Atomic Set Constraints with Projection

Witold Charatonik[1,2] and Jean-Marc Talbot[3]

[1] Max-Planck Institut für Informatik, Saarbrücken, Germany
[2] University of Wrocław, Poland
[3] Laboratoire d'Informatique Fondamentale de Lille, Lille, France

Abstract. We investigate a class of set constraints defined as atomic set constraints augmented with projection. This class subsumes some already studied classes such as atomic set constraints with left-hand side projection and INES constraints. All these classes enjoy the nice property that satisfiability can be tested in cubic time. This is in contrast to several other classes of set constraints, such as definite set constraints and positive set constraints, for which satisfiability ranges from DEXPTIME-complete to NEXPTIME-complete. However, these latter classes allow set operators such as intersection or union which is not the case for the class studied here. In the case of atomic set constraints with projection one might expect that satisfiability remains polynomial. Unfortunately, we show that that the satisfiability problem for this class is no longer polynomial, but CoNP-hard. Furthermore, we devise a PSPACE algorithm to solve this satisfiability problem.

1 Introduction

Set constraints are first-order formulas interpreted over sets of trees. They are defined as inclusions between set expressions. These set expressions are built up from variables, a ranked signature (constants and function symbols) and various set operators. According to the set operators used, different classes of set constraints are defined.

The main domain of application of set constraints is program analysis, often called set-based analysis [2, 13, 15]. Set constraints play also a role in type checking [24] (See [1, 16, 20] for surveys).

The satisfiability problem for different classes of set constraints has been extensively studied and its complexity turns out to be often quite high; in most cases (definite set constraints [15], set constraints with intersection [8], positive and/or negative set constraints [4, 6, 12], positive and negative set constraints with projection [7]) satisfiability is DEXPTIME-hard [8] or even NEXPTIME-hard [5] for some of them.

Nonetheless for some other classes, it turned out that the satisfiability problem is tractable, more precisely cubic time. Atomic constraints are such a class: for these simple constraints, no set operator is allowed. A cubic time algorithm for satisfiability for these constraints is given in [13]. Motivated by program analysis, Müller, Niehren and Podelski introduced in [19] the class of INES constraints (Inclusions over Non-Empty Sets of trees). They correspond syntactically to atomic set constraints but the empty set is no longer an admissible value to instantiate variables. They showed that satisfiability for INES constraints can be tested in cubic time. In the same paper, the authors also investigate atomic set constraints extended with the possibility to force the denotation of some variables (but maybe not all of them) to be non-empty and show that satisfiability remains cubic.

In [13], another extension of atomic set constraints have been studied. There the syntax of constraints was extended to allow the use of projection on the left-hand side of inclusions. It has been shown that satisfiability remains cubic also in this case.

A natural question is then "Can theses two extensions (atomic set constraints with non-emptiness constraints and projection in the left-hand side of inclusions) be combined to obtain a new class for which satisfiability could be tested in cubic time ?". We depict in the table below known results from previous works about complexity of the satisfiability problem for extensions of the class of atomic set constraints. At the first view the answer seems to be "yes"; in the conclusion of an extended version of [19] available from Podelski's web page the authors explicitly say that it should be straightforward. Surprisingly, in Section 3 we show that even in the "simple" case of projections on the left-hand side of inclusions, in presence of non-emptiness constraints, the satisfiability problem becomes CoNP-hard.

	no left-hand side projection	with left-hand side projection
no emptiness constraints	$O(n^3)$	$O(n^3)$ [13]
with emptiness constraints	$O(n^3)$ [19]	**?**

On one hand, projection is a quite intricate operator which can be usually treated easily in the left-hand set of inclusions ([8, 21]) by a simple elimination method.[1] However, it leads to more difficult processing in the right-hand side: several methods have been proposed for classes of set constraints that do not admit projection at all [3, 4, 6, 12, 19, 23] or admit it only on the left-hand side of inclusions [5, 8, 13, 15, 21] but only few concerning projections in the right-hand side [7, 9, 11]. On the other hand, for the latter classes adding projection operation does not increase the complexity of satisfiability problem. So, one may wonder whether projections could be freely added to atomic set constraints without modifying the cubic time complexity for satisfiability. Again our result from Section 3 implies the negative answer: there exists a simple encoding of non-emptiness constraints using projections in the right-hand side of inclusions provided that the signature is rich enough.

Our second contribution is an algorithm for solving atomic set constraints with projections. It covers both extensions with non-emptiness constraints and projection on both sides of inclusions. We show that it works in PSPACE, while the only previously known algorithm covering this class of constraints (devised for the general set constraints with projections [7]) worked in NEXPTIME. There is still a gap between the CoNP lower bound and PSPACE upper bound; we leave open the question about the exact complexity of the problem.

This work is a continuation of the research project initialized by Aiken, Kozen, Vardi, and Wimmers in [3] to characterize the complexity of the satisfiability problem for set constraints depending on the syntax used. Given the NEXPTIME lower bound in the general case it is natural to ask under which assumptions faster algorithms may be found. In [3] the authors give a detailed analysis of the complexity of subclasses of

[1] Known methods for elimination of left-hand side projections either require boolean operations on set expressions or are correct only wrt. least solutions. Therefore they do not apply to the class in question (cf. Examples 1 and 2).

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). This classification, however, does not say much about applications in program analysis where we usually have to deal with symbols of high arity. The approach based on the choice of used set operators seems to be more appropriate here. For example, the complement operation is almost never used in set-based analysis. Contrary to this, projection (on the left-hand side of inclusion) is used in most applications, it is present already in the early paper by Reynolds [21]. The restriction to nonempty sets is often used as an optimization to speed up the solving process.

Due to lack of space some proofs are omitted; they can be found in [10].

2 Atomic Set Constraints with Projection

2.1 Definitions

We consider a fixed signature Σ, that is a set of function symbols equipped with arities. Constants are symbols from Σ with arity 0. We use a, b, c to denote constants and f, g, h to denote symbols of arity at least one. We assume moreover that Σ contains at least one constant and one symbol of arity greater than or equal to 2.[2] We denote by $\Sigma^{>0}$ the set of all symbols from Σ that are not constants. We also use a countable set $V = \{x, y, z, \dots\}$ of variables ranging over sets of trees over the signature Σ.

Trees: Let Π be the free monoid generated by the empty path ϵ and concatenation over $\mathbb{N}^+$, the set of strictly positive natural numbers. Words from Π are called *paths*. Given two paths π, π' we write $\pi.\pi'$ for the concatenation of π and π'.

Let τ be a partial mapping from Π to Σ. The mapping τ is said to be (i) *non-empty* if τ is defined for some π, (ii) *prefix-closed* if for any π and i such that τ is defined for $\pi.i$, then τ is defined for π as well and, (iii) *arity consistent* if for all π, if τ is defined for π and the arity of $\tau(\pi)$ is n, then τ is defined for $\pi.1, \dots, \pi.n$ and undefined for all $\pi.j$ with $n < j$.

