definite"). Bachmair, Ganzinger and Waldmann [8] showed that the satisfiability problem for this class is in NEXPTIME. The relation of the class to other classes of set constraints has not been investigated before. Due to the the apparent difference in syntax, the two classes of definite and of positive set constraints were often (*e.g.*, in [23]) considered incomparable.

Our results. We show that definite set constraints can be seen as a proper subclass of positive set constraints, namely positive constraints without complement and union symbols. We call them *set constraints with intersection* (the only operators forming the expressions are constructors and intersection). We give an algorithm, in a concise presentation, which (1) performs the satisfiability test in exponential time, (2) computes which variables denote the empty set in the least solution, and (3) represents the least solution as a tree automaton. Points (2) and (3) are important for the application to program analysis. Our algorithm

*On leave from University of Wrocław, Poland. Partially supported by KBN grant 8 S503 022 07.

is the fixpoint iteration under the operator that adds immediate consequences under 9 logical axioms. Each consequence is an inclusion between two terms from a set of bounded size; this is used to show the termination and to derive the upper bound. Using a reduction from a tree automata problem (as in [15]), we give an exponential lower bound and thus obtain that the satisfiability problem is DEXPTIME-complete. This is the first time that a class of set constraints (over a general constructor alphabet) falls into this complexity class. Since the classes of definite set constraints and of set constraints with intersection are not only equivalent in expressive power but also linearly inter-reducible, we hereby settle also the complexity question for definite set constraints.

We also give an exponential satisfiability algorithm for the class of *negative set constraints with intersection* which is obtained from (positive) set constraints with intersection by allowing inclusions to be negated. The idea that underlies the general schema of this algorithm is based on the following observation: Every variable appearing in the left-hand term of a negated inclusion is *nonempty* (*i.e.*, must have a nonempty value in every solution). The first part of the algorithm infers the set of nonempty variables in a negative set constraint with intersection, say γ (by fixpoint iteration wrt. a second set of axioms). Roughly, one part of γ is satisfied by any valuation assigning nonempty sets to nonempty variables and the empty set to the other variables. All variables in the other part, γ_{ne}, are nonempty. We have thus reduced the satisfiability problem for γ to the one for γ_{ne} over the interpretation domain of nonempty sets of trees. We establish that set constraints with intersection over the domain of nonempty sets of trees enjoy the property of *independence of negated conjuncts*.[1] This theorem allows us to handle each negated inclusion separately by the entailment algorithm that we devise (as the fixpoint iteration wrt. a third set of axioms). The correctness of the theorem depends crucially on the assumption that the constructor alphabet is infinite. The interpretation of this assumption might be that the constructor alphabet is never fully known, or is always extensible (which fits with the modularity in program analysis).

The independence property is a fundamental property for constraint systems that is of further-reaching interest. Colmerauer has introduced it for the manipulation of inequations in the context of constraint logic programming (CLP) [14]. The property also characterizes (and is characterized by) the semantics of bottom-up and top-down computations [32]. A general study of the property shows its importance in various symbolic computation areas [31]. In constraint data bases, the property allows the efficient containment test between constraint relations [26]. The property is necessary for the inference of constrained functional dependencies in polynomial time [33]. Examples of constraint systems with the independence property are: universal closures of the definite Horn clauses of a logic programming language [31], term equations over finite or infinite trees [14], linear equations over the real numbers [30], various constraint classes over feature trees [7, 6, 39], infinite Boolean algebras with positive constraints [24], and various simple subclasses of constraints (*e.g.*, inequations over numbers where each variable always appears on the same side of the inequation) which may be useful for the application considered in [33].

[1] For comparison, any class of set constraints over the domain of possibly empty sets of trees does not have the independence property: In the constraint $f(x,y) \subseteq 0 \wedge x \not\subseteq 0 \wedge y \not\subseteq 0$, the conjuncts $x \not\subseteq 0$ and $y \not\subseteq 0$ are *not* independent from each other.

Related work. It is not clear whether one should compare the algorithm given by Heintze and Jaffar [20, 22] with ours (the first one, for positive set constraints with intersection) since it takes a different input than ours, *i.e.*, a class of set constraints with different (in fact, more) set operators. Also, their main motivation then was to solve the decidability question of the satisfiability problem. The main difficulty with their algorithm was, in fact, to prove the termination; they do not give its complexity. Their algorithm works by stepwise transformations of tree grammars followed by a separate satisfiability test. Their syntax shares with ours the use of *intersection variables* which, roughly, stand for intersection terms and thus make the syntax homogeneous. They have not considered negative definite set constraints.

Frühwirth, Shapiro, Vardi and Yardeni [15] characterize the membership problem in the least model of a logic program in a subclass of so-called *proper unary-predicate programs* by the same complexity. Their lower bound proof is similar to ours (and, in fact, inspired it); their algorithm is based on 2-way automata transformations. Both, definite set constraints and proper programs, are used as approximations of logic programs. Definite set constraints can express the empty set of solutions (*i.e.*, they can be inconsistent), whereas proper programs can not. Every definite set constraint *in solved form* can be transformed directly into an equivalent proper program. We do not see, however, how a direct transformation for the case of *general* (satisfiable) positive definite set constraints would be possible. One obstacle is, for example, the fact that $f(\bar{x}) \subseteq f(\bar{y}) \wedge \varphi$ is generally not equivalent to $\bar{x} \subseteq \bar{y} \wedge \varphi$. The question arises whether our algorithm (the first one) may replace theirs. This would be the case if every proper program could be transformed directly into an equivalent definite set constraint. It would be interesting (but not in the scope of this paper) to explore whether such a transformation exists.[2]

In [12] we prove the independence property for *inclusion constraints over nonempty sets* of trees (the constraints use no other set operators than the constructors; *i.e.*, they consist of inclusions between Herbrand terms). There, we can give a polynomial entailment algorithm and thus show that the satisfiability problem for negative inclusion constraints over nonempty sets is polynomial. We did then not have the idea of solving the satisfiability problem over the standard interpretation domain of possibly empty sets by reducing it to an interpretation domain where the independence property holds. In this paper, we develop the techniques of the independence proof in [12] further in order to account for the intersection operator. In particular, in the proof of Lemma 1, we give an explicit construction of the lower bounds t_x^p that refers to tree-paths p and to the intersection of all upper bounds τ of the variable x (*i.e.*, terms τ such that $x \subseteq \tau$ occurs).

