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 $\varphi_P$. 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.

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 \(\varphi_P\) 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 \(x = dual(f_{[-1]}^{-1}(f(a,a))) = f_{[-1]}^{-1}(dual(f(a,a))) = f_{[-1]}^{-1}(\Omega_f \cup f(\Omega_a, \top) \cup f(\top, \Omega_a)) = \top\) and thus \(x = dual(\top) = \bot\).

In definite set constraints, union is expressed via conjunction (e.g., \(a \cup b \subseteq y\) by \(a \subseteq y \land b \subseteq y\) 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 exponential time? - There are exponentially many union terms.) The co-definite set constraint

\[
\begin{align*}
y \subseteq f(a,c) \cup f(c,b) \\
y \subseteq f(a,c) \cup f(d,b)
\end{align*}
\]

is satisfiable in conjunction with \(a \subseteq f_{[-1]}^{-1}(y)\) but unsatisfiable in conjunction with \(b \subseteq f_{[-2]}^{-1}(y)\).

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.

\(^1\)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.
We show that 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]. We give an algorithm solving Mishra’s set constraints in exponential time for comparison and for completeness. Path-closed interpretations are a subtle issue which has to be dealt with carefully.

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 $e_1 \subseteq e_r$. We give an algorithm solving Mishra’s set constraints in exponential time for comparison and for completeness. Path-closed interpretations are a subtle issue which has to be dealt with carefully.

Definition 1 A co-definite set constraint is a conjunction of inclusions $e_1 \subseteq e_r$ 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 $\Sigma$ fixing the arity $n \geq 0$ of its function symbols $f, g, \ldots$ and constant symbols $a, b, \ldots$, and an infinite set $\var{\text{Var}}$ 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 $T_{\Sigma}$. We reserve $T_{\Sigma}^\infty$ for the set of infinite 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 $\Sigma$. That is, the values of variables are sets of trees, or: a valuation is a mapping $\alpha : \var{\text{Var}} \to \mathcal{P}(T_{\Sigma})$.

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 $\{f(t_1, \ldots, t_n) | t_1 \in S_1, \ldots, t_n \in S_n\}$. 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 $f_{(k)}^{-1}(S) = \{t | \exists t_1, \ldots, t_n : t_k = t, f(t_1, \ldots, t_k, \ldots, t_n) \in S\}$.

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 1 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 2 (restricted syntax: constraints $\varphi$) A constraint $\varphi$ is a co-definite set constraint in the syntax given below.

\[
\tau ::= x \mid f(a) \mid \tau_1 \cup \tau_2 \mid \bot
\]

\[
\varphi ::= a \subseteq x \mid x \subseteq \tau \mid x \subseteq f_{(k)}^{-1}(u) \mid \varphi_1 \land \varphi_2
\]

Since we can no longer express the empty set by $a \cap b$, we have added the symbol $\bot$, which is the neutral element wrt. $\cup$. By convention, the empty union is $\bot$ (i.e., $\emptyset \cup \emptyset = \bot$); similarly, $\bigcap \emptyset = \top$. 

3
1. \( x \subseteq y \land y \subseteq z \rightarrow x \subseteq z \)

2. \( x \subseteq \tau_1 \cup y \land y \subseteq \tau_2 \rightarrow x \subseteq \tau_1 \cup \tau_2 \)

3. \( \gamma \land u \subseteq f_{[k]}^{-1}(x) \rightarrow \bigwedge_i u_i \subseteq \bigcup_j u_{ij} \) where \( \bigcap_i \bigcup_j u_{ij} = f_{[k]}^{-1}(x, \gamma) \)

4. \( \gamma \land u \subseteq f_{[k]}^{-1}(x) \rightarrow u \subseteq \perp \) if \( f_{[k]}^{-1}(x, \gamma) = \perp \)

5. \( a \subseteq x \land x \subseteq \bigcup_i f_i(\bar{u}_i) \rightarrow \text{false} \)

6. \( a \subseteq x \land x \subseteq \perp \rightarrow \text{false} \)

---

Table 1: Satisfiability-complete axiom scheme for constraints \( \varphi \)