A tree τ is a partial mapping from Π to Σ, which is non-empty, prefix closed and arity consistent. A tree τ is *finite* if τ is defined for only finitely many paths. We denote $\mathcal{T}^{\omega}_{\Sigma}$ (resp. $\mathcal{T}_{\Sigma}$) the set of all trees (resp. of all finite trees) generated over Σ. For a tree τ and a path π, we write $\tau[\pi]$ for the subtree of τ rooted at position π. If τ is not defined for π then also $\tau[\pi]$ is undefined.

Set Constraints: let Σ^{-1} be the set of unary function symbols, called *projections* and defined as $\{f_i^{-1} \mid f \in \Sigma^{>0}$ and i ranges over $1 \dots arity(f)\}$. We call *set expressions*, terms that are built from V,Σ and Σ^{-1}. We use the letters s, t, u, v (possibly with subscripts) to denote set expressions.

[2] This assumption will be used only for the lower-bound proof; the algorithm from Section 4 works also in the unary case. On the other hand, e.g. in the unary case without emptiness constraints all projections can be easily eliminated and the problem becomes polynomial.

Definition 1. *An* atomic set constraint with projection φ *is given by the following grammar (where s, t are some set expressions and x is a variable)*

$$\varphi ::= s \subseteq t \mid x \nsubseteq \varnothing \mid \varphi \wedge \varphi$$

In the definition above, $\wedge$ stands for logical conjunction. However, we will often identify such conjunction with the set of all its conjuncts.

The semantics of our constraints is given by the structure of sets of trees $\mathsf{P}(\mathcal{T}_\Sigma^\omega)$ or by the structure of sets of finite trees $\mathsf{P}(\mathcal{T}_\Sigma)$: the domain of $\mathsf{P}(\mathcal{T}_\Sigma^\omega)$ (resp. of $\mathsf{P}(\mathcal{T}_\Sigma)$) is the set of all subsets of $\mathcal{T}_\Sigma^\omega$ (resp. of $\mathcal{T}_\Sigma$) and function and projection symbols are interpreted as follows: for sets $S, S_1, \ldots, S_n$ of trees (or of finite trees),

$$f(S_1, \ldots, S_n) = \{f(\tau_1, \ldots, \tau_n) \mid \tau_1 \in S_1, \ldots, \tau_n \in S_n\}$$
$$f_i^{-1}(S) = \{\tau_i \mid f(\tau_1, \ldots, \tau_n) \in S\}$$

Finally, both $\mathsf{P}(\mathcal{T}_\Sigma^\omega)$ and $\mathsf{P}(\mathcal{T}_\Sigma)$ interpret $\subseteq$ as set inclusion and $\nsubseteq \varnothing$ as non-emptiness.

A valuation is a mapping that associates a set of trees with each variable from V. It can be then canonically extended to all set expressions.

We say that a valuation σ satisfies an inclusion $s \subseteq t$ iff $\sigma(s)$ is a subset of $\sigma(t)$ and satisfies a non-emptiness constraint $x \nsubseteq \varnothing$ iff $\sigma(x)$ is a non-empty set. We say that σ satisfies a constraint φ if it satisfies each conjunct from φ.

The satisfiability problem is to determine for a given constraint φ whether there exists a valuation σ that satisfies φ. In this case, we say that σ is a solution of φ. When φ admits a solution, we say that φ is satisfiable, otherwise it is said to be unsatisfiable.

2.2 Expressiveness

Non-emptiness constraints $x \nsubseteq \varnothing$ in Definition 1 may be seen as redundant as we can encode them using projection provided that the signature Σ contains a binary function symbol and a constant as follows:

Example 1. The constraint $a \subseteq f_1^{-1}(f(a, x))$ implies that the denotation of the variable x must be non-empty in any solution of this constraint.

Atomic set constraints with projection generalize several classes of set constraints that have already been studied and for which a cubic time satisfiability algorithm has been devised.

□ The atomic set constraints and atomic set constraints with left-hand side projections correspond to restrictions of the syntax from Definition 1: in both of them, non-emptiness constraints $x \nsubseteq \varnothing$ are not allowed. Moreover, in the first one, set expressions s and t from Definition 1 do not contain projection whereas for the second one only s may contain projections.

Satisfiable constraints from these two classes always have a least solution. This property does not hold when projections are added, which proves that atomic set constraints with projections are strictly more expressive.

Example 2. Looking again at the constraint from Example 1, it is easy to find two minimal solutions σ,σ' such that respectively $\sigma(x) = \{a\}$ and $\sigma'(x) = \{f(a, a)\}$ that are not comparable. Hence, there is no least solution for this constraint.

□ INES Constraints have been introduced in [19]: syntactically defined as atomic set constraints, their main feature is that they are interpreted over non-empty set of trees. As for atomic set constraints, a cubic time algorithm to solve satisfiability has been proposed in [19].

We can encode an INES constraint φ in the constraint from Definition 1 by simply adding to φ the non-emptiness constraint $x \not\subseteq \varnothing$ for all variables x occurring in φ.

INES constraints always admit a greatest solution. However, this is not true for atomic constraints with projection as exemplified below. This proves that our class is strictly more expressive than the class of INES constraints.

Example 3. The constraint $f(x, x') \subseteq a$ implies that the denotation of either x or x' is empty. Thus, one can find two incomparable maximal solutions σ,σ' satisfying respectively $\sigma(x) = \varnothing$, $\sigma(x') = \mathcal{T}_\Sigma$ and $\sigma'(x) = \mathcal{T}_\Sigma$, $\sigma'(x') = \varnothing$. It should be noticed that this constraint is simply not satisfiable over INES.

□ Beyond INES, in [19] the authors consider also atomic set constraints enhanced with non-emptiness constraints $x \not\subseteq \varnothing$ and give a cubic time satisfiability algorithm.

It is proved in [19] that every satisfiable constraint φ in this class has a solution where each nonempty variable x contains either only trees starting with one particular function symbol f (if φ contains an inclusion $x \subseteq f(\ldots)$) or it contains trees starting with all function symbols from Σ (if φ does not contain such an inclusion). This is no longer the case for atomic set constraints with projection. For example the constraint $f_i^{-1}(x) \subseteq y \ \wedge y \subseteq a \ \wedge y \subseteq b$ implies that the variable x may contain trees starting with any function symbol from Σ except f. This proves that atomic set constraints with projection are strictly more expressive than atomic set constraints with non-emptiness constraints.

Our final example in this section shows some of the ideas behind our lower-bound proof (cf. the use of markers in the next section). Note that the projection here occurs only on the left-hand side of inclusions.

Example 4. Consider the constraint

$$\begin{array}{lllll} x \not\subseteq \varnothing & \wedge & x \subseteq f(a, z) & \wedge & z \subseteq a \\ & \wedge & f_2^{-1}(x) \subseteq v & \wedge & v \subseteq f(a, a). \end{array}$$

It is not satisfiable because the first two columns imply that intersection of z and v must be nonempty while the third one implies that this intersection is empty.