The only class of set constraints (over the standard interpretation domain of possibly empty sets) that has been considered previously for the incorporation of negation is the one where all Boolean set operators (intersection, union and complement) may be used. The techniques used for solving the corresponding satisfiability problem in the three approaches [17, 2, 10] (based on reductions to problems over tree automata, hypergraphs and the monadic class, respectively) are different from each other and are all incomparable with the technique presented in this paper.

2 Preliminaries

We define that *set constraints with intersection* are conjunctions of inclusions between terms built up with variables, constructors and intersection. The terms can be of arbitrary depth. For the sake of simplicity of presentation, however, we restrict ourselves to flat terms. In Section 4 we show that this restriction does not affect the generality and the complexity measure.

We assume given a ranked alphabet Σ fixing the arity $n \geq 0$ of its function symbols $f, g, a, b, \ldots$ and infinite set $\mathcal{V}$ of variables $x, y, z, u, v, w, \ldots$. We use $\mathcal{V}(E)$ for the set of variables contained in the expression E. We use $t, s, \ldots$ as meta-variables for finite trees (*i.e.*, ground terms), $\tau, \tau_1, \tau_2, \ldots$ for terms of depth ≤ 1, $\theta, \theta_1, \theta_2$ for intersections of terms of depth ≤ 1, and φ for positive constraints and γ for negative constraints (conjunctions of positive and negated inclusions).

$$
\begin{array}{lcl}
\tau & ::= & x \mid f(\bar{u}) \\
\theta & ::= & \tau \mid \tau_1 \cap \ldots \cap \tau_n \\
\varphi & ::= & \theta_1 \subseteq \theta_2 \mid \varphi_1 \wedge \varphi_2 \\
\gamma & ::= & \varphi \mid \theta_1 \not\subseteq \theta_2 \mid \gamma_1 \wedge \gamma_2
\end{array}
$$

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 denote $\mathsf{Terms}(\varphi)$ the set of all flat terms τ (*i.e.*, terms without intersection) occurring in φ. We introduce a fresh variable x_S for each set $S \subseteq \mathsf{Terms}(\varphi)$. We call such a variable an *intersection variable.* If we use the notation x_S we implicitly assume that $S \subseteq \mathsf{Terms}(\varphi)$. The algorithm and the axioms in Table 1 do not refer to constraints φ directly but to constraints φ^R representing φ. The constraints φ^R contain neither intersection nor variables from φ (but only flat terms over intersection variables x_S).

Definition 1 (representation) The representation of a constraint φ is the set φ^R of inclusions of the form $x_S \subseteq x_{S'}$ which we obtain by first replacing in φ maximal terms of the form $\tau_1 \cap \ldots \cap \tau_n$ by $x_{\{\tau_1, \ldots, \tau_n\}}$, and then replacing all remaining terms τ by $x_{\{\tau\}}$.

Example 1 The representation of the constraint

$$
\varphi = \{ \begin{array}{l} u \subseteq a, \\ f(u,v) \cap v \subseteq w, \\ u \cap w \subseteq f(u,v) \ \} \end{array}
$$

is the constraint over intersection variables

$$
\varphi^R = \{ \begin{array}{l} x_{\{u\}} \subseteq x_{\{a\}}, \\ x_{\{f(u,v),v\}} \subseteq x_{\{w\}}, \\ x_{\{u,w\}} \subseteq x_{\{f(u,v)\}} \ \}. \end{array}
$$

The closed form of φ (defined below) will then contain constraints like $a \subseteq x_{\{a\}}$ and $x_{\{u,w\}} \subseteq f(x_{\{u\}}, x_{\{v\}})$.

Our interpretation domain is the set of all sets of finite trees over the ranked alphabet Σ. A valuation is a mapping assigning sets of finite trees to variables.

[2] The framework of Heintze and Jaffar [23] does not yet give such a transformation: The analysis that applied on proper programs yields an accurate approximation involves so-called quantified set expressions. The analysis with definite set constraints given in [23] is less accurate. There, for example, the proper program $P \equiv$ `p(f(xp, a)). q(f(xq, b)) : -p(f(xq, b)).` is non-accurately approximated by the definite set constraint $\mathcal{K}_P \equiv x_p \supseteq 1 \wedge x_q \supseteq 1$ (where 1 denotes the set of all trees).

1. the relation $\subseteq$ is reflexive and transitive
2. $x_S \subseteq x_{\{\tau\}}$ for $\tau \in S$
3. $f(x_{\{u_1\}}, \ldots, x_{\{u_n\}}) \subseteq x_{\{f(u_1,\ldots,u_n)\}}$
4. $x_{\{f(u_1,\ldots,u_n)\}} \subseteq f(x_{\{u_1\}}, \ldots, x_{\{u_n\}})$
5. $\left.\begin{array}{l} f(x_{S_1}, \ldots, x_{S_n}) \subseteq x_S \\ f(x_{S'_1}, \ldots, x_{S'_n}) \subseteq x_{S'} \end{array}\right\} \rightarrow$ $f(x_{S_1 \cup S'_1}, \ldots, x_{S_n \cup S'_n}) \subseteq x_{S \cup S'}$
6. $\mathsf{nonempty}(\tau), \tau \subseteq \tau' \rightarrow \mathsf{nonempty}(\tau')$
7. $\mathsf{nonempty}(\bar{u}) \rightarrow \mathsf{nonempty}(f(\bar{u}))$
8. $\mathsf{nonempty}(f(\bar{u})), f(\bar{u}) \subseteq f(\bar{v}) \rightarrow \bar{u} \subseteq \bar{v}$
9. $\mathsf{nonempty}(\tau), \tau \subseteq f(\bar{u}), \tau \subseteq g(\bar{v}) \rightarrow \mathit{false}$

Table 1. Axioms for positive constraints φ over intersection variables x_S where $S \subseteq \mathsf{Terms}(\varphi)$

Each valuation α can be extended in a canonical way to a mapping α from terms to sets of finite trees.

$$\alpha(f(\tau_1, \ldots, \tau_n)) = \{f(t_1, \ldots, t_n) \mid t_i \in \alpha(\tau_i) \text{ for } i = 1, \ldots, n\}$$
$$\alpha(\tau_1 \cap \ldots \cap \tau_n) = \alpha(\tau_1) \cap \ldots \cap \alpha(\tau_n)$$

Definition 2 ($\cap$-compatibility) We say that a valuation α is $\cap$-compatible if

$$\alpha(x_S) = \bigcap_{\tau \in S} \alpha(\tau)$$

for all intersection variables x_S.