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 \( \text{Var}(E) \) for the set of variables contained in the expression \( E \), and \( \text{Terms}(\varphi) \) for the sets of all flat terms \( \tau \) (i.e., without union) occurring in \( \varphi \). We use \( \Sigma(\varphi) \) for the set of all function symbols occurring in \( \varphi \); 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 \( e \subseteq f^{-1}(e') \) and \( e_1 \cup e_2 \subseteq e \) iff \( e_1 \subseteq e \land e_2 \subseteq e \). 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 \( x \subseteq f_{[1]}^{-1}(y) \land y \subseteq y_1 \land y \subseteq y_2 \). 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 \( \varphi_0 \) computes the fixpoint under the operator that, applied to a constraint \( \varphi \), adds the direct consequences of \( \varphi \) under the axioms given in Table 1 to \( \varphi \). The algorithm is presented in Table 2. In the case of Axioms 3 and 4, the operator adds only the direct consequences that are obtained by applications where the constraint \( \gamma \) is instantiated to \( \varphi \) (as opposed to: a subpart of \( \varphi \)).\(^3\) Computing the expressions \( f_{[k]}^{-1}(x, \gamma) \)

\(^2\)We do not want to assume that the signature \( \Sigma \) 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.

\(^3\)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 \( \varphi \equiv u \subseteq f_{[1]}(x), x \subseteq f(v), x \subseteq a \) under Axiom 3; it is redundant wrt. the consequence \( u \subseteq \perp \) by
\( \varphi := \varphi_0 \)

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

Table 2: Algorithm solving a constraint \( \varphi_0 \)

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 \( \varphi^C \) is the closed form of \( \varphi \). Note that \( \varphi^C \) is not closed under all (possibly redundant) consequences under the axioms in Table 1.

We will next introduce automaton constraints \( \psi \) (Section 3.1). These form a subclass of co-definite set constraints which directly exhibit their greatest solution (Remark 2). We can construct, with each constraint \( \varphi \), an automaton constraint \( \Psi(\varphi) \) (Section 3.2). We use \( \Psi(\varphi) \) for computing the expressions \( \overset{-1}{f} \overbrace{\{x, \gamma\}} \) 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 \( \gamma \) 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 \( \varphi \).) Furthermore, if the constraint \( \varphi \) 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 this section.

**Theorem 1** The algorithm in Table 2 computes the closed form \( \varphi^C \) of the input constraint \( \varphi \) in single exponential time. The constraint \( \varphi \) is unsatisfiable if and only if \( \varphi^C \) contains false; otherwise, the greatest solution of \( \varphi \) is presented by \( \Psi(\varphi^C) \).

**Proof.** See Propositions 1, 2 and 3 in Section 4. \( \square \)

**Theorem 2** The satisfiability problem for co-definite set constraints is DEXPTIME-complete.

**Proof.** See Propositions 3 and 4 in Section 5. \( \square \)

### 3.1 Automaton constraints \( \psi \)

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 \( x_i \in \text{Var} \).

instantiating \( \gamma \) with \( \varphi \). Note that conjunction \( \wedge \) is idempotent; the conjuncts \( u \subseteq \overset{-1}{f} \overbrace{\{x\}} \) in the axioms is, of course, instantiated to a conjunct of \( \varphi \).
Definition 3 (automaton constraint \( \psi \)) An automaton constraint \( \psi \) 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 \( \bot \) or of the form \( \bigcup_j f_j(\bar{q}_j) \).

A variable \( q \) is unbounded in \( \psi \) if \( q \) is different from all \( q_i \)'s on the left-hand side in \( \psi \).

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

Remark 2 The value of a variable \( q \) in the greatest solution of an automaton constraint \( \psi \) is the language \( \mathcal{L}(A^\psi(q)) \) of the top-down tree automaton \( A^\psi(q) \) constructed directly from \( \psi \); 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 \( \varphi \), we can extract an automaton constraint \( \Psi(\varphi) \) from \( \varphi \) 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 \( x_i \in \text{Var} \). We note \( \cap\text{-Var} \) the set that these intersection variables \( q \) form. We use also \( \cap S \) as another notation for \( q \) that stands for the intersection of the variables in \( S \subseteq \text{Var} \). The proper upper bounds \( \tau \) of a variable \( x \) in \( \varphi \) are the terms of the form \( \tau = \bigcup_j f_j(\bar{n}_j) \) such that \( x \subseteq \tau \) lies in \( \varphi \). Note that \( \tau \) may be \( \bot \).