3 Complexity Lower Bound

We show in this section that the satisfiability problem for atomic set constraints with projection is CoNP-hard. We prove this statement by presenting a polynomial reduction from the unsatisfiability problem for propositional formulas in conjunctive normal form to the satisfiability problem of these set constraints. We recall that CoNP is the class of problems whose complements are in NP and that a problem is CoNP-hard (respectively CoNP-complete) if its complement is NP-hard (respectively NP-complete).

We consider propositional variables ranged over by u. A literal l is either a propositional variable u or its negation $\neg u$. A clause is a disjunctive formula $l_1 \vee \ldots \vee l_n$ where l_i is a literal. We say that a clause C is non-trivial if the disjunction contains at least one literal and each variable u appears at most once in C either positively u or negatively $\neg u$. A CNF formula is a finite conjunction of non-trivial clauses.

The Problem CNF-UNSAT is as follows :

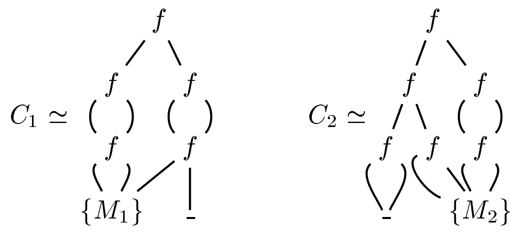
- INPUT :
  - $u_1, \ldots, u_n$: a non-empty sequence of propositional variables.
  - $C_1 \wedge \ldots \wedge C_p$: a CNF formula written over $u_1, \ldots, u_n$.
- ANSWER : Yes, if $C_1 \wedge \ldots \wedge C_p$ is not satisfiable. No, otherwise.

It is well-known that satisfiability of CNF formulas is NP-complete, hence CNF-UNSAT is CoNP-complete.

Before presenting formally how CNF-UNSAT can be polynomially reduced to the satisfiability problem of atomic set constraints with projection, let us sketch the main ideas of this reduction. Some of these ideas have been used before in [17, 18] to prove hardness for satisfiability or entailment problems for constraints overs trees or sets of trees. We describe first these ideas in terms of trees and then turn when needed to sets of trees.

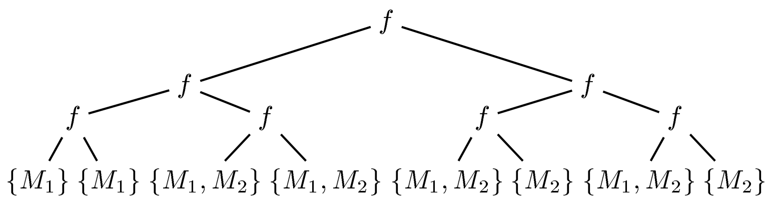
The first point is that Boolean valuations for a sequence of propositional variables $u_1, \ldots, u_n$ can be viewed as a path of length n in a complete binary tree assuming that 1 stands for $true$ and 2 for $false$. For instance, for the sequence u_1, u_2, u_3, the path 1.2.1 encodes the valuation associating $true$ with u_1 and u_3 and $false$ with u_2.

The second idea is to notice that valuations satisfying a clause C do not need to be depicted as a full binary tree, but that a dag representation is sufficient. For instance, the clauses $C_1 = u_1 \vee u_3$ and $C_2 = \neg u_1 \vee \neg u_2$ can be represented as follows



We label leaves of the dags with some markers distinguishing the valuations (paths) satisfying the clause from the valuations that do not. For instance, in the representation of C_1 and of C_2, we used $\{M_1\}$ and $\{M_2\}$ respectively to point out valuations satisfying this clause and _ for the others.

Now, if we unfold these two dags to full binary trees and overlap one with the other we obtain the tree

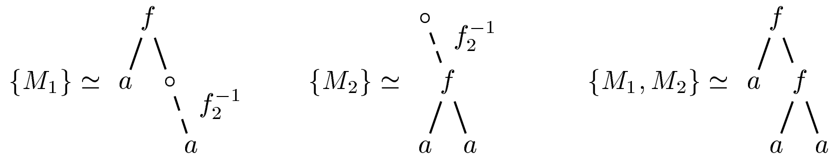


where each leaf corresponds to a valuation (described by the path leading to this leaf) and is labeled with the set of markers identifying the clauses satisfied by this valuation. For example, the node identified by the path 1.2.1 is labeled with $\{M_1, M_2\}$ which indicates that both clauses C_1 and C_2 are satisfied by this valuation. In the general case we will have that the formula in question is satisfiable if and only if this corresponding tree contains a node labeled with all markers.

Now, let us come to a "set" point of view. Encoding of markers and their use to produce an unsatisfiable set constraint in case of a satisfiable formula is the novelty of this reduction. Let us give some intuitions about that. We will use $p + 1$ markers, one for each clause and one "tester". Each marker represents a set of trees. These sets are constructed in such a way that the intersection of any p of them is nonempty while the intersection of all $p + 1$ is empty. A node in the overlapped tree above labeled with several markers represents the intersection of the sets represented by all these markers. Now if the formula $C_1 \wedge \ldots \wedge C_p$ is satisfiable then the overlapped tree contains a node marked with $\{M_1, \ldots, M_p\}$, hence the set represented by this node intersected with the set represented by the tester is empty, which together with a non-emptiness constraint make the corresponding set constraint unsatisfiable.

The crucial part is the definition of the markers. It heavily depends on the use of projections and is a generalization of Example 4, where the two lines define respectively the marker and the tester in the case of $p = 1$. The rough idea is that a constraint of the form $f_2^{-1}(x) \subseteq x'$ is very similar to a constraint of the form $x \subseteq f(\_, x')$. The important difference is that in the second case the variable x may contain only trees whose root is labelled with the symbol f while in the first case it may contain terms starting with a different symbol. On the picture below we use dashed lines for the constraints of the first form and solid lines for the constraints of the second form.

For $p = 2$, the markers $\{M_1\}$, $\{M_2\}$ can be depicted as the following two trees on the left:

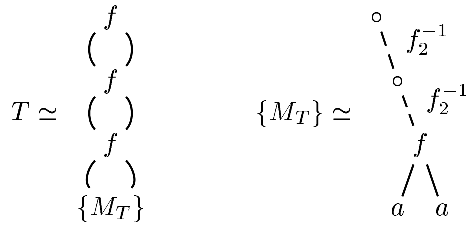


Sets of trees represented by the markers are partially described by means of projections. The main part in the markers is the right-most path, that is paths of the form 2^*. For the makers $\{M_i\}$ we force the node 2^{i-1} to be an f: for $\{M_1\}$, this is ϵ and for $\{M_2\}$ the node at path 2. The rest of this path is defined as projection f_2^{-1}.

