Co-definite Set Constraints

Witold Charatonik* Andreas Podelski

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

Abstract. In this paper, we introduce the class of co-definite set constraints. This is a natural subclass of set constraints which, when satisfiable, have a *greatest* solution. It is practically motivated by the set-based analysis of logic programs with the greatest-model semantics. We present an algorithm solving co-definite set constraints and show that their satisfiability problem is DEXPTIME-complete.

1 Introduction

Set constraints and set-based analysis form an established research topic. It combines theoretical investigations ranging from expressiveness and decidability to program semantics and domain theory, with direct practical applications to type inference, optimization and verification of imperative, functional, logic and reactive programs (see [1,14,20] for overviews).

In set-based analysis, the problem of reasoning about runtime properties of programs is transferred to the problem of solving set constraints. The design of a system for a particular program analysis problem (for a particular class of programs) involves two steps: (1) single out a subclass of set constraints and devise an algorithm for solving set constraints in this subclass, and (2) define a mapping $P \mapsto \varphi_P$ from programs into this subclass and show the soundness of the abstraction of P by a distinguished solution of φ_P. The advantage with respect to other static-analysis methods is the common to all constraint-based approaches: the logical formulation of the problems allows for their classification and for the reuse of optimized implementations. It is thus important to classify the arising constraint-solving problems and devise algorithms for them.

In this paper, we define the subclass of *co-definite* set constraints. This is a natural subclass of set constraints which, when satisfiable, have a *greatest* solution. We present an algorithm solving co-definite set constraints in DEXPTIME. The algorithm involves some novel adaptations of standard techniques for solving set constraints to the new situation where the solutions range over sets of *infinite* trees and must be constructed by co-induction (and not by induction as with least solutions). We show how one can encode the problem of emptiness of intersection of tree automata in a direct way. Thus, the satisfiability problem is DEXPTIME-complete.

* On leave from Wrocław University. Partially supported by Polish KBN grant 8T11C02913.

The new class of co-definite set constraints is practically motivated by the set-based analysis of *reactive logic programs* (called perpetual processes in [16]). Their semantics is defined by the greatest fixpoint of the immediate consequence operator T_P, which at the same time is the greatest model. The semantics is defined not over finite but over infinite trees.[1] Our algorithm accounts for either case. In [21], we show that the greatest solution of the co-definite set constraint φ_P that we assign to the program P is larger than the greatest model of P. The error diagnosis for concurrent constraint programs (the static prediction of the inevitability of failure or deadlock), which is presented in [21], is based on that fact and employs the algorithm presented here.

Related work. Heintze and Jaffar [11,12] formulated the general problem of solving set constraints and gave the first decidability result for a subclass of set constraints which they called *definite*, for the reason that all satisfiable constraints in the class have a least solution. They have singled out this subclass for the analysis of logic programs with the (standard) least model semantics. The present authors [7] have recently characterized the complexity for this subclass (DEXPTIME). The general problem is NEXPTIME-complete [4,5].

Definite and co-definite set constraints are not dual with respect to their syntax. We must exclude constraints of the form $f(x,y) \subseteq f(a,a) \cup f(b,b)$ which do not have a greatest solution. They are also not dual with respect to the constraint solving problem (although the two complexity characterizations might suggest this). Although one can directly dualize the Boolean set operators and also the tree constructors, this is not the case for the projection operator. 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 [11] does therefore *not* compute the greatest solution. (The greatest solution of $x = f_{(1)}^{-1}(f(a,a))$ is $\{a\}$, but starting from this constraint that algorithm yields $x = \bot$ whose greatest solution is $\emptyset$. This is because $dual(f_{(1)}^{-1}(f(a,a))) = f_{(1)}^{-1}(dual(f(a,a))) = f_{(1)}^{-1}(\Omega_f \cup f(\Omega_a, \top) \cup f(\top, \Omega_a)) = \top$ and thus $x = dual(\top) = \bot$).

In definite set constraints, union is expressed via conjunction (e.g., $a \cup b \subseteq y$ by $a \subseteq y \wedge b \subseteq y$ and need not be dealt with explicitly. Co-definite set constraints employ union as an operator over terms, and conjunction introduces intersection additionally. The next example shows that our algorithm must combine ("multiply out") intersections of unions of terms. (How can this be done in single

[1] The reactive logic program $P \equiv p(f(x)) \leftrightarrow p(x)$ illustrates the difference between infinite and finite trees. When interpreted over *finite* trees, the greatest model is the empty set; otherwise, it is the singleton containing the infinite tree $f(f(f(\ldots)))$. In either case, the execution of the call of $p(x)$ does not fail. More generally, one can characterize finite failure by the greatest model in the case of infinite trees, but not in the case of finite trees. In [21] we use co-definite set constraints to approximate the greatest model; we have to interpret them over sets of *infinite trees* in order to apply this approximation to the prediction of finite failure of logic programs and of errors in concurrent constraint programs.

exponential time? - There are exponentially many union terms.) The co-definite set constraint

$$\begin{array}{l} y \subseteq f(a,c) \cup f(c,b) \wedge \\ y \subseteq f(a,c) \cup f(d,b) \end{array} \tag{1}$$

is satisfiable in conjunction with $a \subseteq f_{(1)}^{-1}(y)$ but unsatisfiable in conjunction with $b \subseteq f_{(2)}^{-1}(y)$. In the last case, however, the naive method would derive only $b \subseteq c \cup b$, which does not give unsatisfiability.