A valuation α satisfies the constraint φ if for all conjuncts $\theta_1 \subseteq \theta_2$ in φ we have $\alpha(\theta_1) \subseteq \alpha(\theta_2)$. If φ is a constraint over intersection variables then α must also be $\cap$-compatible to satisfy φ. We say that φ is satisfiable if it has a solution (*i.e.*, a valuation satisfying φ).

3 Satisfiability

Each axiom in Table 1 translates to a basic fact. Axiom 2: Each set contains its intersection with other sets. Axioms 3 and 4: The two representations of the set $f(\bar{u})$ that are possible in φ^R are equal. In particular, for $n = 0$, Axiom 3 yields $a \subseteq x_{\{a\}}$ for all constant symbols $a \in \Sigma$. Axiom 5: Given two inclusions, the intersection of the two smaller sets is smaller than intersection of the two larger sets. Axioms 6 and 7 are used to infer the non-emptiness of terms. The expression $\mathsf{nonempty}(\bar{u})$ in the hypothesis of Axiom 7 stands for the conjunction $\mathsf{nonempty}(u_1) \wedge \ldots \wedge \mathsf{nonempty}(u_k)$ where k is the arity of the function symbol f in the conclusion. We obtain the empty conjunction, *i.e.*, *true*, in the special case where $k = 0$, *i.e.*, f is a constant symbol. This non-emptiness information is needed in the Axiom 8, the decomposition rule. Note that if one of the variables in $\bar{u}$ is empty, then $f(\bar{u})$ is the empty set regardless of the other variables' values; then $f(\bar{u}) \subseteq f(\bar{v})$ is satisfied, while $\bar{u} \subseteq \bar{v}$ may not be satisfied. Axiom 9: If each tree in some nonempty set starts with two different function symbols then the constraint is not satisfiable.

Proposition 1 The axioms in Table 1 are valid under all $\cap$-compatible interpretations over the domain of sets of trees.

Proof. The proof is done by inspection of each axiom. □

We next define two notions of normal form for constraints φ. The first one, φ^C, is the outcome of the algorithm (applied on the representation of φ), from which the second one, φ^S, extracts the least solution (if a solution exists).

Definition 3 (closed form) The closed form φ^C of a constraint φ is the smallest set of constraints which contains φ^R and is closed under consequences of axioms in Table 1.

Definition 4 (solved form) The solved form φ^S of a constraint φ is the subset of φ^C of all inclusions $f(\bar{u}) \subseteq x$ between a nonempty term and a variable, *i.e.*,

$$\varphi^S = \{f(\bar{u}) \subseteq x \in \varphi^C \mid \mathsf{nonempty}(f(\bar{u})) \in \varphi^C\}.$$

The solved form φ^S is essentially a representation of a tree automaton, where variables correspond to states and constraints to transitions. It has a canonical solution α, which is its least one; the value of the variables that do not occur in φ^S is the empty set, and the values of all others can be defined by the least fixed point solution of the corresponding regular system of equations

$$x = \bigcup_{f(\bar{u}) \subseteq x \in \varphi^S} f(\bar{u})$$

for each variable x occurring in φ^S.

Theorem 1 (satisfiability) A constraint φ is satisfiable if and only if its closed form φ^C does not contain *false*.

Proof. The "only if" direction is clear from Proposition 1. For the other direction, let φ^S be the solved form of φ and α its least solution. We now prove that α defines a solution of φ^C (Claim 1), that the denotation of the intersection variables is the expected one (Claim 2), and, finally, that we obtain a solution β of the constraint φ by setting $\beta(u) = \alpha(x_{\{u\}})$ for all variables u that occur in φ (Claim 3). In the following we will say that a term τ is nonempty if $\mathsf{nonempty}(\tau) \in \varphi^C$, otherwise we say that τ is empty.

Claim 1 The valuation α satisfies all constraints in φ^C.

Proof. There are three possible forms of constraints in $\varphi^C - \varphi^S$: $x \subseteq \tau$ with empty variable x, $f(\bar{u}) \subseteq \tau$ with empty $f(\bar{u})$, and $x \subseteq \tau$ with nonempty x. In the first case, by Axiom 6, the variable x does not occur in φ^S, so $\alpha(x) = \emptyset$ and the constraint is trivially satisfied. In the second case, by Axiom 7, there is an empty variable u_i in $\bar{u}$. From the previous case we know that $\alpha(u_i) = \emptyset$, so $\alpha(f(\bar{u})) = \emptyset$ and the constraint is satisfied. Below we show that α satisfies all constraints of the third form, that is $x \subseteq \tau$ with nonempty variable x. Again there are three possibilities:

- τ is a variable, say $\tau = y$. Since $x \subseteq y \in \varphi^C$, we have that for any constraint $\tau' \subseteq x \in \varphi^S$, by transitivity $\tau' \subseteq y \in \varphi^S$. By its definition, the valuation α satisfies $x \subseteq y$.

- τ is a constant symbol, say $\tau = a$. Since $x \subseteq a \in \varphi^C$ and *false* $\notin \varphi^C$, there are no constraints of the form $f(\bar{u}) \subseteq x$ in φ^C with nonempty $f(\bar{u})$ and f being a symbol different from a (otherwise, by Axioms 1 and 9, *false* $\in \varphi^C$). Hence, no term with leading symbol different from a belongs to $\alpha(x)$, and the valuation α satisfies $x \subseteq a$.

- τ is a composed term, say $\tau = f(x_1, \ldots, x_n)$. Take any tree $t \in \alpha(x)$. If t is of the form $g(t_1, \ldots, t_m)$, then there must be a constraint $g(u_1, \ldots, u_m) \subseteq x \in \varphi^S$ such that $g(\bar{u})$ is nonempty and $t_i \in \alpha(u_i)$ for all i. Now, if $g \neq f$, then again *false* $\in \varphi^C$. Hence $t = f(t_1, \ldots, t_n)$ and $f(u_1, \ldots, u_n) \subseteq x \in \varphi^S$. Axiom 8 yields $u_i \subseteq x_i \in \varphi^C$, and case $\tau = y$ above implies $t_i \in \alpha(x_i)$. It follows that $t \in \alpha(f(x_1 \ldots, x_n))$ holds. Thus, the valuation α satisfies $x \subseteq f(x_1, \ldots, x_n)$. $\square$

Claim 2 The valuation α is $\cap$-compatible.