Roughly speaking, the meaning of $\{M_1\}$ is a non-empty set of trees satisfying the following requirement: the root of the tree is labeled with f and the position 1 with a. Additionally, if the position 2 is labeled with f then the position 2.2 is labeled with a. Now, concerning $\{M_2\}$, its meaning is a non-empty set of trees satisfying the following requirement: if the root is labeled with f then the positions 2, 2.1 and 2.2 are labeled respectively with f, a and a.

Now, the marker $\{M_1, M_2\}$ produced by overlapping trees from the clauses is described above as the last tree on the right. Note that this tree is totally defined, there is no more projection f_2^{-1} over it. This property implies that the intersection with the set represented by the tester is empty. The meaning of $\{M_1, M_2\}$ must satisfy both requirements of $\{M1\}$ and $\{M_2\}$; the only set satisfying them is the singleton $\{f(a(f(a,a)))\}$.

The last point of the encoding is the tester. The left-hand part of the picture below is used for overlapping with the whole binary tree, the right-hand part is the actual tester.



The tester $\{M_T\}$ represents any non-empty set of trees that satisfy the requirement: if the positions ϵ and 2 of the tree are labeled with f then positions 2.2, 2.2.1 and 2.2.2 are labeled respectively by f, a and a.

Note that the intersection of the sets represented by $\{M_T\}$ with any of $\{M_1\}, \{M_2\}$ is nonempty but the intersection of $\{M_T\}$ with $\{M_1, M_2\}$ is empty. Indeed, the set $\{f(a,a)\}$ satisfies the requirements of both $\{M_1\}$ and $\{M_T\}$ whereas the set $\{a\}$ satisfies the requirements of both $\{M_2\}$ and $\{M_T\}$. But no non-empty set can satisfy the requirements of $\{M_1\}$, $\{M_2\}$ (that is, of $\{M_1, M_2\}$) and $\{M_T\}$ at the same time.

We show now formally that we can built in polynomial time from any instance of the CNF-UNSAT problem a set constraint with projection that is satisfiable iff the answer to the instance of CNF-UNSAT is Yes.

We consider a signature Σ containing a symbol f which is at least binary and another function symbol. For simplicity, we will assume f to be binary and that the other function symbol is a constant a.

We consider an instance of CNF-UNSAT given by $\{u_1, \ldots, u_n, C_1 \wedge \ldots \wedge C_p\}$ (for short, $\{\bar{u}, \bar{C}\}$) and define its encoding $Enc(\bar{u}, \bar{C})$. We use variables x, x^i_j, y^i_j, w_j with $1 \leq i \leq p$, $1 \leq j \leq n+1$ and z^i_k, v^i with $1 \leq i, k \leq p$. The variables x^i_j, y^i_j are used to encode Boolean valuations (for the ith clause and the jth variable), the variables w_j to encode the top part of the tester and the variables z^i_k, v^i to encode respectively the markers for clauses and the marker of the tester. We let

$$Enc(\bar{u}, \bar{C}) = \begin{cases} x \not\subseteq \varnothing \wedge x \subseteq w_1 \wedge Enc_{Test}(\bar{u}) \wedge Enc_{MTest}(\bar{C}) \\ \displaystyle\bigwedge_{1 \leq i \leq p} x \subseteq x^i_1 \wedge \bigwedge_{1 \leq i \leq p} Enc_{Cl}(i, \bar{u}, \bar{C}) \wedge \bigwedge_{1 \leq i \leq p} Enc_{Mark}(i, \bar{C}) \end{cases}$$

$$Enc_{Test}(\bar{u}) = \bigwedge_{1 \leq j \leq n} w_j \subseteq f(w_{j+1}, w_{j+1}) \wedge w_{n+1} \subseteq v^1$$

$$Enc_{MTest}(\bar{C}) = \bigwedge_{1 \leq i \leq p} f_2^{-1}(v^i) \subseteq v^{i+1} \wedge v^{p+1} \subseteq f(a,a)$$

$$Enc_{Mark}(i, \bar{C}) = \bigwedge_{1 \leq k \leq p, k \neq i} f_2^{-1}(z^i_k) \subseteq z^i_{k+1} \wedge z^i_i \subseteq f(a, z^i_{i+1}) \wedge z^i_{p+1} \subseteq a$$

$$Enc_{Cl}(i,\bar{u},\bar{C}) = \bigwedge_{1\leq j\leq n} y^i_j \subseteq f(y^i_{j+1}, y^i_{j+1}) \wedge y^i_{n+1} \subseteq z^i_1 \wedge \bigwedge_{1\leq j\leq n} Enc_{LC}(i,j,\bar{u},\bar{C})$$

$$Enc_{LC}(i,j,\bar{u},\bar{C}) = \begin{cases} x^i_j \subseteq f(y^i_{j+1}, x^i_{j+1}) & \text{if } u_j \text{ occurs in } C_i \\ x^i_j \subseteq f(x^i_{j+1}, y^i_{j+1}) & \text{if } \neg u_j \text{ occurs in } C_i \\ x^i_j \subseteq f(x^i_{j+1}, x^i_{j+1}) & \text{if } u_j \text{ and } \neg u_j \text{ don't occur in } C_i \end{cases}$$

Proposition 1. *A CNF formula $C_1 \wedge \ldots \wedge C_p$ written over variables $u_1, \ldots, u_n$ is satisfiable if and only if the corresponding set constraint $Enc(\{u_1, \ldots, u_n, C_1, \ldots, C_p\})$ is not satisfiable.*

Theorem 1. *Let Σ be a signature containing at least two function symbols, one of these symbols having an arity greater than or equal to two. Then satisfiability problem for atomic set constraints with projection is Co-NP-hard both over sets of trees $\mathsf{P}(\mathcal{T}^\omega_\Sigma)$ and sets of finite trees $\mathsf{P}(\mathcal{T}_\Sigma)$.*

Proof. *Obvious from Proposition 1 noticing that the size of the constraint $Enc(\{\bar{u},\bar{C}\})$ is polynomial in the size of $\{\bar{u},\bar{C}\}$, the input of CNF-UNSAT.*

Remark 1. Consider an introduction of projections of constant symbols to the syntax of set constraints with the semantics $a^{-1}(S) = \mathcal{T}_\Sigma$ if $a \in S$ and $a^{-1}(S) = \varnothing$ otherwise. Then the satisfiability problem becomes DEXPTIME-hard by a simple reduction from the emptiness of intersection of deterministic top-down tree automata.

4 A Satisfiability Algorithm

4.1 Preliminaries

To handle constraints more easily we will assume them to be *flat*.

Definition 2. *A constraint φ is said to be flat if it is generated by the following grammar ($x, y, x_1, \ldots, x_n$ being variables)*

$$\varphi ::= x \subseteq y \mid x = f(x_1, \ldots, x_n) \mid x = f_i^{-1}(y) \mid x \nsubseteq \varnothing \mid \varphi \wedge \varphi \mid \bot$$

where equality $x = t$ is interpreted as $x \subseteq t \wedge t \subseteq x$ and $\bot$ represents the value False. As soon as $\bot$ occurs in a constraint φ, φ is not satisfiable.