Analyzing logic programs with the *least* model semantics, Mishra [18] has used a class of set constraints with a non-standard interpretation over non-empty *path-closed* sets of finite trees, which also have a greatest solution. In that interpretation, $f(x,y) \subseteq f(a,a) \cup f(b,b)$ has a greatest solution (which assigns both variables x and y the set $\{a,b\}$). Heintze and Jaffar [13] have shown that Mishra's analysis is less accurate than theirs in two ways, due to the choice of the greatest solution and due to the choice of the non-standard interpretation, respectively. The choice of the non-standard interpretation over path-closed sets of trees is not traded with by a lower complexity. Our hardness proof for co-definite set constraints carries over to Mishra's set constraints. This is because the tree automata used in the reduction can be chosen deterministic [22].

2 Definitions

A (general) set expression e is built up by: variables, tree constructors, the Boolean set operators and the *projection* operator [11]. If e does not contain the complement operator, then e is called a *positive* set expression. A (general) set constraint is a conjunction of inclusions of the form $e \subseteq e'$.

Definition 1. A *co-definite* set constraint is a conjunction of inclusions $e_l \subseteq e_r$ between positive set expressions, where the set expressions e_l on the left-hand side of $\subseteq$ are furthermore restricted to contain only variables, constants, unary function symbols and the union operator (that is, no projection, intersection or function symbol of arity greater than one).

We assume given a ranked alphabet Σ fixing the arity $n \geq 0$ of its function symbols $f, g, \ldots$ and constant symbols $a, b, \ldots$, and an infinite set Var of variables $x, y, z, u, v, w, \ldots$. The formulations and results in this paper apply to either case: finite trees, or infinite trees. We then say simply *trees* and use the notation T_Σ. By infinite trees we mean trees, whose branches are infinite or finite.

We interpret set constraints over $\mathcal{P}(T_\Sigma)$, the domain of sets of trees over the signature Σ. That is, the values of variables are sets of trees, or: a valuation is a mapping $\alpha : \mathsf{Var} \rightarrow \mathcal{P}(T_\Sigma)$. Tree constructors are interpreted as functions over sets of trees: the constant a is interpreted as $\{a\}$, the function f applied to the sets $S_1, \ldots, S_n$ yields the set $\{f(t_1, \ldots, t_n) \mid t_1 \in S_1, \ldots, t_n \in S_n\}$. 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 $f_{(k)}^{-1}(S) = \{t \mid \exists t_1, \ldots t_n : t_k = t,\ f(t_1, \ldots, t_k, \ldots, t_n) \in S\}$.

The next remark (which is proven by checking all cases of possible inclusions) implies an important property: if a co-definite set constraint is satisfiable, then it has a greatest solution.

Remark 2. The solutions of co-definite set constraints are closed under arbitrary union. □

For the formal treatment, we will use co-definite set constraints in a restricted form, which we will simply call constraints.

Definition 3 (restricted syntax: constraints φ). A constraint φ is a co-definite set constraint in the syntax given below.

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

For better manipulation of constraints, we have added the symbol $\bot$, which is interpreted as the empty set and is the neutral element wrt. $\cup$. By convention, the empty union is $\bot$ (i.e., $\bigcup \emptyset = \bot$); similarly, $\bigcap \emptyset = \top$.

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 $\mathsf{Var}(E)$ for the set of variables contained in the expression E, and $\mathsf{Terms}(\varphi)$ for the sets of all *flat* terms τ (*i.e.*, without union) occurring in φ. We use $\Sigma(\varphi)$ for the set of all function symbols occurring in φ; this set is finite.[2]

Given a co-definite set constraint, we can transform it into an equivalent one of restricted syntax easily. We eliminate function and union symbols on the left-hand side by using the equivalences $f(e) \subseteq e'$ iff $e \subseteq f^{-1}_{(1)}(e')$ and $e_1 \cup e_2 \subseteq e$ iff $e_1 \subseteq e \wedge e_2 \subseteq e$. We flatten the terms on the right-hand side by replacing intersection with conjunction and by introducing a fresh variable for each subexpression occurring on the right-hand side of inclusions. Since we are interested in the greatest solution of the initial constraint, it is enough to write only one inclusion (instead of equality) between the new variable and the expression. For example, we replace the inclusion $x \subseteq f^{-1}_{(1)}(y_1 \cap y_2)$ by $x \subseteq f^{-1}_{(1)}(y) \wedge y \subseteq y_1 \wedge y \subseteq y_2$. The transformation does not change the complexity measure. The number of new variables is linear in the size of the initial constraint.

3 Algorithm

The algorithm for solving a constraint φ_0 computes the fixpoint under the operator that, applied to a constraint φ, adds the direct consequences of φ under the axioms given in Table 1 to φ. The control of the algorithm is presented in Table 2. Compared to the usual fixpoint saturation, the only difference is that in the

[2] We do not want to assume that the signature Σ is finite. This is important for the use of set constraints in (modular) program analysis: the constructor alphabet is never fully known, or is assumed to be extensible.