Proof. The inclusion $\alpha(x_{\{\tau_1,\ldots,\tau_n\}}) \subseteq \alpha(x_{\{\tau_1\}}) \cap \ldots \cap \alpha(x_{\{\tau_n\}})$ follows directly from Axiom 2 and Claim 1.

For the other inclusion, we will prove that for any tree t and any two sets $S, S' \subseteq \mathsf{Terms}(\varphi)$, if $t \in \alpha(x_S)$ and $t \in \alpha(x_{S'})$ then $t \in \alpha(x_{S\cup S'})$. Then the claim follows directly.

The proof goes by induction on the structure of t. If t is a constant symbol, say $t = a$, then by the definition of α, $a \subseteq x_S \in \varphi^S$ and $a \subseteq x_{S'} \in \varphi^S$. By Axiom 5, $a \subseteq x_{S\cup S'} \in \varphi^S$ and we are done.

Now let $t = f(t_1, \ldots, t_n)$. By the definition of α, there exist constraints $f(x_{S_1}, \ldots, x_{S_n}) \subseteq x_S$ and $f(x_{S'_1}, \ldots, x_{S'_n}) \subseteq x_{S'}$ such that $t_i \in \alpha(x_{S_i})$ and $t_i \in \alpha(x_{S'_i})$ for $i = 1, \ldots, n$. The induction hypothesis yields $t_i \in \alpha(x_{S_i\cup S'_i})$ for $i = 1, \ldots, n$. Axiom 5 yields $f(x_{S_1\cup S'_1} \ldots, x_{S_n\cup S'_n}) \subseteq x_{S\cup S'} \in \varphi^C$. Since $t_i \in \alpha(x_{S_i\cup S'_i})$ for $i = 1, \ldots, n$, we know that $\alpha(x_{S_i\cup S'_i}) \neq \emptyset$. This implies $\mathsf{nonempty}(x_{S_i\cup S'_i}) \in \varphi^C$ by the definition of α. Thus, Axiom 7 yields $\mathsf{nonempty}(f(x_{S_1\cup S'_1}, \ldots, x_{S_n\cup S'_n})) \in \varphi^C$, and, therefore, $f(x_{S_1\cup S'_1}, \ldots, x_{S_n\cup S'_n}) \subseteq x_{S\cup S'} \in \varphi^S$. Hence, by the definition of α, $t = f(t_1, \ldots, t_n) \in \alpha(x_{S\cup S'})$. This completes the induction step. $\square$

Claim 3 Let β be a valuation such that $\beta(u) = \alpha(x_{\{u\}})$ for all variables u occurring in φ. Then β is a solution of φ.

Proof. By Axioms 3 and 4 we have $\beta(f(u_1, \ldots, u_n)) = \alpha(x_{\{f(u_1,\ldots,u_n)\}})$. By Claim 1, β satisfies all constraints $\tau_1 \subseteq \tau_2 \in \varphi$. By Claim 2, β satisfies all constraints of the form $\tau \subseteq \tau_1 \cap \ldots \cap \tau_n$ and $\tau_1 \cap \ldots \cap \tau_n \subseteq \tau$ in φ. This proves Claim 3 and completes the proof of Theorem 1. $\square$

The solution β of φ, which is induced by the least solution α of φ^S via the equality $\beta(u) = \alpha(x_{\{u\}})$, is the *least* solution of φ. A variable u has the empty set as value under β if and only if $x_{\{u\}}$ does not occur in φ^S.

4 Complexity

If the input constraint of size n contains nested expressions, we flatten it (in the standard way). We may hereby introduce new variables, but the total number of variables and the size of $\mathsf{Terms}(\varphi)$ remain linear in n.

We define the algorithm by fixed point iteration. We start with the representation φ^R of φ, as defined in Definition 1. At each iteration step, we add the direct consequences under the axioms in Table 1 of the set of constraints derived so far (we add $x \subseteq x$ only if x occurs). When the fixed point is reached, the obtained set φ^C is the closed form of φ, as defined in Definition 3. Hence, by Theorem 1, satisfiability is tested by checking if φ^C contains *false*. The solved form φ^S of

φ, as defined in Definition 4, gives the explicit representation of the least solution of φ.

Upper bound. How many iteration are possible before the fixed point is reached? For an input φ of size n, the size of $\mathsf{Terms}(\varphi)$ is bounded by cn for some constant c. Thus, we have to introduce 2^{cn} intersection variables. The terms that occur in φ^C are either any of these variables, or flat terms built up from these variables and function symbols from Σ. The number of such terms is bounded by $|\Sigma| \cdot (2^{cn})^k$ where k is the maximal arity of a function symbol in Σ. The constraints in the closed form of φ are either inclusions between pairs of terms, or non-emptiness constraints for terms. Hence there are at most $(|\Sigma|^2 + 1) \cdot (2^{cn})^{2k}$ constraints. The number of iterations is bounded by the number of constraints in φ^C.

How much time do we need in each iteration? The most costly operation is adding consequences of Axiom 5. This axiom involves $2k+2$ variables. Therefore, checking whether there are consequences under this axiom requires $O((2^{cn})^{2k+2})$ time. Hence, the whole algorithm can be done in $O(2^{c'n})$ time, where the constant c' depends only on the ranked alphabet Σ.

Lower bound. We prove the DEXPTIME-hardness of the satisfiability problem for set constraints with intersection by the reduction of the problem of the emptiness of the intersection of a sequence of tree automata, a known DEXPTIME-complete problem (see [15] and [38]).

