Relating Set Constraints and Constraint Logic Programming with Finite Sets*

Agostino Dovier
Dip. Scientifico-Tecnologico,
Univ. di Verona.
Strada Le Grazie,
37134 Verona (I).
dovier@sci.univr.it

Carla Piazza
Dip. di Matematica e Informatica,
Univ. di Udine.
Via Le Scienze 206,
33100 Udine (I).
piazza@dimi.uniud.it

Gianfranco Rossi
Dip. di Matematica,
Univ. di Parma.
Via M. D'Azeglio 85/A,
43100 Parma (I).
gianfr@prmat.math.unipr.it

Abstract

We compare two (apparently) rather different set-based constraint languages, and we show that, in spite of their different origins and aims, there are large classes of constraint formulae for which both proposals provide suitable procedures for testing constraint satisfiability with respect to a given privileged interpretation. Specifically, we present a technique for reducing any Set constraint to a purely set-theoretic first-order formula. This formula can be viewed as a constraint of a Constraint Logic Programming (CLP) language working on set-based domains. In particular, we consider the language CLP($\mathcal{SET}$) but similar considerations could be done starting from any CLP language endowed with (nested) sets. Moreover, we show how the satisfiability over domains of infinite sets (that are not usually dealt with by CLP($\mathcal{SET}$)) of a non-trivial sub-class of Set constraints can be checked by the constraint solver of CLP($\mathcal{SET}$).

Keywords: Set constraints, Constraint Logic Programming with Sets.

---

*A preliminary version of this paper has been presented at the *Workshop on Set Constraints and Constraint-based Program Analysis* (Pisa, Italy, October 1998) and at the *APPIA-GULP-PRODE'98* (La Coruña, Spain, July 1998).

1 Introduction

Generally speaking, *set-based constraints* can be defined as formulae of a first-order language $\mathcal{L}$ whose literals are built out of classical set-theoretic symbols.

A particular class of set-based constraints, simply called *Set constraints*, have been extensively studied in recent years and used as a convenient formalism for program analysis (see [1] for a brief introduction to Set constraints).

Basic Set constraints are conjunctions of literals of the form

$$e_1 \subseteq e_2$$

where e_1 and e_2 are *set expressions* constructed using variables, constant and function symbols, and the union, intersection, and complement of set expressions. Set expressions denote sets of Herbrand terms. Several extensions to basic Set constraints have also been defined: in particular, we consider those adding projections and negative constraints $e_1 \not\subseteq e_2$ [2, 22].

The problem of dealing with set-based constraints has been faced also in the context of constraint logic programming (CLP) languages [27] with sets.

$CLP(SC)$ [28, 20] is one such language, where the admissible constraints are exactly Set constraints. In $CLP(SC)$, therefore, the constraints are based on the inclusion symbol, and the domain of the interpretation is that of the rational solutions of Set constraints. The constraint solving algorithms are drawn from the decidability results for Set constraints.

Other CLP languages with sets, such as CLPS [29], Conjunto [21], and $CLP(\mathcal{SET})$ [19, 17], on the contrary, are based on quite different notions of set-based constraints. In this paper we will concentrate, in particular, on the CLP language $CLP(\mathcal{SET})$ ($CLP(\mathcal{SET})$ is the CLP counterpart of the extended logic language {log} [16]). A $CLP(\mathcal{SET})$-constraint is a conjunction of literals of the form

$$t_1\ \pi_1\ t_2 \quad \text{or} \quad \pi_2(t_1, t_2, t_3), \quad \pi_1 \text{ in } \{\in, \notin, \doteq, \neq, \parallel, \not\parallel\}, \pi_2 \text{ in } \{\cup_3, \not\cup_3\}$$

where $\parallel$ is the binary predicate symbol stating whether two sets are disjoint, $\cup_3$ is the predicative version of the union operator, and the t_i are terms possibly denoting finite sets that can be *nested* (i.e., sets of sets), and *hybrid* (i.e., sets of standard terms, not denoting sets). Moreover, more complex formulae, such as *Restricted Universal Quantifiers* (see, e.g., [8]) and *intensionally defined sets* [6], can be defined in the language itself by exploiting the expressive power of the whole CLP($\mathcal{SET}$) language.

It is worth mentioning that most of the results obtained for $CLP(\mathcal{SET})$, and for set-based languages in general, are strongly related to the results obtained in the *Computable Set Theory* area ([8]). C.S.T. was mainly developed at the New York University in the Eighties, in reply to the need to potentiate the inferential engine for Theorem provers and for the implementation of the imperative language SETL ([32]). The general problem was that of identifying computable classes of formulae of suitable sub-theories of the general Zermelo-Fraenkel set-theory. Although a lot of similarities exist between the area of Set constraints and C.S.T., unfortunately no explicit connection has been established so far between them.

Since the two classes of set-based constraints mentioned above, namely $CLP(\mathcal{SET})$-constraints and Set constraints, differ considerably each other, the challenging problem is trying to compare them and showing that, in spite of their different origins and aims, there are large classes of set-based constraints for which both proposals provide suitable procedures for testing constraint satisfiability with respect to a given privileged interpretation.