1.	$x \subseteq y \wedge y \subseteq z \to x \subseteq z$	
2.	$x \subseteq \tau_1 \cup y,\ y \subseteq \tau_2 \to x \subseteq \tau_1 \cup \tau_2$	
3.	$\gamma \wedge u \subseteq f^{-1}_{(k)}(x) \to \bigwedge_i u \subseteq \bigcup_j u_{ij}$	where $\bigcap_i \bigcup_j u_{ij} = f^{-1}_{(k)}(x, \gamma)$
4.	$\gamma \wedge u \subseteq f^{-1}_{(k)}(x) \to u \subseteq \bot$	if $f^{-1}_{(k)}(x, \gamma) = \bot$
5.	$a \subseteq x \wedge x \subseteq \bigcup_i f_i(\bar{u}_i) \to \mathit{false}$	if $a \neq f_i$ for all i
6.	$a \subseteq x \wedge x \subseteq \bot \to \mathit{false}$	

Table 1. Satisfiability-complete axiom scheme for constraints φ

case of Axioms 3 and 4, the operator adds only the direct consequences that are obtained by applications where the constraint γ is instantiated to φ (as opposed to: a subpart of φ). Applying the axioms to subparts of a constraint with, say, m conjuncts would amount to applying the axioms 2^m times. All applications to proper subparts are redundant. For example, $u \subseteq v$ could be inferred from $\varphi \equiv u \subseteq f^{-1}_{(1)}(x), x \subseteq f(v), x \subseteq a$ under Axiom 3; the algorithm infers only a stronger consequence $u \subseteq \bot$ by instantiating γ with φ. Note that conjunction $\wedge$ is idempotent; the conjunct $u \subseteq f^{-1}_{(k)}(x)$ in the axioms is, of course, instantiated to a conjunct of φ. Computing the expressions $f^{-1}_{(k)}(x, \gamma)$ in Axioms 3 and 4 is involved; we will discuss this in Section 3.3.

A constraint obtained as the fixpoint under the operator of the algorithm is in *closed form*, and φ^C is the closed form of φ. Note that, as we mentioned above, φ^C is not closed under all (possibly redundant) consequences under the axioms in Table 1.

We will next introduce automaton constraints ψ (Section 3.1). These form a subclass of co-definite set constraints which directly exhibit their greatest solution (Remark 7). We can construct, with each constraint φ, an automaton constraint $\Psi(\varphi)$ (Section 3.2). We use $\Psi(\varphi)$ for computing the expressions $f^{-1}_{(k)}(x, \gamma)$ in Axioms 3 and 4 (Section 3.3). (To give some intuition: As indicated by the example (1) in the introduction, we cannot apply the projection operator on terms τ directly but we have to first combine them and transform them into expressions with intersections below the function symbol. This leads us out of the restricted syntax of constraints φ.) Furthermore, if the constraint φ is in closed form then it has the same greatest solution as $\Psi(\varphi)$.

Before going into more detail, we summarize the main results of the paper.

Theorem 4. The algorithm in Table 2 computes the closed form φ^C of the input constraint φ in deterministic, single exponential time. The constraint φ is unsatisfiable if and only if φ^C contains *false*; otherwise, the greatest solution of φ is presented by $\Psi(\varphi^C)$.

Proof. See Propositions 17, 18 and 19 in Section 4. □

$\varphi := \varphi_0$
Repeat
  apply Axioms 1 and 2 to φ
  apply Axioms 3 and 4 to φ where γ is instantiated to φ
  apply Axioms 5 and 6 to φ
  add all direct direct consequences to φ
Until φ does not change or φ contains *false*
If φ contains *false*
  then "φ_0 is unsatisfiable"
  else "φ_0 is satisfiable" and $\varphi_0^C := \varphi$ ("φ is closed form of φ_0")

Table 2. Algorithm solving a constraint φ_0

Theorem 5. The satisfiability problem for co-definite set constraints is DEXPTIME-complete.

Proof. See Propositions 19 and 22 in Section 5. □

3.1 Automaton constraints ψ

We assume given a set q-Var of variables $q, q', \ldots$ which we want to distinguish from variables $x, y, \ldots$ in Var. Later we will take variables q that stand for intersections $x_1 \cap \ldots \cap x_k$ of variables $x_i \in \mathsf{Var}$.

Definition 6 (automaton constraint ψ). An automaton constraint ψ is a conjunction of the form $\psi \equiv \bigwedge_i q_i \subseteq E_i$ such that

- the variables q_i are pairwise different, and
- each expression E_i is either $\perp$ or of the form $\bigcup_j f_j(\bar{q}_j)$.

A variable q is *unbounded in* ψ if q is different from all q_i's on the left-hand side in ψ.

The interpretation of automaton constraints is as usual. A valuation is now a mapping from q-Var to sets. The next remark justifies the name automaton constraint.

Remark 7. The value of a variable q in the greatest solution of an automaton constraint ψ (in both cases of finite and infinite trees) is the language $\mathcal{L}(\mathcal{A}^\psi(q))$ of the top-down tree automaton $\mathcal{A}^\psi(q)$ constructed directly from ψ; in particular, the emptiness of the value of q can be tested in polynomial time.

We give the construction of the automata and the proof of the remark in the Appendix since we did not find it in the literature; it must, however, be folklore (cf. also [2]).