Given a sequence of n bottom-up tree automata $A_1, \dots, A_n$, we introduce a fresh set variable q for each state q of each automaton. We assume wlog. that each tree automaton A_i has only one final state, which note q_{A_i}. We simulate each transition rule of the form $f(q_1, \dots, q_n) \to q$ by a set constraint $f(q_1, \dots, q_n) \subseteq q$. It is easy to see that the intersection of the languages recognized by the automata is empty if and only if the conjunction of these constraints together with $q_{A_1} \cap \dots \cap q_{A_n} \subseteq a \cap b$ is satisfiable, where a and b are two different constant symbols.

The following complexity characterization is an immediate consequence of the analysis above.

Theorem 2 The satisfiability problem for positive set constraints with intersection is DEXPTIME-complete.

$2'$. $x \subseteq x_S, x \subseteq x_{S'} \to x \subseteq x_{S \cup S'}$

$$5'. \quad \left. \begin{array}{l} x_S \subseteq f(x_{S_1}, \dots, x_{S_n}) \\ x_S \subseteq f(x_{S'_1}, \dots, x_{S'_n}) \end{array} \right\} \to$$
$$x_S \subseteq f(x_{S_1 \cup S'_1}, \dots, x_{S_n \cup S'_n})$$

$7'$. $\mathsf{nonempty}(f(\bar{u})) \to \mathsf{nonempty}(\bar{u})$

10. $\tau \not\subseteq \tau' \to \mathsf{nonempty}(\tau)$

11. $\mathsf{nonempty}(x),\ \varphi,\ y \subseteq x \to \mathit{false}$ if $x \overset{\varphi}{\leadsto}_p y$ for $p \neq \varepsilon$

Table 2. Additional axioms for C_2-closure of negative constraints γ over intersection variables x_S where $S \subseteq \mathsf{Terms}(\gamma)$

5 Entailment, independence, negation

The three notions that we consider in this section are closely connected with each other. A constraint system has the independence property if the constraints cannot express (*i.e.*, entail) disjunctions. Formally, given the constraints φ and $\varphi_1, \dots, \varphi_n$, the implication $\varphi \to \varphi_1 \vee \dots \vee \varphi_n$ is valid iff one of the n implications $\varphi \to \varphi_1, \dots, \varphi \to \varphi_n$ is valid. An equivalent formulation of the property states that the satisfiability problem of a conjunction with any number of negated constraints can be reduced to "independent" sub-problems with exactly one negated constraint. Namely, the conjunction $\varphi \wedge \neg\varphi_1 \wedge \dots \wedge \neg\varphi_n$ is satisfiable iff the n conjunctions $\varphi \wedge \neg\varphi_1, \dots, \varphi \wedge \neg\varphi_n$ are satisfiable. As a direct algorithmic consequence, the satisfiability problem of the conjunction of φ with n negated constraints can be reduced to n entailment problems (*i.e.*, the dual of the validity of the n implications $\varphi \to \varphi_1, \dots, \varphi \to \varphi_n$).

From now on we assume that the constructor alphabet is infinite.[3] Our approximation of the upper bound in Theorem 5 requires that the maximal arity of function symbols occurring in the constraint is bounded by a constant.

The formulation of the next results requires a series of technical definitions.

[3]The results in [10] might suggest that the problem becomes trivial under this assumption. This is not the case: The constraint $x \subseteq a, y_1 \subseteq f(x), y_2 \subseteq f(x), y_1 \not\subseteq \emptyset, y_2 \not\subseteq y_1$ is not satisfiable, while the corresponding monadic formula has a model.

12. $f(\bar{u}) \subseteq f(\bar{v}) \leftrightarrow \bar{u} \subseteq \bar{v}$
$f(\bar{u}) \subseteq f(\bar{t}) \leftrightarrow \bar{u} \subseteq \bar{t}$
$f(\bar{t}) \subseteq f(\bar{v}) \leftrightarrow \bar{t} \subseteq \bar{v}$ if $f(\bar{t})$ occurs in φ

13. $x \subseteq t \rightarrow t \subseteq x$

Table 3. Additional axioms for C_3-closure of positive constraints φ over nonempty intersection variables x_S where $S \subseteq \mathsf{Terms}(\varphi)$

Definition 5 (minimal upper bound) Given a positive constraint φ, we call $f(\bar{u})$ a proper upper bound of x if $x \subseteq f(\bar{u}) \in \varphi$. We say that $f(\bar{u})$ is a minimal upper bound of x if $\bar{u} \subseteq \bar{v} \in \varphi$ for each proper upper bound $f(\bar{v})$ of x.

If a constraint is closed under Axiom $5'$ in Table 2, then every variable that has a proper upper bound has a minimal upper bound.

Definition 6 (reachability $\overset{\varphi}{\rightsquigarrow}_p$) If x has no proper upper bound in φ then $x \overset{\varphi}{\rightsquigarrow}_\varepsilon x$. Otherwise, let $f(\bar{u})$ be the minimal upper bound of x. Then $x \overset{\varphi}{\rightsquigarrow}_{i.p} y$ if $u_i \overset{\varphi}{\rightsquigarrow}_p y$.

We next define three notions of closed forms of constraints, referring to axioms in Table 1, 2 and 3. Axiom 11 in Table 2 expresses an "occurs-check." For example, the constraint $\gamma = \{x \not\subseteq x', x \subseteq f(y), y \subseteq x\}$ is unsatisfiable.

The axioms in Table 3 introduce a new type of constraints, namely inclusions between finite trees t of arbitrary depth and terms τ. The restriction in the formulation of Axiom 12 is needed for algorithmic reasons. Without it, the closure of, for example, the constraint $a \subseteq x \wedge f(x) \subseteq x$ under the consequences of the axiom would be an infinite set of constraints. (This is not a problem in the case of upper bounds: A variable can have only one ground proper upper bound.) In order to understand Axiom 13, note that if α satisfies $x \subseteq t$ then $\alpha(x) = \{t\}$.

Definition 7 (closed forms γ^{C_2}, $\gamma^{C_2}_{ne}$, γ^{C_3}) The C_2-closed form γ^{C_2} of a constraint γ is the smallest set of constraints that contains γ^R and is closed under consequences of axioms in Tables 1 and 2.
The ne-C_2-closed form $\gamma^{C_2}_{ne}$ of a constraint γ is the (positive) constraint

$$\gamma^{C_2}_{ne} = \{\tau_1 \subseteq \tau_2 \mid \mathsf{nonempty}(\tau_1) \in \gamma^{C_2}\}.$$

The C_3-closed form γ^{C_3} of a constraint γ is the smallest set of constraints which contains $\gamma^{C_2}_{ne}$ and is closed under consequences of Axioms 1–5 in Table 1, Axioms $2'$ and $5'$ in Table 2, and Axioms 12 and 13 in Table 3.