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

Definition 4 \( (\text{lub}(x, \varphi), \text{lub}(q, \varphi)) \) The least upper bound of the variable \( x \) in the constraint \( \varphi \) is an intersection of terms \( \tau \),

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

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

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

The expression \( E = \text{lub}(q, \varphi) \) 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 5 (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 \( q \),

\[
\text{FDNF}(E) = \bigcup\{ f(\bigcap S_1, \ldots, \bigcap S_n) \mid f \in \Sigma, \ n = \text{arity}(f), \ S_1 \subseteq \text{Var}(E), \ldots, S_n \subseteq \text{Var}(E), \forall i \in I \ \exists j \in J_i : f = f_{ij} \land u_{ij,1} \in S_1 \land \ldots \land u_{ij,n} \in S_n \}.
\]
for all $q \in \cap\text{-Var}(\varphi)$

$$E_q := \text{hub}(q, \varphi)$$  \hspace{1cm} (Definition 4)

$$E_q^\prime := \text{FDNF}(E_q)$$  \hspace{1cm} (Definition 5)

$$\Psi(\varphi) := \bigwedge_{q \in \cap\text{-Var}(\varphi)} q \subseteq E_q^\prime$$  \hspace{1cm} (Definition 6)

construct transition table of automata $A^{\Psi(\varphi)}(q)$ \hspace{1cm} (same for all $q$; Definition 10 in Appendix)

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

test emptiness of $L(A^{\Psi(\varphi)}(q))$ \hspace{1cm} (Remark 2)

for all inclusions $u \subseteq f_{[k]}^{-1}(x)$ in $\varphi$

$$E_{[k]}^x := \text{pre-}f_{[k]}^{-1}(E_x', \Psi(\varphi))$$  \hspace{1cm} (Definition 7)

$$f_{[k]}^{-1}(x, \varphi) := \text{FCNF}(E_{[k]}^x)$$  \hspace{1cm} (Definition 8)

Table 3: Subprocedure computing $f_{[k]}^{-1}(x, \varphi)$ for all inclusions $u \subseteq f_{[k]}^{-1}(x)$ in constraint $\varphi$

**Example 1** If $E = (f(u, v_1) \cup f(u, v_2)) \cap f(u, u)$ then $\text{FDNF}(E)$ 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 $\bigcup \emptyset$, we take $E = a \cap b$ and have $\text{FDNF}(E) = \emptyset$. If $E = \top$ then $\text{FDNF}(E) = \top$.

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

**Definition 6** (\(\Psi(\varphi)\)) The automaton constraint corresponding to the constraint $\varphi$ is

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

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

### 3.3 Projection $f_{[k]}^{-1}(x, \varphi)$

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

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

$$L(A^{\Psi(\varphi)}(f(q_1, \ldots, q_n))) = \emptyset$$  \hspace{1cm} (2)
where we set $L(A^\psi(f(q_1, \ldots, q_n))) = f(L(A^\psi(q_1)), \ldots, L(A^\psi(q_n)))$. Otherwise (i.e., if the values of $q_1, \ldots, q_n$ are all nonempty, and condition (2) does not hold), one can infer $u \subseteq q_k$ (we set $E_u = q_k$).

In general, $E_x$ is of the form $E_x = \bigcup_i f_i(q_{i1}, \ldots, q_{in_i})$. Now, assume $f(q_1, \ldots, q_n)$ is a member of this union. If condition (2) is satisfied, then this member can be discarded from the union. Otherwise, we add $q_k$ to the union which forms $E_u$.

**Definition 7 (pre-$f^{-1}_{[k]}(E, \psi)$)** The $k$-th **pre-projection** of $f$ applied to an expression $E = \bigcup_i f_i(q_{i1}, \ldots, q_{in_i})$ with respect to the automaton constraint $\psi$, is the union of intersections

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

We set $\text{FDNF}(_) = \bot$.

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 $\varphi$. 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 $\bigwedge u \subseteq \bigcup_j u_{ij}$, which we then can add to $\varphi$, remaining within the restricted syntax of constraints.

**Definition 8 (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 $x \in \text{Var}$,

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

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

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

**Definition 9 ($f^{-1}_{[k]}(x, \varphi)$)** The $k$-th projection of $f \in \Sigma$ applied to the variable $x \in \text{Var}$ wrt. to the constraint $\varphi$ is an intersection of unions of variables $u_{ij} \in \text{Var}$,

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

Given a constraint $\varphi$, 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 $\varphi$. 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 1 (FDNF)** For any expression $E$ of the form $\bigcap_{i \in I} \bigcup_{j \in J_i} f_{ij}(\bar{u}_{ij})$, the equality $\alpha(E) = \alpha(\text{FDNF}(E))$ holds for every valuation $\alpha$. 