3.2 Constructing $\Psi(\varphi)$

Given a constraint φ, we can extract an automaton constraint $\Psi(\varphi)$ from φ which is equivalent to its subpart consisting of the conjuncts of the form $x \subseteq \bigcup_j f_j(\bar{u}_j)$. The variables q in $\Psi(\varphi)$ stand for intersections $x_1 \cap \ldots \cap x_n$ of variables $x_i \in \mathsf{Var}$. We note $\cap$-Var the set that these *intersection variables* q form. We use also $\bigcap S$ as another notation for q that stands for the intersection of the variables in $S \subseteq \mathsf{Var}$. The *proper upper bounds* τ of a variable x in φ are the terms of the form $\tau = \bigcup_j f_j(\bar{u}_j)$ such that $x \subseteq \tau$ lies in φ. Note that τ may be $\bot$.

We next define their combination, for variables x as well as for intersections q.

Definition 8 ($lub(x, \varphi)$, $lub(q, \varphi)$). The least upper bound of the variable x in the constraint φ is an intersection of terms τ,

$$lub(x, \varphi) = \bigcap \{ \bigcup_j f_j(\bar{u}_j) \mid x \subseteq \bigcup_j f_j(\bar{u}_j) \text{ lies in } \varphi \}.$$

The least upper bound of an intersection $q = x_1 \cap \ldots \cap x_n$ is $lub(q, \varphi) = \bigcap_{i=1}^{n} lub(x_i, \varphi)$.

If x does not have proper upper bounds in φ then $lub(x, \varphi) = \top$. Also, note that $\top \cap \ldots \cap \top = \top$ and $\tau \cap \top = \tau$.

The expression $E = lub(q, \varphi)$ is an intersection of unions of proper terms $f(\bar{u})$. We transform such an expression E into a union of terms $f(\bar{q})$ over intersections of variables q, hereby using a variant of the disjunctive normal form (the computation of the standard one would here require doubly-exponential time).

Definition 9 (FDNF). The *full disjunctive normal form* of $E = \bigcap_{i \in I} \bigcup_{j \in J_i} f_{ij}(\bar{u}_{ij})$ is a union of terms $f(\bar{q})$ over intersection variables q,

$$\begin{aligned} \mathit{FDNF}(E) = \bigcup \{ f(\bigcap S_1, \ldots, \bigcap S_n) \mid\ & f \in \Sigma,\ n = \text{arity}(f), \\ & S_1 \subseteq \mathsf{Var}(E), \ldots, S_n \subseteq \mathsf{Var}(E), \\ \forall i \in I\ \exists j \in J_i : f = f_{ij} \wedge\ & u_{ij,1} \in S_1 \wedge \ldots \wedge u_{ij,n} \in S_n \}. \end{aligned}$$

We set $\mathit{FDNF}(\top) = \top$ and $\mathit{FDNF}(\bot) = \bot$.

Example 10. If $E = (f(u, v_1) \cup f(u, v_2)) \cap f(u, u)$ then $\mathit{FDNF}(E)$ 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. Using the convention that $\bigcup \emptyset = \bot$, we take $E = a \cap b$ and have $\mathit{FDNF}(E) = \bot$.

Given a constraint φ, we note $\cap$-$\mathsf{Var}(\varphi)$ the set of all q standing for intersections $x_1 \cap \ldots \cap x_n$ of variables $x_i \in \mathsf{Var}(\varphi)$ occurring in φ. We now can give the construction of the automaton constraint $\Psi(\varphi)$ from the constraint φ.

Definition 11 ($\Psi(\varphi)$). The automaton constraint corresponding to the constraint φ is

$$\Psi(\varphi) \equiv \bigwedge_{q \in \cap\text{-}\mathsf{Var}(\varphi)} q \subseteq \mathit{FDNF}(lub(q, \varphi)).$$

We discard from $\Psi(\varphi)$ all inclusions of the form $q \subseteq \top$.

3.3 Projection $f^{-1}_{(k)}(x, \varphi)$

Given a conjunct $u \subseteq f^{-1}_{(k)}(x)$ in the constraint φ, and the (unique) expression E_x such that $x \subseteq E_x$ lies in $\Psi(\varphi)$, we want to express $f^{-1}_{(k)}(E_x)$ (the projection $f^{-1}_{(k)}$ applied to E_x) as an expression E_u such that we can add $u \subseteq E_u$ to φ.

Assume that E_x is of the form $E_x = f(q_1, \dots, q_n)$. Then one can infer $u \subseteq \bot$ if the value of at least one of $q_1, \dots, q_n$ is the empty set in the greatest solution of $\Psi(\varphi)$. This is the case if one of the automata $\mathcal{A}^{\Psi(\varphi)}(q_i)$ constructed from $\Psi(\varphi)$ recognizes the empty set. This again can be expressed as

$$\mathcal{L}(\mathcal{A}^{\Psi(\varphi)}(f(q_1, \dots, q_n))) = \emptyset \tag{2}$$

where we set $\mathcal{L}(\mathcal{A}^{\psi}(f(q_1, \dots, q_n))) = f(\mathcal{L}(\mathcal{A}^{\psi}(q_1)), \dots, \mathcal{L}(\mathcal{A}^{\psi}(q_n)))$. Otherwise (i.e., if the values of $q_1, \dots, q_n$ are all nonempty, and condition (2) does not hold), one can infer $u \subseteq q_k$.

In general, E_x is of the form $E_x = \bigcup_i f_i(q_{i1}, \dots, q_{in_i})$. Now, assume that $f(q_1, \dots, q_n)$ is a member of this union. If condition (2) is satisfied, then this member can be discarded from the union. Otherwise, q_k becomes a member of the union on the right-hand side of the infered constraint $u \subseteq q_k \cup \dots$.