Considering flat constraints is not a loss of generality: for any constraint φ one can compute in linear time in the size of φ a flat constraint φ' equivalent to φ.

Given a constraint φ, we write Σ_φ for the set of all function symbols f from Σ occurring in φ either as a function symbol f or as a projection f_i^{-1}. The set $\Sigma^{>0}_\varphi$ is the set Σ_φ restricted to symbols that are not constants.

4.2 Annotated paths - Path constraints

We introduce a new kind of constraints called *path constraints* defined over annotated paths.

Annotated Paths: An *annotated path* κ is an element of the free monoid generated by the empty path ϵ and concatenation over $\{\langle f, i\rangle \mid f \in \Sigma_\varphi^{>0}$ and $1 \leq i \leq arity(f)\}$.

Given a tree τ and an annotated path κ, we define the subtree $\tau[\kappa]$ of τ rooted at κ as (i) τ if $\kappa = \epsilon$, as (ii) $\tau_i[\kappa']$ if $\kappa = \langle f, i\rangle.\kappa'$ and $\tau = f(\tau_1, \ldots, \tau_n)$, and (iv) undefined otherwise.

It should be noticed that whereas paths express only positions in a tree, annotated paths express also conditions over the labeling of the tree; for the trees $\tau = f(a, b)$ and $\tau' = g(a, b)$, both $\tau[1]$ and $\tau'[1]$ are defined and equal to a. But for the annotated path $\langle f, 1\rangle$, the tree $\tau[\langle f, 1\rangle]$ is defined and equal to a, whereas $\tau'[\langle f, 1\rangle]$ is undefined as the root of τ' is a symbol different from f.

Given a set of trees S, we write $S[\kappa]$ for the set $\{\tau[\kappa] \mid \tau \in S\}$. It is always defined although it can be empty. For an annotated path κ, we define its *restriction over* $\mathbb{N}^+$ (denoted $\mathsf{res}(\kappa)$) as the path π obtained from κ by removing annotating function symbols.

Path Constraints: a path constraint ψ is written as $x[\kappa] \subseteq y$. Its semantics is as follows: a valuation σ satisfies $x[\kappa] \subseteq y$ iff $\sigma(x)[\kappa] \subseteq \sigma(y)$. Note that if $\kappa = \langle f, 1\rangle$ is an annotated path of length 1 then $x[\kappa] \subseteq y$ is just a different notation for $f_i^{-1}(x) \subseteq y$.

We define a binary relation $\vdash$ between flat constraints and path constraints inductively as follows:

- $\varphi \vdash x[\epsilon] \subseteq y$ if $x \subseteq y \in \varphi$.
- $\varphi \vdash x[\langle f, i\rangle] \subseteq x_i$ if $x = f(x_1, \ldots, x_n) \in \varphi$ or $f_i^{-1}(x) = x_i \in \varphi$.
- $\varphi \vdash x[\kappa.\kappa'] \subseteq y$ if there exists z such that $\varphi \vdash x[\kappa] \subseteq z$ and $\varphi \vdash z[\kappa'] \subseteq y$.

Proposition 2. *If $\varphi \vdash x[\kappa] \subseteq y$ then any solution σ of φ, $\sigma(x)[\kappa] \subseteq \sigma(y)$.*

Proof. By induction over the length of κ.

Let us consider an annotated path κ such that for some x and y, $\varphi \vdash x[\kappa] \subseteq y$. There may exist a solution σ for which $\sigma(x)$ is non-empty but nonetheless $\sigma(x)[\kappa]$ is empty. For instance, for the constraint $\varphi_1 = x \nsubseteq \varnothing \wedge f_1^{-1}(x) = y \wedge y = g(z)$ there exists a solution σ such that $\sigma(x) = \{a\}, \sigma(y) = \sigma(z) = \varnothing$, so $\sigma(x)[\langle f, 1\rangle]$ is empty. Now considering the constraint $\varphi_2 = \varphi_1 \wedge x = f(x_1, x_2)$, one can deduce that for any solution σ' of φ_2, both $\sigma'(x)[\langle f, 1\rangle]$ and $\sigma'(x)[\langle f, 1\rangle.\langle g, 1\rangle]$ is not empty. The annotated paths $\langle f, 1\rangle$ and $\langle f, 1\rangle.\langle g, 1\rangle$ become unavoidable for x in the constraint φ_2.

We define $\mathcal{UAP}(x, \varphi)$ the set of *unavoidable annotated paths* for the variable x and the constraint φ as the least set satisfying :

- $\epsilon \in \mathcal{UAP}(x, \varphi)$
- if $\kappa \in \mathcal{UAP}(x, \varphi)$, $\varphi \vdash x[\kappa] \subseteq y$ and $y = f(\bar{y})$ then $\kappa.\langle f, i\rangle \in \mathcal{UAP}(x, \varphi)$ for all $1 \leq i \leq arity(f)$.

Proposition 3. *For all solutions σ for φ, for all variables x, for all annotated paths κ from $\mathcal{UAP}(x, \varphi)$, if $\sigma(x)$ is non-empty then for all trees τ in $\sigma(x)$, $\tau[\kappa]$ is defined.*

Proposition 4. *For all variables x in φ, if there exists an unavoidable annotated path κ in $\mathcal{UAP}(x, \varphi)$ such that $\varphi \vdash x[\kappa] \subseteq y$, $\varphi \vdash x[\kappa] \subseteq z$ and for two different function symbols f, g, the two constraints $y = f(\bar{y})$, $z = g(\bar{z})$ belong to φ then for any solution σ for φ, $\sigma(x)$ is empty.*

(I1) $x \subseteq x$; $x \subseteq y, y \subseteq z \to x \subseteq z$
(I2) $x \nsubseteq \varnothing, x = f(\bar{x}), y = f(\bar{y}), x \subseteq y \to \bar{x} \subseteq \bar{y}$
(I3) $x \nsubseteq \varnothing, x = f(\bar{x}), y = f_i^{-1}(z), x \subseteq z \to x_i \subseteq y$
(I4) $x = f_i^{-1}(y), y \subseteq z, z = f(\bar{z}) \to x \subseteq z_i$
(I5) $x = f_i^{-1}(x'), y = f_i^{-1}(y'), x' \subseteq y' \to x \subseteq y$

(N1) $\bar{x} \nsubseteq \varnothing, x = f(\bar{x}) \to x \nsubseteq \varnothing$
(N2) $x \nsubseteq \varnothing, x = f_i^{-1}(x') \to x' \nsubseteq \varnothing$
(N3) $x \nsubseteq \varnothing \to y \nsubseteq \varnothing$ if exists κ in $\mathcal{UAP}(x, \varphi)$ such that $\varphi \vdash x[\kappa] \subseteq y$