8
Proof. To see that \(\alpha(E) \subseteq \alpha(\text{FDNF}(E))\), 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 \(\alpha(f(u_1, \ldots, u_n) \cap f(v_1, \ldots, v_n)) = \alpha(f(u_1 \cap v_1, \ldots, u_n \cap v_n))\), 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 \(\text{FDNF}(E)\).

To see that \(\alpha(\text{FDNF}(E)) \subseteq \alpha(E)\), take the partial ordering on tuples of intersections defined by \((\cap S_1, \ldots, \cap S_n) < (\cap S'_1, \ldots, \cap S'_n)\) (which we abbreviate by \(\cap S < \cap S'\)) if \(S_i \subseteq S'_i\) holds for all \(i = 1, \ldots, n\). We observe that, if \(\cap S < \cap S'\), then \(\alpha(f(\cap S) \cup f(\cap S')) \subseteq \alpha(f(\cap S))\). Discard from \(\text{FDNF}(E)\) all disjuncts that are not minimal in this ordering, and call the result \(F\). By the observation above, \(\alpha(\text{FDNF}(E)) = \alpha(F)\). We have to show that \(\alpha(F) \subseteq \alpha(E)\). Take any disjunct \(f(\cap S)\) from \(F\). We will show that the value of this disjunct under \(\alpha\) 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 \(\bar{u}_{ij_i} \in S\). 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 \(\cap S\), 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 \(\alpha(f(\cap S)) = \alpha(\bigcap_{i \in I} f(\bar{u}_{ij_i}))\). \qed

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

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

Proposition 1 (Soundness) The axioms in Table 1 are valid. In particular, if a constraint \(\varphi\) is satisfiable then its closed form \(\varphi^C\) does not contain \textit{false}.

Proof. The proof is done by inspection of each axiom. The validity of Axioms 3 and 4 follows from consecutive applications of Lemma 1, Remark 2, and Lemma 2. \qed

Proposition 2 (Completeness) If the closed form \(\varphi^C\) of a constraint \(\varphi\) does not contain \textit{false} then \(\varphi\) is satisfiable. Moreover, the greatest solution of \(\varphi\) is the greatest solution of the automaton constraint \(\Psi(\varphi^C)\).

Proof. Let \(\alpha\) be the valuation defined by \(\alpha(x) = \mathcal{L}(\mathcal{A}(x))\), where \(\mathcal{A}(x)\) is the automaton corresponding to \(\Psi(\varphi)\) and the variable \(x\). By Remark 2, the unique extension of \(\alpha\) to \(\cap\text{-Var}(\varphi)\) is the greatest solution of \(\Psi(\varphi^C)\). Below we show that \(\alpha\) satisfies each conjunct in \(\varphi\). Since \(\varphi\) implies \(\Psi(\varphi^C)\), this will show that \(\alpha\) is the greatest solution of \(\varphi\).

The conjuncts of the form \(x \subseteq \bigcup_i f_i(\bar{u}_i)\) are trivially satisfied, since \(\text{FDNF}(\text{lub}(x, \varphi^C)) \subseteq \text{FDNF}(x)\).

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 \(t \not\in \alpha(\bigcup_i f_i(\bar{u}_i) \cup \bigcup_j y_j)\); we will show \(t \not\in \alpha(x)\). Since \(t \not\in \alpha(y_j)\) 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 \(\varphi^C\) (which includes the case of empty union \(y_j \subseteq \perp\)). Since \(t \not\in \alpha(\text{FDNF}(\text{lub}(y_j, \varphi^C)))\) for all \(j\), there is a constraint of the above form in \(\varphi^C\) such that \(t \not\in \alpha(\bigcup_k f_{jk}(\bar{u}_{jk}))\). By Axiom 2, \(\varphi^C\) contains a constraint
bounded by

Complexity

Lemma 3 For any intersection \( q \), \( \text{FDNF}(\text{lb}(q, \varphi)) \) can be computed in time exponential in the size of \( \varphi \).