The next lemma, which employs the notions of closed forms defined above, expresses a necessary condition for the entailment of a constraint $x \subseteq y$.

Lemma 1 Let φ be a positive constraint. If *false* $\notin \varphi^{C_2}$ then there exists a valuation α mapping variables in $\mathcal{V}(\varphi^{C_2}_{ne})$ to nonempty sets of trees that satisfies

$$\varphi^{C_2}_{ne} \wedge \bigwedge_{x \subseteq y \notin \varphi^{C_3}} x \not\subseteq y.$$

Proof. Since φ^{C_2} is closed under Axiom 11 in Table 2, the set $\{p \mid \exists y \;\; x \overset{\varphi}{\rightsquigarrow}_p y\}$ is finite for each nonempty variable x. For each such pair (x,p) we define a tree t^p_x. If $x \overset{\varphi}{\rightsquigarrow}_p y$ and $y \subseteq t \in \varphi^{C_3}$ for some tree t then we set $t^p_x = t$. Otherwise we define t^p_x by induction on the length of p, starting from the longest and then proceeding to shorter paths.

Let p be a maximal path such that $x \overset{\varphi}{\rightsquigarrow}_p y$ and $y \subseteq t \notin \varphi^{C_3}$ for any tree t. Let a^p_x be a fresh constant symbol (here we use the assumption that Σ is infinite) and put $t^p_x = a^p_x$.

Now suppose t^p_x is defined for all paths p of length n, and we define it for paths of length $n-1$. Let $x \overset{\varphi}{\rightsquigarrow}_p y$, and let $f(\bar{u})$ be the minimal upper bound of y. Then, for $i = 1, \ldots, n$ where $n = \mathit{arity}(f)$, we have $x \overset{\varphi}{\rightsquigarrow}_{p.i} u_i$ and $t^{p.i}_x$ is already defined. We put $t^p_x = f(t^{p.1}_x, \ldots, t^{p.n}_x)$.

We define a *lower bound completion* φ^{lbc} of φ as $\varphi^{C_3} \cup \{t^p_x \subseteq y \mid x \overset{\varphi}{\rightsquigarrow}_p y\}$. By construction φ^{lbc} is closed under all axioms relevant for C_3-completion, except transitivity.

We define α as the least solution of the system of equations

$$x = \bigcup_{f(\bar{u}) \subseteq x \in \varphi^{C_2}_{ne}} f(\bar{u}) \cup \bigcup_{t \subseteq y, y \subseteq x \in \varphi^{lbc}} t.$$

We note the following facts:

- If there exists a tree t such that $x \subseteq t \in \varphi^{C_2}$, then $\alpha(x) = \{t\}$; otherwise, t^ε_x does not occur in φ^{C_2}.
- If t^p_x does not occur in φ^{C_3}, then it occurs only once in φ^{lbc}.

  This is because $t^p_x \neq t^{p'}_{x'}$ for $(p,x) \neq (p',x')$.
- The valuation α satisfies $\varphi^{C_2}_{ne}$.

  The proof is analogous to the proof of Theorem 1 (Claims 1 and 2).

- $t_x^p \in \alpha(z)$ iff $y \subseteq z \in \varphi^{C_3}$ where $x \overset{\varphi}{\rightsquigarrow}_p y$.

  The proof goes by induction on the structure of t_x^p.

- The valuation α satisfies $\bigwedge_{x \subseteq y \notin \varphi^{C_3}} x \not\subseteq y$.

  This follows from the fact that $t_x^\varepsilon \in \alpha(y)$ iff $x \subseteq y \in \varphi^{C_3}$.

This summarizes the proof of Lemma 1. $\square$

Example 2 The lower bound completion φ^{lbc} of the constraint

$$\begin{aligned}\varphi = \{\ & x \subseteq g(u,v),\ u \subseteq f(z),\\ & v \subseteq f(z),\ f(z) \subseteq z,\\ & g(u,u) \subseteq y\}\end{aligned}$$

contains the constraints

$$\begin{aligned}& a^{1.1}_{x_{\{x\}}} \subseteq x_{\{z\}},\ a^{1.2}_{x_{\{x\}}} \subseteq x_{\{z\}},\\ & f(a^{1.1}_{x_{\{x\}}}) \subseteq x_{\{u\}},\ f(a^{1.2}_{x_{\{x\}}}) \subseteq x_{\{v\}},\\ & g(f(a^{1.1}_{x_{\{x\}}}), f(a^{1.2}_{x_{\{x\}}})) \subseteq x_{\{x\}}.\end{aligned}$$

Morover, we have that

$$\begin{aligned}& \alpha(x_{\{x\}}) = \{g(f(a^{1.1}_{x_{\{x\}}}), f(a^{1.2}_{x_{\{x\}}}))\},\\ & \alpha(x_{\{u\}}) \supset \{f(a^{1.1}_{x_{\{x\}}})\},\\ & \alpha(x_{\{v\}}) \supset \{f(a^{1.2}_{x_{\{x\}}})\},\\ & \alpha(x_{\{z\}}) \supset \{f^n(a^{1.1}_{x_{\{x\}}}), f^n(a^{1.2}_{x_{\{x\}}}) \mid n \in \mathcal{N}\},\\ & \alpha(x_{\{y\}}) \supset \{g(f(a^{1.1}_{x_{\{x\}}}), f(a^{1.1}_{x_{\{x\}}}))\}.\end{aligned}$$

We here write "$\supset$" because there are more lower bounds for these variables. Note that we have to use two different constants to obtain $t^\varepsilon_{x_{\{x\}}} \notin \alpha(x_{\{y\}})$.

We need the notions of entailment and independence for constraints over the domain of *nonempty* sets for the satisfiability test on negative constraints over the domain of possibly empty sets.

Definition 8 (entailment $\models_+$ over $\mathcal{P}^+(T_\Sigma)$) Given two positive constraints φ and φ', we write $\varphi \models_+ \varphi'$ if every valuation over the domain $\mathcal{P}^+(T_\Sigma)$ of nonempty sets of trees that satisfies φ also satisfies φ'.