(C1) $x \nsubseteq \varnothing, y = f(\bar{y}), z = g(\bar{z}) \to \perp$
if exists κ in $\mathcal{UAP}(x, \varphi)$ such that $\varphi \vdash x[\kappa] \subseteq y, \varphi \vdash x[\kappa] \subseteq z$
(C2) $x \subseteq y, y = g(\bar{y}), f_i^{-1}(x) = x', x' \nsubseteq \varnothing \to \perp$

Fig. 1. The axioms Ax

4.3 A Satisfiability Test

The Axioms Ax*:* we give in Figure 1 a set of axioms Ax on which our satisfiability test is based. To shorten the presentation, we write $\bar{x}$ a sequence $x_1, \ldots, x_k$ of variables; the length k is most of the time fixed by the context. For instance, writing $f(\bar{x})$ fixes k to be the arity of f. Similarly, we write $\bar{x} \subseteq \bar{y}$ to denote the constraints $x_1 \subseteq y_1, \ldots, x_k \subseteq y_k$, $\bar{x} \nsubseteq \varnothing$ to denote the constraints $x_1 \nsubseteq \varnothing, \ldots, x_k \nsubseteq \varnothing$.

Let us discuss the axioms from Ax: the axioms $(\mathsf{I1}-5)$ as well as $(\mathsf{N1}-2)$ and $(\mathsf{C2})$ are immediate. The axiom $(\mathsf{N3})$ is less obvious and subsumes other axioms that we may think about; it is more general than $x \nsubseteq \varnothing, x \subseteq y \to y \nsubseteq \varnothing$ as ϵ is always unavoidable and $x \subseteq y$ implies $\varphi \vdash x[\epsilon] \subseteq y$. It makes also redundant the axiom $x \nsubseteq \varnothing, x = f(\bar{x}) \to \bar{x} \nsubseteq \varnothing$, as $x = f(\bar{x})$ implies $\varphi \vdash x[i] \subseteq x_i$ for all $1 \leq i \leq \mathit{arity}(f)$. Furthermore, from the following constraint, using the axiom $(\mathsf{N3})$,

$$\begin{aligned} x \nsubseteq \varnothing \wedge x \subseteq y \wedge y = f(y_1, y_2) \wedge\ & g_2^{-1}(y_1) = y' \\ \wedge\ x \subseteq z \wedge\ f_1^{-1}(z) = z' \ & \wedge z' = g(z_1, z_2) \end{aligned}$$

one can deduce $y' \nsubseteq \varnothing, z_1 \nsubseteq \varnothing, z_2 \nsubseteq \varnothing$ as the paths $\langle f, 1\rangle.\langle g, 1\rangle, \langle f, 1\rangle.\langle g, 2\rangle$ are unavoidable for x.

Additionally, when the structure of sets of finite trees $\mathsf{P}(\mathcal{T}_\Sigma)$ is considered, a new axiom (OC) addressing the occur-check problem is required.

(OC) $x \nsubseteq \varnothing \to \perp$ if $\mathcal{UAP}(x, \varphi)$ is infinite.

This axiom is pretty unusual: traditionally [8, 19], occur-check is stated as the existence of a non-trivial path from a non-empty variable to itself. So, one may expect an axiom such as: (OC_1) $x \nsubseteq \varnothing \to \perp$ if $\varphi \vdash x[\kappa] \subseteq x$ for an annotated path $\kappa \neq \epsilon$. Unfortunately, this axiom is too strong as $\perp$ could be deduced from the constraint $x \nsubseteq \varnothing \wedge f_1^{-1}(x) = y \wedge y \subseteq x$ and obviously, this constraint is satisfiable over sets of finite trees $\mathsf{P}(\mathcal{T}_\Sigma)$—for example the valuation assigning $\{a, f(a)\}$ to x and $\{a\}$ to y is a solution. One may try to make the axiom (OC_1) weaker by reasoning not about annotated paths, but only about unavoidable annotated paths. We would have an axiom such as: $(\mathsf{OC}_2)\ \ x \not\subseteq \varnothing \rightarrow \bot \qquad$ if $\varphi \vdash x[\kappa] \subseteq x$ for $\kappa \in \mathcal{UAP}(x,\varphi)$ and $\kappa \neq \epsilon$. Unfortunately, the axiom (OC_2) is too weak; consider the constraint

$$\begin{array}{rl} x \not\subseteq \varnothing \wedge x \subseteq y_1 \wedge h_1^{-1}(y_1) = y_2 \wedge & y_2 = h(y_1) \\ x \subseteq z_1 \wedge \quad z_1 = h(z_2) & \wedge\, h_1^{-1}(z_2) = z_1 \end{array}$$

One can see that this constraint admits no solution over sets of finite trees; nevertheless, $\bot$ can not be deduced using the axiom (OC_2) as there is no non-trivial unavoidable annotated path κ leading from a variable to itself (intuitively, there is an unavoidable path from the intersection $y_1 \cap z_1$ to itself).

So, we require simply for the occur-check the set $\mathcal{UAP}(x,\varphi)$ to be infinite and we will prove below that this statement is correct.

Theorem 2. *The axioms* Ax *are valid for* $\mathsf{P}(\mathcal{T}_\Sigma^\omega)$. *The axioms* $\mathsf{Ax} + (\mathsf{OC})$ *are valid for* $\mathsf{P}(\mathcal{T}_\Sigma)$.

Proof. By routine check for the axioms $(\mathsf{I1}-\mathsf{5})$, $(\mathsf{N1}-\mathsf{2})$ *and* $(\mathsf{C2})$. *Using Proposition 3 for the axiom* $(\mathsf{N3})$. *Using Proposition 4 for the axiom* $(\mathsf{C1})$. *Finally, for the axiom* (OC), *in case of sets of finite trees* $\mathsf{P}(\mathcal{T}_\Sigma)$, *let us show that for any variable* x, *if* $x \not\subseteq \varnothing$ *in* φ *and* $\mathcal{UAP}(x,\varphi)$ *is infinite, then* φ *admits no solution. Function symbols from* Σ *occurring in* φ *have fixed arities, and thus, these arities must be bounded. As* $\mathcal{UAP}(x,\varphi)$ *is infinite, by Koenig' Lemma, we can extract from it an infinite sequence of annotated paths* $\kappa_0, \kappa_1, \kappa_2, \ldots$ *such that* κ_i *is a prefix of* κ_{i+1}. *So, by Proposition 3, for any tree* τ *in* $\sigma(x)$ *and for any* κ_i, $\tau[\kappa_i]$ *is defined. Obviously, this requirement can not be satisfied by some finite tree. So,* φ *is not satisfiable over* $\mathsf{P}(\mathcal{T}_\Sigma)$.

A Satisfiability Test: we describe here how the axioms Ax (or $\mathsf{Ax} + (\mathsf{OC})$) can be used for defining a test for satisfiability for a constraint φ.