Lemma 4 For any expression \( E = \bigcup_{i \in I} \bigcap_{j \in I} u_{ij} \), the expression \( \text{FCNF}(E) \) 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. \qed

Proposition 4 (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].\(^4\) 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. \(\Box\)

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 Path-closed set constraints

In this section we will consider the class of set constraints that was originally introduced by Mishra [18] and which we call path-closed set constraints. The class is syntactically larger; terms \(f(x_1, \ldots, x_n)\) may occur also on the left-hand side of an inclusion (union on the left-hand side is trivial). The interpretation is now over non-empty path-closed sets of trees. More precisely, a valuation \(\alpha\) satisfies the inclusion \(E \subseteq E'\) between two expressions \(E\) and \(E'\) if and only if \(\alpha(E)\) is a non-empty set and \(PC(\alpha(E)) \subseteq PC(\alpha(E'))\). Here, \(PC(X)\) denotes the path-closure of the set \(X\) of trees, i.e., the smallest path-closed set of trees containing \(X\). A regular set \(X\) is path-closed if it is recognized by a deterministic top-down tree automaton [10]. If the constraint in this class is satisfiable, the greatest solution always exists. (This would not be true if we added the empty set to the interpretation domain; take \(f(x, y) \subseteq \bot\).

We now define the algorithm for solving path-closed set constraints. In a first step, the constraints of the form \(f(x_1, \ldots, x_n) \subseteq \tau\) with \(n > 0\) are replaced by \(x_1 \subseteq f^{-1}_{[1]}(\tau) \wedge \ldots \wedge x_n \subseteq f^{-1}_{[n]}(\tau)\). In a second step, the satisfiability of the obtained constraint is tested. This step uses our previous algorithm modified as follows. We apply the rule

\[ \bigcup_i f(a_i) = f\left(\bigcup_i u_{i,1}, \ldots, \bigcup_i u_{i,n}\right) \]

for function symbols of arity \(n\). Thus we obtain always a deterministic top-down tree automata. If the value of a variable is determined to be the empty set, then the algorithm results “unsatisfiable”.

The correctness of the algorithm follows from the fact that for any sequence \(S_{11}, \ldots, S_{mn}\) of non-empty sets it holds that \(PC(\bigcup_{i=1}^m f(S_{i1}, \ldots, S_{im})) = PC(f(\bigcup_{i=1}^m S_{i1}, \ldots, \bigcup_{i=1}^m S_{im}))\), and from the following lemma:

Lemma 5 In the interpretation over path-closed sets defined above, the formula \(f(x_1, \ldots, x_n) \subseteq \tau\) and the formula \(x_1 \subseteq f^{-1}_{[1]}(\tau) \wedge \ldots \wedge x_n \subseteq f^{-1}_{[n]}(\tau)\) are equivalent.\(^4\)

\(^4\)The lower bound requires that the signature contains at least two function symbols, one of them having arity \(\geq 2\).
Proof. For the one direction, we assume $\alpha(f(x_1, \ldots, x_n)) \subseteq \alpha(\tau)$. We prove that for any $i = 1, \ldots, n$, $\alpha(x_i) \subseteq \alpha(f^{-1}_{ij}(\tau))$. Take any tree $t_i \in \alpha(x_i)$. We need to find trees $t_1, \ldots, t_i-1, t_i+1, \ldots, t_n$ such that $f(t_1, \ldots, t_n) \in \alpha(\tau)$. This is trivial from the non-emptiness of $\alpha(x_1), \ldots, \alpha(x_n)$.

For the other direction, we assume $\alpha(x_i) \subseteq \alpha(f^{-1}_{ij}(\tau))$ for all $i$. We prove $\alpha(f(x_1, \ldots, x_n)) \subseteq PC(\alpha(\tau))$. Take any $f(t_1, \ldots, t_n) \in \alpha(f(x_1, \ldots, x_n))$. We know that $t_i \in \alpha(x_i)$, so $t_i \in \alpha(f^{-1}_{ij}(\tau))$. By the definition of projection, there exist trees $t_1^1, \ldots, t_i^{i-1}, t_i^{i+1}, \ldots, t_n^i$ for all $i$ such that $f(t_1^1, \ldots, t_i^i, \ldots, t_n^i) \in \alpha(\tau)$. Then, $f(t_1, \ldots, t_n) \in PC(\alpha(\tau))$ holds.$^5$

**Theorem 3** The satisfiability problem of path-closed set constraints is DEXPTIME-complete.

*Proof.* We have shown the upper bound. The lower bound follows from Seidl’s characterization of the problem of the emptiness of the intersection for deterministic top-down tree automata [22].

7 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 $\nu$-level in the hierarchy of the fixpoint calculus of Niwiński [19]. The essential difference between the fixpoint expressions on the $\nu$-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).