From now on, we use the metavariables S and S' referring to sets of terms of the given constraint. Thus, $\bigcap S$ is another notation for a term θ.

The next result yields an algorithm for testing entailment.

Theorem 3 (entailment) If the positive constraint φ is satisfiable over the domain $\mathcal{P}^+(T_\Sigma)$, and if we set $\varphi' = \varphi \cup \{\mathsf{nonempty}(x) \mid x \in \mathcal{V}(\varphi)\}$, then

$$\varphi \models_+ \bigcap S \subseteq \bigcap S' \quad \text{if and only if} \quad x_S \subseteq x_{S'} \in (\varphi')^{C_3}.$$

Proof. By the validity of our axioms, *false* $\notin (\varphi')^{C_2}$. Using Lemma 1, the proof then follows the one of Claim 3 of Theorem 1. $\square$

We formulate the independence property as the independence of negated conjuncts in a constraint γ.

Theorem 4 (independence) If the positive constraint φ is satisfiable over $\mathcal{P}^+(T_\Sigma)$, then the constraint

$$\varphi \cup \bigcup_i \{\bigcap S_i \not\subseteq \bigcap S_i'\}$$

is satisfiable over $\mathcal{P}^+(T_\Sigma)$ if and only if each constraint

$$\varphi \cup \{\bigcap S_i \not\subseteq \bigcap S_i'\}$$

is satisfiable over $\mathcal{P}^+(T_\Sigma)$ for every i.

Proof. If $\varphi \cup \{\bigcap S_i \not\subseteq \bigcap S_i'\}$ is satisfiable for all i, then, by Theorem 3, $x_{S_i} \subseteq x_{S_i'} \notin (\varphi')^{C_3}$. By Lemma 1, there exists a valuation α for which the corresponding valuation (defined as in Claim 3 of Theorem 1) satisfies $\varphi \cup \bigcup_i \{\bigcap S_i \not\subseteq \bigcap S_i'\}$. $\square$

The following characterization of the satisfiability of negative constraints γ is based on a decomposition of γ and the independence property. Together with Theorem 3 it yields a satisfiability test for γ.

Theorem 5 (negation) Given the constraint

$$\gamma = \varphi \cup \bigcup_i \{\bigcap S_i \not\subseteq \bigcap S_i'\},$$

we set $\mathcal{N}(\gamma) = \{v \in \mathcal{V}(\gamma) \mid \mathsf{nonempty}(x_{\{v\}}) \in \gamma^{C_2}\}$ and

$$\varphi_{ne} = \{\theta_1 \subseteq \theta_2 \in \varphi \mid \mathcal{V}(\theta_1 \subseteq \theta_2) \subseteq \mathcal{N}(\gamma)\}.$$

Then γ is satisfiable if and only if

$$\varphi_{ne} \not\models_+ \bigcap S_i \subseteq \bigcap S_i'$$

for all i such that $x_{S_i'}$ is nonempty in γ^{C_2}.

Proof. The "only if" direction of the proof is obvious. For the "if" direction, suppose $\varphi_{ne} \not\models_+ \bigcap S_i \not\subseteq \bigcap S_i'$ for all i. By the independence theorem,

$$\gamma_{ne} = \varphi_{ne} \cup \bigcup_i \{\bigcap S_i \not\subseteq \bigcap S_i'\}$$

is satisfiable over the domain of nonempty sets. If β is a solution of this constraint, we update it to a solution of γ (over the original interpretation domain) by putting $\beta(v) = \emptyset$ for all $v \in \mathcal{V}(\gamma) - \mathcal{N}(\gamma)$. $\square$

Theorem 6 The satisfiability problem for negative set constraints with intersection is DEXPTIME-complete.

Proof. We apply the complexity analysis of Section 4 to the new sets of axioms. $\square$

6 Definite set constraints

In this section we show that set constraints with intersection are equivalent in expressive power with definite set constraints and that both classes are linearly inter-reducible.

We need to introduce the set operation of projection. The projection $f_{(i)}^{-1}(S)$ of a set S on the i-th position with respect to the constructor symbol f is the set of all subtrees of those trees in S that are labeled with f.

$$f_{(i)}^{-1}(S) = \{t_i \mid f(t_1, \ldots, t_i, \ldots, t_n) \in S\}$$

Definite set constraints are conjunctions of inclusions of the form $exp_1 \subseteq exp_2$, where the left-hand side exp_1 is built up using set variables, constants, function applications, projections, set union and intersection, while the right-hand side exp_2 is built up using set variables, constants and function applications only.[4]

To definite constraints. Constraints that are in the language of set constraints with intersection and not in the language of definite set constraints contain an inclusion with intersection on its right-hand side. Nested occurrences of intersection may be replaced by the introduction of new variables. Unnested occurrences of intersection on the right-hand side of an inclusion may be removed according to the rule $exp \subseteq exp_1 \cap exp_2$ iff $exp \subseteq exp_1$ and $exp \subseteq exp_2$.

From definite constraints. We have to remove all occurrences of unions and projections on the left-hand sides of inclusions. Removing unions on the left-hand side is analogous to removing intersections on the right-hand side. The remaining problem is to remove projections. Again, nested occurrences of projection may be replaced by the introduction of new variables. We remove unnested occurrences of projection using the equivalence

$$f_{(i)}^{-1}(X) \subseteq Y \;\text{ iff }\; X \cap f(\top, \ldots \top) \subseteq f(\top, \ldots Y, \ldots \top)$$

where Y occurs on the i-th position of the term $f(\top, \ldots, Y, \ldots, \top)$. Here, $\top$ denotes the set of all terms. The symbol $\top$ is not in the signature, but it is sufficient to add the conjunction of the inclusions $f(\top, \ldots, \top) \subseteq \top$ and $x \subseteq \top$ for each function symbol $f \in \Sigma$ and each variable x that occurs in the constraint.

The following characterization follows from the above reductions and Theorem 1. It continues to hold for negative definite set constraints.

[4] In [20, 21] the relation $\supseteq$ is used instead of $\subseteq$ and left-hand and right-hand sides are interchanged.

Theorem 7 The satisfiability problem for definite set constraints is DEXPTIME-complete.