Definition 12 (pre-$f^{-1}_{(k)}(E, \psi)$). The k-th *pre-projection* of f applied to an expression $E = \bigcup_i f_i(q_{i1}, \dots, q_{in_i})$ with respect to the automaton constraint ψ, is the union of intersections

$$\text{pre-}f^{-1}_{(k)}(E, \psi) = \bigcup\{q_{ik} \mid f = f_i, \mathcal{L}(\mathcal{A}^{\psi}(f_i(q_{i1}, \dots, q_{in_i}))) \neq \emptyset\}.$$

By applying the pre-projection we obtain expressions E such that the inclusions $u \subseteq E$ are not yet directly expressible in the restricted syntax of constraints φ. We can, however, transform a union of intersection variables into an intersection of unions using a variant of the conjunctive normal form (the computation of the standard one would here require doubly-exponential time). We then obtain an expression of the form $E' = \bigcap_i \bigcup_j u_{ij}$. We can express $u \subseteq E'$ as the conjunction $\bigwedge_i u \subseteq \bigcup_j u_{ij}$, which we then can add to φ, remaining within the restricted syntax of constraints.

Definition 13 (FCNF). The *full conjunctive normal form* of a union of intersection variables $E = \bigcup_{i \in I} \bigcap_{j \in J_i} u_{ij}$ is an intersection of unions of variables $x \in \mathsf{Var}$,

$$\mathit{FCNF}(E) = \bigcap\{\bigcup S \mid S \subseteq \mathsf{Var}(E),\ \forall i \in I\ \exists j \in J_i :\ \ u_{ij} \in S\}.$$

We set $\mathit{FCNF}(\bot) = \bot$.

Now we compose the operations above and obtain the full projection operation.

Definition 14 ($f^{-1}_{(k)}(x, \varphi)$). The k-th projection of $f \in \Sigma$ applied to the variable $x \in \mathsf{Var}$ wrt. to the constraint φ is an intersection of unions of variables $u_{ij} \in \mathsf{Var}$,

$$f^{-1}_{(k)}(x, \varphi) = \mathit{FCNF}(\text{pre-}f^{-1}_{(k)}(\mathit{FDNF}(\mathit{lub}(x, \varphi)), \Psi(\varphi))).$$

for all $q \in \cap\text{-}\mathsf{Var}(\varphi)$

$E_q \ := \ lub(q, \varphi)$	(Definition 8)
$E'_q \ := \ FDNF(E_q)$	(Definition 9)
$\Psi(\varphi) \ := \ \bigwedge_{q \in \cap\text{-}\mathsf{Var}(\varphi)} q \subseteq E'_q$	(Definition 11)
construct transition table of automata $\mathcal{A}^{\Psi(\varphi)}(q)$	*(same for all q; Definition 23 in Appendix)*
for all $q \in \cap\text{-}\mathsf{Var}(\varphi)$	
test emptiness of $\mathcal{L}(\mathcal{A}^{\Psi(\varphi)}(q))$	(Remark 7)
for all inclusions $u \subseteq f^{-1}_{(k)}(x)$ in φ	
$E^x_{(k)} \ := \ \mathsf{pre}\text{-}f^{-1}_{(k)}(E'_x, \Psi(\varphi))$	(Definition 12)
$f^{-1}_{(k)}(x, \varphi) \ := \ FCNF(E^x_{(k)})$	(Definition 13)

Table 3. Subprocedure computing $f^{-1}_{(k)}(x, \varphi)$ for all inclusions $u \subseteq f^{-1}_{(k)}(x)$ in constraint φ

Given a constraint φ, we compute the projections $f^{-1}_{(k)}(x, \varphi)$ simultaneously for all variables x such that an inclusion $u \subseteq f^{-1}_{(k)}(x)$ exists in φ. The corresponding subprocedure is presented in Table 3.

4 Correctness of the algorithm

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

Lemma 15 (FDNF). For any expression E of the form $\bigcap_{i \in I} \bigcup_{j \in J_i} f_{ij}(\bar{u}_{ij})$, 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, $\alpha(\mathit{FDNF}(E)) = \alpha(F)$. 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 $j_i \in J_i$ such that $f_{ij_i} = f$ and $\bar{u}_{ij_i} \in \overline{S}$. Hence, for all $k = 1, \ldots, \text{arity}(f)$, 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. The expression $\bigcap_{i \in I} f(\bar{u}_{ij_i}))$ occurs in the disjunctive normal form of E and $\alpha(f(\overline{\bigcap S})) = \alpha(\bigcap_{i \in I} f(\bar{u}_{ij_i}))$. □

Lemma 16 (FCNF). For any expression E of the form $\bigcup_{i \in I} \bigcap_{j \in J_i} u_{ij}$, the equality $\alpha(E) = \alpha(\mathit{FCNF}(E))$ holds for every valuation α.

Proof. The proof is similar to the proof of Lemma 15; 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. □

Proposition 17 (Soundness). The axioms in Table 1 are valid. In particular, if a constraint φ is satisfiable then its closed form φ^C does not contain *false*.

Proof. The proof is done by inspection of each axiom. The validity of Axioms 3 and 4 follows from consecutive applications of Lemma 15, Remark 7, and Lemma 16. □

Proposition 18 (Completeness). If the closed form φ^C of a constraint φ does not contain *false* then φ is satisfiable. Moreover, the greatest solution of φ is the greatest solution of the automaton constraint $\Psi(\varphi^C)$.