---

$^5$The complexity of the satisfiability test does not change if we add the empty set to the interpretation domain. Applying the equivalence:

$$f(x_1, \ldots, x_n) \subseteq \tau \Leftrightarrow (x_1 \subseteq \bot) \lor \ldots \lor (x_n \subseteq \bot) \lor (x_1 \not\subset \bot \land \ldots \land x_n \not\subset \bot \land f(x_1, \ldots, x_n) \subseteq \tau)$$

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

A Automaton constraints and automata

A (finite non-deterministic top-down tree) automaton is a tuple \( \mathcal{A} = (\Sigma', Q, \delta, q_{\text{init}}, Q_\top) \) consisting of its finite alphabet \( \Sigma' \subseteq \Sigma \), finite set of states \( Q \), (non-deterministic) transition function \( \delta : Q \times \Sigma' \to \mathcal{P}(Q) \) (where \( Q \) stands for the set of all tuples over \( Q \)), initial state \( q_{\text{init}} \) and the set \( Q_\top \) of all-accept states. The tree automaton \( \mathcal{A} \) accepts a tree \( t \) (or: \( t \) lies in the language \( L(\mathcal{A}) \) recognized by \( \mathcal{A} \)) if 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. The emptiness of such an 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 \( \delta_q(q, f) \). If the label of the node of \( t \) is a constant symbol, then the set \( \delta_q(q, f) \) 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 \( \Sigma' \) or not).

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

**Definition 10** (\( \mathcal{A}^\psi(q) \)) The automaton corresponding to the automaton constraint \( \psi \) and the variable \( q_0 \in q\text{-Var}(\psi) \) is the tuple \( \mathcal{A}^\psi(q_0) = (\Sigma(\psi), \text{Var}(\psi), \delta_\psi, q_0) \) where

- the alphabet is the set \( \Sigma(\psi) \) of function symbols occurring in \( \psi \);
- the states are the variables \( q \) occurring in \( \psi \);
- the set \( \delta_\psi(q, f) \), i.e., the transition function \( \delta_\psi \) 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 \( \psi \) (which is then unique),
  - the empty set \( \emptyset \) if \( q \subseteq \bot \) is a conjunct in \( \psi \);
- the initial state is \( q_0 \);
- the all-accept states are the unbounded variables in \( \psi \).

If the variable \( q_0 \in q\text{-Var} \) does not occur at all in \( \psi \), then \( \mathcal{A}^\psi(q_0) = (\emptyset, \{q_0\}, \emptyset, q_0, \{q_0\}) \) (an automaton accepting \( T_\Sigma \)).

It is clear that \( L(\mathcal{A}^\psi(q)) \) is the empty set if \( q \subseteq \bot \) is in \( \psi \) and the set \( T_\Sigma \) of all trees if \( q \) is unbounded in \( \psi \). More generally, the statement below holds.

**Observation 1** The valuation \( \alpha : q \mapsto L(\mathcal{A}^\psi(q)) \) is the greatest solution of the automaton constraint \( \psi \).
Proof. We will first show that any solution $\beta$ of $\psi$ is smaller than the valuation $\alpha$. We extend $\beta$ to a mapping over all states of the automata by setting $\beta(\top) = T_{\top}$. 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 $A^\psi(p)$ on $t$ that satisfies the following invariant: If $A^\psi(q)$ 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 $A^\psi(q)$ 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 $\psi$, then the automaton goes to the state $\top$ in all positions $p, 1, \ldots, p, n$. Since $A^\psi(\top)$ recognizes the set $T_{\top}$ of all trees, our invariant is satisfied. Now suppose $q'$ is not unbounded. Since $\beta$ is a solution, on the right-hand side of the inclusion constraining $q$ in $\psi$ 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, $t|_p \in \beta(q_i)$ for $i = 1, \ldots, n$. But then, by the definition of $A^\psi(q)$, $(q_1, \ldots, q_n) \in \delta_\psi(q', f)$. 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 $\alpha$ satisfies every inclusion $q \subseteq E$ in $\psi$. Again, if $L(A^\psi(q))$ is empty, nothing is to show; otherwise, we take an element $t = f(t_1, \ldots, t_n)$ from $L(A^\psi(q))$. By the definition of $L(A^\psi(q))$, there exists a run of $A^\psi(q)$ 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, $t_i \in L(A^\psi(q_i))$. That is, $t \in f(L(A^\psi(q_1)), \ldots, L(A^\psi(q_n))) = f(\alpha(q_1), \ldots, \alpha(q_n))$. By the definition of $A^\psi(q)$, the expression $E$ is of the form $E = f(q_1, \ldots, q_n) \cup E'$. Therefore, $t \in \alpha(E)$. \square

References