7 Conclusion

We have singled out a class of set constraints by a simple condition on the syntax. We have given an algorithm, in a concise representation, which performs the satisfiability test in exponential time. The algorithm computes which variables denote the empty set in the least solution and presents the least solution as a tree automaton, which is important for its use in program analysis. We have proved the DEXPTIME-completeness of the satisfiability problem and thus given the first such complexity characterization for a general class of set constraints (*i.e.*, over a general ranked alphabet). We have shown the equivalence between this class and the class of definite set constraints regarding expressiveness and complexity. We could thus settle the complexity question of that class. We have then incorporated negation into our class and have shown that the complexity of the satisfiability problem remains the same. We have built on previous work of ours to prove the independence property for our class of set constraints if the empty set is excluded as a value. We give an entailment algorithm which works under the same condition. Starting from a simple observation, we have shown how one can reduce the original satisfiability problem to one where we can exploit the independence property and the entailment algorithm.

This work fits into a more general line of research. The satisfiability problem for several classes of set constraints is NEXPTIME-complete [8, 40, 10, 11]. The question arises for which natural and useful subclasses faster algorithms testing satisfiability exist. Aiken, Kozen, Vardi, and Wimmers [1] give a detailed analysis of the complexity of subclasses of positive set constrains which are obtained by restricting the ranked alphabet of constructor symbols (for example, positive set constraints over unary trees have a DEXPTIME-complete satisfiability problem). It is folklore that the problem can be solved in cubic time for *atomic set constraints*, which are positive set constraints without any set operation. It is trivial to see that the algorithm presented in this paper can be modified and combined with the polynomial entailment algorithm in [12] to yield a polynomial satisfiability test for *negative atomic set constraints.* Still nothing (except DEXPTIME-hardness) is known about *set constraints with union.* The problem is not dual to the one for set constraints with intersection (for example, even over the domain of nonempty sets of trees $f(\bar{x}) \subseteq f(\bar{y}) \cup f(\bar{z})$ is not equivalent to $\bar{x} \subseteq \bar{y} \cup \bar{z}$).

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] H. Aït-Kaci and A. Podelski. Entailment and disentailment of order-sorted feature constraints. In A. Voronkov, editor, *Proceedings of the Fourth International Conference on Logic Programming and Automated Reasoning*, Springer LNAI 698, pages 1–18. Springer-Verlag, July 1993.

[7] H. Aït-Kaci, A. Podelski, and G. Smolka. A feature-based constraint system for logic programming with entailment. *Theoretical Computer Science*, 122(1–2):263–283, Jan. 1994.

[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. Set constraints in some equational theories. In *1st International Conference Constraints in Computational Logics*, LNCS 845, pages 304–319. Springer-Verlag, 1994. Also to appear in *Information and Computation*.

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

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

[12] W. Charatonik and A. Podelski. The independence property of a class of set constraints. In *Conference on Principles and Practice of Constraint Programming*, LNCS 1118, pages 76–90. Springer-Verlag, 1996.

[13] A. Cheng and D. Kozen. A complete Gentzen-style axiomatization for set constraints. In *ICALP: Annual International Colloquium on Automata, Languages and Programming*, LNCS 1099, 1996.

[14] A. Colmerauer. Equations and inequations on finite and infinite trees. In *Proceedings of the 2nd International Conference on Fifth Generation Computer Systems*, pages 85–99, 1984.

[15] 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.

[16] 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.

[17] 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.*

[18] N. Heintze. Set based program analysis. PhD thesis, School of Computer Science, Carnegie Mellon University, 1992.

[19] N. Heintze. Set based analysis of arithmetic. Draft manuscript, July 1993.

[20] 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.

[21] 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.

[22] N. Heintze and J. Jaffar. A decision procedure for a class of set constraints. Technical report, School of Computer Science, Carnegie Mellon University, Feb. 1991. 42 pages.

[23] N. Heintze and J. Jaffar. Set constraints and set-based analysis. In *Proceedings of the Workshop on Principles and Practice of Constraint Programming*, LNCS 874, pages 281–298. Springer-Verlag, 1994.

[24] R. Helm, K. Marriott, and M. Odersky. Constraint-based query optimization for spatial databases. In *Tenth ACM Symposium on the Principles of Database Systems*, pages 181–191, Denver, CO, May 1991.

[25] 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.

[26] P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51:26–52, 1995. (Preliminary version in *Proc. 9th ACM PODS*, 299–313, 1990.).

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

[28] D. Kozen. Set constraints and logic programming (abstract). In *1st International Conference Constraints in Computational Logics*, LNCS 845. Springer-Verlag, 1994. Also to appear in *Information and Computation.*

[29] D. Kozen. Rational spaces and set constraints. In *TAPSOFT: 6th International Joint Conference on Theory and Practice of Software Development*, LNCS 915, pages 42–61. Springer-Verlag, 1995.

[30] J. Lassez and K. McAloon. Applications of a canonical form for generalized linear constraints. In *Proceedings of the International Conference on 5th Generation Computer Systems*, pages 703–710, Tokyo, Japan, Dec. 1988.

[31] J. L. Lassez and K. McAloon. A constraint sequent calculus. In *Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 52–61, June 1990.

[32] M. J. Maher. A logic programming view of CLP. In D. S. Warren, editor, *Proceedings of the 10th International Conference on Logic Programming*, pages 737–753, Budapest, Hungary, June 1993. The MIT Press.

[33] M. J. Maher. Constrained dependencies. In U. Montanari, editor, *Proceedings of the First International Conference on Principles and Practice of Constraint Programming (CP'95)*, Lecture Notes in Computer Science, pages 170–185, Cassis, France, 19–22 Sept. 1995. Springer-Verlag.

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

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

[36] M. Müller, J. Niehren, and A. Podelski. Inclusion constraints over non-empty sets of trees. To appear in M. Dauchet, editor, *Proceedings of the 9th International Joint Conference on Theory and Practice of Software Development (TAPSOFT)*, Springer LNCS, Springer-Verlag, April 1997.

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

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

[39] G. Smolka and R. Treinen. Records for Logic Programming. *Journal of Logic Programming*, 18(3):229–258, Apr. 1994.

[40] 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.

[41] T. E. Uribe. Sorted unification using set constraints. In *11th International Conference on Automated Deduction*, LNAI 607, pages 163–177. Springer-Verlag, 1992.

[42] 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.