Proof. Let α be the valuation defined by $\alpha(x) = \mathcal{L}(\mathcal{A}(x))$, where $\mathcal{A}(x)$ is the automaton corresponding to $\Psi(\varphi^C)$ and the variable x. By Remark 7, the unique extension of α to $\cap\text{-}\mathsf{Var}(\varphi)$ is the greatest solution of $\Psi(\varphi^C)$. Below we show that α satisfies each conjunct in φ. Since φ implies $\Psi(\varphi^C)$, this will show that α is the greatest solution of φ.

Since $\mathit{FDNF}(\mathit{lub}(x, \varphi^C))$ is equivalent to an intersection of the expressions $\bigcup_i f_i(\bar{u}_i)$, the conjuncts of the form $x \subseteq \bigcup_i f_i(\bar{u}_i)$ are trivially satisfied.

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 $t \notin \alpha(x)$. Since $t \notin \alpha(y_j)$ for all j, the variables y_j cannot be unbounded in $\Psi(\varphi^C)$. Hence, every variable y_j occurs in a constraint of the form $y_j \subseteq \bigcup_k f_{jk}(\bar{u}_{jk})$ in φ^C (which includes the case of empty union $y_j \subseteq \bot$). Since $t \notin \alpha(\mathit{FDNF}(\mathit{lub}(y_j, \varphi^C)))$ for all j, there is a constraint of the above form in φ^C such that $t \notin \alpha(\bigcup_k f_{jk}(\bar{u}_{jk}))$. By Axiom 2, φ^C 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 $\mathit{FDNF}(\mathit{lub}(x, \varphi^C)) = \bigcup_i f_i(\bar{q}_i)$. 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 Axioms 3,4 and Lemma 16, φ^C 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, $t \notin \alpha(u)$.

The last case are the constraints of the form $a \subseteq x$. Again, if $a \notin \alpha(x)$ then x is bounded in $\Psi(\varphi^C)$ and a does not occur in $\mathit{FDNF}(\mathit{lub}(x, \varphi^C))$. But then, by Axiom 5 or 6, $\mathit{false} \in \varphi^C$, which is a contradiction. □

5 Complexity

Proposition 19 (upper bound). The algorithm in Table 2 computes the closed form φ^C of the input constraint φ in deterministic, single exponential time.

Proof. For an input φ of size n, the number of flat terms and variables that occur in φ is bounded by n. Each derived inclusion involves a variable in $\mathcal{V}(\varphi)$ on the left-hand side and a union of variables and flat terms on the right-hand side. All these flat terms occur in φ. Thus, the number of derived inclusions is bounded by $n \cdot 2^n$. At each iteration of the algorithm, the consequences of all (pairwise combinations of) inclusions under Axioms 1–2 are computed. This amounts to a cost of $O((n2^n)^2)$. Adding consequences of Axioms 3 and 4 is done in exponential time (say, $O(2^{n^c})$) by the lemmas below and by the polynomial time complexity of the emptiness test for tree automata (also in the case of Büchi tree automata [24]). There may be at most $n2^n$ iterations. Adding consequences of Axioms 5 and 6 costs at most $n2^n$, since the number of inclusions $a \subseteq x$ is bounded by n and number of inclusions with x on the left-hand side is bounded by 2^n. Hence, the whole algorithm runs in time $O(((n2^n)^2 + 2^{n^c}) \cdot n2^n + n2^n)$.

Lemma 20. For any intersection q, $\mathit{FDNF}(\mathit{lub}(q, \varphi))$ can be computed in time exponential in the size of φ.

Proof. Let $E = \mathit{lub}(q, \varphi)$ and n be the size of φ. To compute $\mathit{FDNF}(E)$, we check, for all $f \in \Sigma(\varphi)$ and all sequences $(S_1, \dots, S_{a(f)})$ such that $S_i \subseteq \mathcal{V}(E)$ for $i = 1, \dots, a(f)$, 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 $f(\bigcap S_1, \dots, \bigcap S_{a(f)})$ is bounded by $|\Sigma(\varphi)|(2^n)^k$, where k is the maximal arity of a symbol in $\Sigma(\varphi)$. Hence, the number of these terms is bounded by 2^{n^c} for some constant c (note that $k < n$). To check the condition, we have to run through the constraints $x \subseteq \bigcup_j f_j(\bar{u}_j)$ such that x occurs in the intersection q. The number of such constraints is bounded by $n2^n$. For each such constraint, checking if there exists a j such that $f = f_j$ and $u_{j,1} \in S_1, \dots, u_{j,a(f)} \in S_{a(f)}$ can be done in time polynomial in the size of the constraint and the sequence $(S_1, \dots, S_{a(f)})$ (which is polynomial in n). Therefore, the whole procedure runs in time $O(2^{n^c} \cdot n2^n \cdot \mathit{poly}(n))$, which is single exponential. □

Lemma 21. For any expression $E = \bigcup_{i\in I} \bigcap_{j \in J_i} u_{ij}$, the expression $\mathit{FCNF}(E)$ can be computed in time exponential in the number of variables in $\mathcal{V}(E)$.

Proof. The proof is analogous to the proof of the lemma above. □

Proposition 22 (lower bound). The problem of the satisfiability of the co-definite set constraints is DEXPTIME-hard.