We first consider the axioms from Ax and (OC) as inference rules: premises are left-hand sides of implications and conclusions the right-hand sides. Note that some rules like $(\mathsf{N3}), (\mathsf{C1}), (\mathsf{OC})$ have side conditions concerning unavoidable paths.

We denote $Sat(\varphi)$ as the least set of constraints containing φ and closed under those inference rules, that is if premises of an inference rule belong to $Sat(\varphi)$ and if the side condition is satisfied, then the conclusion belongs to $Sat(\varphi)$. We will moreover require that the variables from $Sat(\varphi)$ are exactly those from φ; this is guaranteed by an application of the axiom $x \subseteq x$ from Ax only for variables from φ.

Actually, the axioms $\mathsf{Ax} + (\mathsf{OC})$ can be viewed as an operator Sat over constraints that, given a constraint φ computes all consequences of φ under these axioms. Moreover, we can notice that for φ viewed as a collection of basic constraints, the set $\mathcal{UAP}(x,\varphi)$ is monotonic in the sense that $\varphi \subseteq \varphi'$ implies $\mathcal{UAP}(x,\varphi) \subseteq \mathcal{UAP}(x,\varphi')$. Thus, Sat is a monotonic operator over constraints: for two constraints φ, φ', if $\varphi \subseteq \varphi'$ then $Sat(\varphi) \subseteq Sat(\varphi')$. So, over the lattice with φ as least element, the operator Sat admits a least fix-point that we denote $Sat^\omega(\varphi)$.

Theorem 3. *if* $\bot \in Sat^\omega(\varphi)$ *then* φ *is unsatisfiable.*

Proof. Immediate from Theorem 2

We prove now the converse of Theorem 3: if $\perp$ does not belong to $Sat^{\omega}(\varphi)$ then φ admits a solution. We build from $Sat^{\omega}(\varphi)$ a valuation σ_S defined for any variable x as follows (i) $\sigma_S(x) = \varnothing$ if $x \nsubseteq \varnothing \notin Sat^{\omega}(\varphi)$ and (ii) otherwise, $\sigma_S(x)$ is the set of all trees τ satisfying

for all f, π, κ, y s.t. $\pi = \mathsf{res}(\kappa), \tau[\kappa]$ is defined and $Sat^{\omega}(\varphi) \vdash x[\kappa] \subseteq y$

if $y = f(\bar{y}) \in Sat^{\omega}(\varphi)$ then $\tau(\pi) = f$ and

if $z = f_i^{-1}(y), y \nsubseteq \varnothing \in Sat^{\omega}(\varphi)$ and $z \nsubseteq \varnothing \notin Sat^{\omega}(\varphi)$ then $\tau(\pi) \neq f$

Theorem 4. *if $\perp \notin Sat^{\omega}(\varphi)$ then σ_S is a solution for $Sat^{\omega}(\varphi)$ and thus for φ.*

4.4 A PSPACE algorithm

We devise in this section a PSPACE algorithm deciding whether a constraint φ is unsatisfiable, that is according to Theorems 3 and 4 deciding whether $\perp$ belongs to $Sat^{\omega}(\varphi)$. To present and prove correctness of our algorithm requires some auxiliary propositions.

Proposition 5. *For any constraint φ of size $|\varphi|$ with n variables,*

- *for any i, the size of $Sat^{i}(\varphi)$ is upper-bounded by $|\varphi|^2 + 2|\varphi| + 1$;*
- *there exists an integer p such that $p \leq n^2 + n + 1$ and $Sat^{p}(\varphi) = Sat^{\omega}(\varphi)$.*

Testing that an annotated path κ is unavoidable for a variable x and a constraint φ is polynomial:

Proposition 6. *Let φ be a constraint and κ an annotated path. Deciding whether $\kappa \in \mathcal{UAP}(x, \varphi)$ can be done in polynomial time in $|\varphi|$ and $|\kappa|$ where $|\varphi|$ is the size of φ and $|\kappa|$ the length of κ.*

Moreover, we can show that if premises and side-conditions for the axioms $(\mathsf{N3})$ $(\mathsf{C1})$ hold for some annotated paths then they hold for annotated paths whose length is at most exponential in the number of variables in φ,

Proposition 7. *Let φ be a constraint and n the number of variables occurring in φ.*

- *for all κ in $\mathcal{UAP}(x, \varphi)$ such that $\varphi \vdash x[\kappa] \subseteq y$, there exists κ' in $\mathcal{UAP}(x, \varphi)$ such that $\varphi \vdash x[\kappa'] \subseteq y$ and $|\kappa'| \leq 2^n$.*
- *for all κ in $\mathcal{UAP}(x, \varphi)$ such that $\varphi \vdash x[\kappa] \subseteq y$ and $\varphi \vdash x[\kappa] \subseteq z$ with $y = f(\bar{y})$ and $z = g(\bar{z})$ then there exists κ' in $\mathcal{UAP}(x, \varphi)$, y', z' two variables and f', g' two different function symbols such that $\varphi \vdash x[\kappa'] \subseteq y'$ and $\varphi \vdash x[\kappa'] \subseteq z'$ with $y' = f'(\bar{y'})$ and $z' = g'(\bar{z'})$ and $|\kappa'| \leq 2^{n^2}$.*

Finally, we show that $\mathcal{UAP}(x, \varphi)$ is infinite if it contains a path sufficiently long and this length on which infiniteness can be tested is simply exponential in the number of variables occurring in φ,

Proposition 8. *Let φ be a constraint and n the number of variables occurring in φ, if $\mathcal{UAP}(x, \varphi)$ contains a path κ of length equal $2^n + 1$ then $\mathcal{UAP}(x, \varphi)$ is infinite.*

Proposition 9. *Testing side-conditions for the inference rules $(\mathsf{N3})$, $(\mathsf{C1})$ and (OC) is in PSPACE.*

Proof. *We present a non-deterministic polynomial-space algorithm to test the side-condition for the rule* (N3)*; algorithms for* (C1) *and* (OC) *are similar.*

This algorithm is based on the result of Proposition 7, that is if there exists an unavoidable path κ for x in φ leading to y then there is an unavoidable path κ' of length at most 2^n for x in φ leading to y.

> $V := \{x\}$; $V_n := \varnothing$; $c := 1$
> *While* $y \notin V$ *and* $c \leq 2^n$ *do*
>   *Guess* $f \in \Sigma$ *and* $i \leq arity(f)$ *s.t.* $u = f(\bar{v}) \in \varphi$ *for some* $u \in V$
> $V_n := \{w \mid \varphi \vdash z[\langle f, i\rangle] \subseteq w$ *and* $z \in V\}$
> $c := c + 1$; $V := V_n$
> *Endwhile*
> *Return* ($y \in V$)

Obviously, this algorithm uses only polynomial space in n: the cardinality of the sets V and V_n is bounded by n and the counter c can be coded on $\log(2^n) \leq n + 1$ bits. This non-deterministic algorithm can be turned into a deterministic one using the Savitch's theorem [22] (NPSPACE(n) = PSPACE(n^2)).