Proof. The proof follows by the reduction of the problem of the emptiness of the intersection of tree automata [9].[3] For given n tree automata, let $\varphi_1, \ldots, \varphi_n$ be the constraints bounding the variables $X_1, \ldots, X_n$ to the languages of the automata. Then, the constraint

$$a \subseteq f_{(1)}^{-1}(f(a, X_1 \cap \ldots \cap X_n))$$

is satisfiable if and only if the intersection of the languages is nonempty. □

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.

6 Conclusion

We have defined a class of set constraints which arises in program analysis and error diagnosis, and we have given the complexity-theoretic characterization of its constraint-solving problem. We have applied our techniques also to the already existing class of path-closed set constraints and characterized its complexity too.

We now need to refine the abstract fixpoint strategy of our algorithm in order to improve its practical efficiency. In succession to the technical report [6] on which this paper is based, Devienne, Talbot and Tison [8] have already given a strategy for our algorithm which can achieve an exponential speedup. Unfortunately, their setup relies on bottom-up tree automata (in bit-vector representation) and thus, as the authors point out, applies to the case of finite trees only. Our algorithm uses top-down tree automata and accounts for both cases (where, again, the case of infinite trees is the only relevant one for analyzing the operational semantics).

Kozen has given an equational axiomatization of the algebra of sets of trees in [15]. It would be useful to modify this axiomatization in order to account for the projection operator and thus fix the algebraic laws underlying our algorithm.

To our knowledge, this is the first time that automata over infinite trees have been used to represent solutions of set constraints. The represented sets of infinite trees appear in the ν-level in the hierarchy of the fixpoint calculus of Niwiński [19]. The essential difference between the fixpoint expressions on the ν-level and our set constraints formalisms seems to be the projection operator; for the addition of intersection to the fixpoint expressions see [3]. 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 [17] (there, however, not over infinite trees but arbitrary first-order domains).

[3] The lower bound requires that the signature contains at least two function symbols, one of them having arity ≥ 2.

Appendix: Automaton constraints and automata

A (finite non-deterministic top-down tree) automaton is a tuple $\mathcal{A} = \langle \Sigma', Q, \delta, q_{sfinit}, Q_\top \rangle$ consisting of its finite alphabet $\Sigma' \subseteq \Sigma$, finite set of states Q, (non-deterministic) transition function $\delta : Q \times \Sigma' \to \mathcal{P}(\overline{Q})$ (where $\overline{Q}$ stands for the set of all tuples over Q), initial state q_{sfinit} and the set $Q_\top$ of *all-accept* states. The tree automaton $\mathcal{A}$ *accepts* a tree t (or: t lies in the language $\mathcal{L}(\mathcal{A})$ *recognized* by $\mathcal{A}$) iff there exists a run of $\mathcal{A}$ on t; this acceptance condition works for finite as well as for infinite trees. In the case of infinite trees, the automaton corresponds to a Büchi tree automaton where all states are final states. In both cases of finite and infinite trees, the emptiness of the automaton can be tested in polynomial time [23]. A run of $\mathcal{A}$ on the tree t assigns to the root the initial state and to each node of t a state q such that: if t is labeled with the function symbol $f \in \Sigma'$ of arity k, then the states assigned to the k successor nodes form a tuple that lies in the set $\delta_\psi(q, f)$. If the label f is a constant symbol, then the set $\delta_\psi(q, f)$ must contain the empty tuple. If the state assigned to the node is an all-accept state, $q \in Q_\top$, then the successor nodes are assigned any states (whether the node label f lies in the alphabet Σ' or not).

Given an automaton constraint ψ, we first define the family of automata $\mathcal{A}^\psi(q)$ (one for each variable $q \in \mathsf{q\text{-}Var}$, all with the same transition table δ_ψ) and then show that it recognizes exactly the greatest solution of ψ.

Definition 23 ($\mathcal{A}^\psi(q)$). The *automaton $\mathcal{A}^\psi(q_0)$ corresponding to* the automaton constraint ψ and the variable $q_0 \in \mathsf{q\text{-}Var}(\psi)$ is the tuple $\langle \Sigma(\psi), \mathsf{Var}(\psi), \delta_\psi, q_0 \rangle$ where

- the alphabet is the set $\Sigma(\psi)$ of function symbols occurring in ψ;
- the states are the variables q occurring in ψ;
- the set $\delta_\psi(q, f)$, i.e., the transition function δ_ψ applied on a state q and a function symbol f, is
  - the set $\{\bar{q}_j \mid f_j = f\}$ if $q \subseteq \bigcup_j f_j(\bar{q}_j)$ is a conjunct in ψ (which is then unique),
  - the empty set $\emptyset$ if $q \subseteq \bot$ is a conjunct in ψ;
- the initial state is q_0;
- the all-accept states are the unbounded variables in ψ.

If the variable $q_0 \in \mathsf{q\text{-}Var}$ does not occur at all in ψ, then $\mathcal{A}^\psi(q_0)$ is the tuple $\langle \emptyset, \{q_0\}, \emptyset, q_0, \{q_0\} \rangle$ (an automaton accepting T_Σ).

It is clear that $\mathcal{L}(\mathcal{A}^\psi(q))$ is the empty set if $q \subseteq \bot$ is in ψ and the set T_Σ of all trees if q is unbounded in ψ. More generally, for both finite and infinite trees, the statement below holds.

Observation 1. The valuation $\alpha : q \mapsto \mathcal{L}(\mathcal{A}^\psi(q))$ is the greatest solution of the automaton constraint ψ.