Proposition 10. *For any constraint φ, $Sat^{i+1}(\varphi)$ can be computed from $Sat^i(\varphi)$ with an algorithm running in polynomial-space in $|\varphi|^2 + 2|\varphi| + 1$.*

Proof. *The computation is done as follows: we start from a constraint S equal to $Sat^i(\varphi)$ and we iterate until S is unchanged the following steps:*

1. *we saturate S by the axioms $(I1-5)$, $(N1-2)$ and $(C2)$ in polynomial time in the size of $Sat^i(\varphi)$ (upper-bounded by $|\varphi|^2 + 2|\varphi| + 1$; Proposition 5);*
2. *for any variable x and any constraint $y \not\subseteq \varnothing$, we test if the side-condition of $(N3)$ holds for $\mathcal{UAP}(x, Sat^i(\varphi))$ in polynomial-space in the size of $Sat^i(\varphi)$ (Proposition 9); if it holds, we add $y \not\subseteq \varnothing$ to S;*
3. *for any variable x such that $x \not\subseteq \varnothing \in S$, any pair of constraints $y = f(\bar{y}), z = g(\bar{y})$ from φ, we test if the side-condition of $(C1)$ holds for $\mathcal{UAP}(x, Sat^i(\varphi))$ in polynomial-space in the size of $Sat^i(\varphi)$ (Proposition 9); if it holds, we add $\perp$ to S;*
4. *for an interpretation over sets of finite trees: for any variable x such that $x \not\subseteq \varnothing \in S$, we test if the side-condition of (OC) holds for $\mathcal{UAP}(x, Sat^i(\varphi))$ in polynomial-space in the size of $Sat^i(\varphi)$ (Proposition 9); if it holds, we add $\perp$ to S;*

The constraint $Sat^{i+1}(\varphi)$ is the constraint S obtained at the end of the iteration process. As the size of $Sat^{i+1}(\varphi)$, is bounded by $|\varphi|^2 + 2|\varphi| + 1$, this latter provides also an upper-bound on the number of possible iterations.

Theorem 5. *Satisfiability for atomic set constraints with projection is in PSPACE.*

Proof. *Immediate, combining results from Propositions 5 and 10.*

References

1. A. Aiken. Set Constraints: Results, Applications, and Future Directions. In *Proc. of the* 2^{rd} *International Workshop on Principles and Practice of Constraint Programming*, LNCS 874, pages 326–335. 1994.

2. A. Aiken. Introduction to Set Constraint-Based Program Analysis. In *Science of Computer Programming*, 35(2): 79-111, 1999.
3. 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, 1993.
4. A. Aiken, D. Kozen and E. Wimmers. Decidability of Systems of Set Constraints with Negative Constraints. *Information and Computation*, 122(1):30–44, oct 1995.
5. L. Bachmair, H. Ganzinger and U. Waldmann. Set constraints are the monadic class. In *Proc. of the* 8^{th} *IEEE Symp. on Logic in Computer Science*, pages 75–83, 1993.
6. W. Charatonik and L. Pacholski. Negative Set Constraints with Equality. In *Proc. of the* 9^{th} *IEEE Symp. on Logic in Computer Science*, pages 128–136, 1994.
7. W. Charatonik and L. Pacholski. Set Constraints with Projections are in NEXPTIME. In *Proc. of the* 35^{th} *Symp. on Foundations of Computer Science*, pages 642–653, 1994.
8. W. Charatonik and A. Podelski. Set Constraints with Intersection. In *Proc. of the* 12^{th} *IEEE Symp. on Logic in Computer Science*, pages 362–372, 1997.
9. W. Charatonik and A. Podelski. Co-definite Set Constraints. In *Proc. of the* 9^{th} *International Conference on Rewriting Techniques and Applications*, LNCS 1379, pages 211–225, 1998.
10. W. Charatonik and J.-M. Talbot. Atomic Set Constraints with Projection. *Research Report Max-Planck-Institut für Informatik.* MPI-I-02-008, 2002.
11. P. Devienne, J.-M. Talbot and S. Tison. Solving classes of set constraints with tree automata. In *Proc. of the* 3^{rd} *International Conference on Principles and Practice of Constraint Programming*, LNCS 1330, pages 62–76, 1997.
12. R. Gilleron, S. Tison and M. Tommasi. Solving Systems of Set Constraints with Negated Subset Relationships. In *Proc. of the* 34^{th} *Symp. on Foundations of Computer Science*, pages 372–380, 1993.
13. N. Heintze. *Set Based Program Analysis*. PhD thesis, Carnegie Mellon University, 1992.
14. N. Heintze and J. Jaffar. A Finite Presentation Theorem for Approximating Logic Programs. In *Proc. of the* 17^{th} *Symp. on Principles of Programming Languages*, pages 197–209, 1990.
15. N. Heintze and J. Jaffar. A Decision Procedure for a Class of Herbrand Set Constraints. In *Proc. of the* 5^{th} *IEEE Symp. on Logic in Computer Science*, pages 42–51, 1990.
16. N. Heintze and J. Jaffar. Set constraints and set-based analysis. In *Proc. of the Workshop on Principles and Practice of Constraint Programming*, LNCS 874, pages 281–298, 1994.
17. F. Henglein and J. Rehof. The complexity of subtype entailment for simple types. In *Proc. of the* 12^{th} *IEEE Symp. on Logic in Computer Science*, pages 362–372, 1997.
18. J. Niehren, M. Müller and J.-M. Talbot. Entailment of Atomic Set Constraints is PSPACE-Complete. In *Proc. of the* 14^{th} *IEEE Symp. on Logic in Computer Science*, pages 285–294, 1999.
19. M. Müller, J. Niehren and A. Podelski. Inclusion Constraints over Non-Empty Sets of Trees. In *Proc. of* 7^{th} *International Joint Conference CAAP/FASE - (TAPSOFT'97)*, LNCS 1214, pages 345–356, 1997.
20. L. Pacholski and A. Podelski. Set Constraints: A Pearl in Research on Constraints. In *Proc. of the* 3^{rd} *International Conference on Principles and Practice of Constraint Programming*, LNCS 1330, pages 549–561, 1997. Tutorial.
21. J. C. Reynolds. Automatic computation of data set definitions. *Information Processing*, 68:456–461, 1969.
22. Savitch, W. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences 4*(2), 177–192.
23. K. Stefansson. Systems of Set Constraints with Negative Constraints are NEXPTIME-Complete. In *Proc. of the* 9^{th} *IEEE Symp. on Logic in Computer Science*, pages 137–141, 1994.
24. T. E. Uribe. Sorted Unification using Set Constraints. In *Proc. of the* 11^{th} *International Conference on Automated Deduction*, LNAI 607, pages 163–177, 1992.