Proof. We will first show that any solution β of ψ is smaller than the valuation α. We extend β to a mapping over all states of the automata by setting $\beta(\top) = T_\Sigma$. We will show that $\beta(q) \subseteq \alpha(q)$ for all states q. If $\beta(q)$ is empty, then the inclusion is trivially satisfied; otherwise, take any tree $t \in \beta(q)$. By induction of the depth of the positions p in t, we will construct a run of $\mathcal{A}^\psi(q)$ on t that satisfies the following invariant: If $\mathcal{A}^\psi(q)$ 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 q and $t \in \beta(q)$. Let $\mathcal{A}^\psi(q)$ 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 at the positions $p.1, \ldots, p.n$. If q' is $\top$ or an unbounded variable in ψ, then the automaton goes to the state $\top$ in all positions $p.1, \ldots, p.n$. Since $\mathcal{A}^\psi(\top)$ recognizes the set T_Σ of all trees, our invariant is satisfied. Now suppose q' is not unbounded. Since β is a solution, on the right-hand side of the inclusion constraining q in ψ must occur an expression of the form $f(q_1, \ldots, q_n)$, with $t|_p \in \beta(f(q_1, \ldots, q_n))$, that is, $t_i \in \beta(q_i)$ for $i = 1, \ldots, n$. But then, by the definition of $\mathcal{A}^\psi(q)$, $(q_1, \ldots, q_n) \in \delta_\psi(q', f)$. By taking this transition we satisfy the invariant and are thus able to extend the definition of the run to all positions in t. Hence, $t \in \alpha(q)$.

For the other direction of the proof, we will show that α satisfies every inclusion $q \subseteq E$ in ψ. Again, if $\mathcal{L}(\mathcal{A}^\psi(q))$ is empty, nothing is to show; otherwise, we take an element $t = f(t_1, \ldots, t_n)$ from $\mathcal{L}(\mathcal{A}^\psi(q))$. By the definition of $\mathcal{L}(\mathcal{A}^\psi(q))$, there exists a run of $\mathcal{A}^\psi(q)$ on t, starting from q. Let $(q, f, q_1, \ldots, q_n)$ be the first transition used in this run. By the definition of a run, there are runs on t_i starting from q_i, and, hence, $t_i \in \mathcal{L}(\mathcal{A}^\psi(q_i))$. That is, $t \in f(\mathcal{L}(\mathcal{A}^\psi(q_1)), \ldots, \mathcal{L}(\mathcal{A}^\psi(q_n))) = f(\alpha(q_1), \ldots, \alpha(q_n))$. By the definition of $\mathcal{A}^\psi(q)$, the expression E is of the form $E = f(q_1, \ldots, q_n) \cup E'$. Therefore, $t \in \alpha(E)$. $\square$

References

1. A. Aiken. Set constraints: Results, applications and future directions. In *Proceedings of the Workshop on Principles and Practice of Constraint Programming*, LNCS 874, pages 326–335. Springer-Verlag, 1994.
2. A. Arnold and M. Nivat. Formal computations of non deterministic recursive program schemes. *Mathematical Systems Theory*, 13:219–236, 1980.
3. 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.
4. 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.
5. 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.
6. W. Charatonik and A. Podelski. Set constraints for greatest models. Technical Report MPI-I-97-2-004, Max-Planck-Institut für Informatik, April 1997. www.mpi-sb.mpg.de/~podelski/papers/greatest.html.

7. W. Charatonik and A. Podelski. Set constraints with intersection. In G. Winskel, editor, *Twelfth Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 362–372. IEEE, June 1997.
8. P. Devienne, J.-M. Talbot, and S. Tison. Solving classes of set constraints with tree automata. Technical Report IT-303, Laboratoire d'Informatique Fondamentale de Lille, May 1997.
9. 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.
10. F. Gécseg and M. Steinby. *Tree Automata.* Akademiai Kiado, 1984.
11. 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.
12. 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.
13. N. Heintze and J. Jaffar. Semantic types for logic programs. In F. Pfenning, editor, *Types in Logic Programming*, pages 141–156. MIT Press, 1992.
14. 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.
15. D. Kozen. Logical aspects of set constraints. In *1993 Conference on Computer Science Logic*, LNCS 832, pages 175–188. Springer-Verlag, Sept. 1993.
16. J. W. Lloyd. *Foundations of Logic Programming.* Symbolic Computation. Springer-Verlag, Berlin, Germany, second, extended edition, 1987.
17. 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.
18. P. Mishra. Towards a theory of types in Prolog. In *IEEE International Symposium on Logic Programming*, pages 289–298, 1984.
19. 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.
20. L. Pacholski and A. Podelski. Set constraints - a pearl in research on constraints. In G. Smolka, editor, *Proceedings of the Third International Conference on Principles and Practice of Constraint Programming - CP97*, volume 1330 of *Springer LNCS*, Berlin, Germany, October 1997. Springer-Verlag.
21. A. Podelski, W. Charatonik, and M. Müller. Set-based error diagnosis of concurrent constraint programs. submitted for publication, 1997.
22. H. Seidl. Haskell overloading is DEXPTIME-complete. *Information Processing Letters*, 52:57–60, 1994.
23. W. Thomas. *Handbook of Theoretical Computer Science*, volume B, chapter Automata on Infinite Objects, pages 134–191. Elsevier, 1990.
24. M. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Sciences*, 32, 1986.