The result of this comparison can serve immediately to allow the CLP($\mathcal{SET}$) interpreter to be used as an easily available tool for testing Set constraints.[1] With this respect, CLP($\mathcal{SET}$) can be seen just as an alternative to CLP(SC), with some *pros* and *cons* with respect to it, as it will emerge from the comparison itself.

The additional value of using the CLP($\mathcal{SET}$) framework, however, resides in the various additional features of CLP($\mathcal{SET}$)—such as nested sets, other set predicates ($\in$, $\notin$,...), the availability of the computed solutions—as well as in those that will be possibly added to CLP($\mathcal{SET}$) in the near future—such as multisets and compact-lists [18]. In the long-term, this extra power of the language would allow one to devise new more general notions of set-based constraints. This could be exploited, for instance, to extend the notion of *set-based program analysis* usually found in the literature (e.g.,[26, 13, 7]), so as to be able to capture different properties of programs, thus allowing, after all, different-grained set-based abstract interpretations [14].

The main technical result of the paper is to show how a constraint solver (namely, the CLP($\mathcal{SET}$) solver) which is able to decide satisfiability of set-based formulae over finite sets—by using a "constructive" approach—can be used as well to decide the satisfiability of non-trivial sub-classes of Set constraints over *infinite* sets of terms. The result is specific for the CLP($\mathcal{SET}$) language but the techniques used are general enough to be easily adaptable to other CLP languages endowed with (nested) sets (e.g., CLPS [29]).

The approach adopted is that of reducing the constraint satisfiability problem over one class of constraints, namely Set constraints, to the same problem for the other class, namely CLP($\mathcal{SET}$)-constraints. This is obtained by: using a uniform approach and notation in presenting the two proposals; augmenting the CLP($\mathcal{SET}$) language of [19] by allowing it to deal with also the union and disjoint constraints, as recently proposed in [17]; developing a suitable rewriting procedure for translating the whole class of Set constraints to equivalent CLP($\mathcal{SET}$)formulae; characterizing a few sub-classes of Set constraints for which satisfiability can be always decided using CLP($\mathcal{SET}$), even though solutions of the Set constraints involve infinite sets.

Note that the comparison is mainly concerned with the expressive power of the two languages. However, some hints about computational complexity of the constraint satisfiability procedures are also put forward.

The paper is organized as follows. In Section 2 we briefly present the two languages to be compared and we point out problems involved in translating one language into the other. In Section 3 we present a procedure for translating Set constraints into CLP($\mathcal{SET}$) formulae and we show that it can be used as a partial test for Set constraint satisfiability. The following three sections (Sections 4, 5, and 6) precisely characterize three sub-classes of Set constraints for which CLP($\mathcal{SET}$) can be used to obtain a complete satifiability test. In particular, for the class $\mathcal{S}_1$ (Section 5) we show that CLP($\mathcal{SET}$) can be used to decide satisfiability of Set constraints even in presence of solutions which involve infinite sets. In Section 7 we analyze the expressive power of the three analyzed classes, compared to well-known classes of Set constraints. Finally, some conclusions are drawn in Section 8 along with some hints about possible future work.

2 The Languages to be Compared

The main differences between the two classes of languages we are going to compare, namely Set constraints and CLP($\mathcal{SET}$)-constraints, are summarized below.

[1] An interpreter of the CLP($\mathcal{SET}$) language—implemented in SICStus Prolog—is available at http://www.math.unipr.it/~gianfr/setlog/interpreter

2.1 Set Constraints

The notation and the definitions sketched here can be found, e.g., in [1, 28].

Constant symbols: $\underline{0}, \underline{1}$, and a nonempty set of first-order constant symbols (e.g., a, b).

Function symbols: $\cup, \cap, \complement$, where $\complement$ stands for the complement operator, and, possibly, a set of function symbols (e.g., f, g). Also their projections on the i-th argument (e.g., for f, f_i^{-1}) can be used as term-constructors. Let Σ be the set of constant and function symbols save $\underline{0}, \underline{1}, \cup, \cap, \complement$ and $ar : \Sigma \longrightarrow \mathbb{N}$ the arity function.

Σ is assumed to be finite. However, there is only one point in which this requirement is needed, and it will be discussed in Section 3.

Predicate symbol: $\subseteq$ (used also in negative literals).

Interpretation domain: The set of (flat) sets of ground first-order terms over the signature Σ, i.e., the powerset $\wp(T_\Sigma)$ of T_Σ; we will also use $\wp_{fin}(T_\Sigma)$ to denote the set of *finite* subsets of T_Σ.

Interpretation: With $(\cdot)^{\mathcal{P}}$ we denote the interpretation function for the language of Set constraints, defined as:

$$\begin{array}{rcll} (f(a_1, \ldots, a_n))^{\mathcal{P}} & = & \{f(t_1, \ldots, t_n) \,:\, \bigwedge_{i=1}^n t_i \in a_i^{\mathcal{P}}\} & \text{for } f \in \Sigma \\ (f_i^{-1}(a))^{\mathcal{P}} & = & \{t_i \,:\, \exists t_1, \ldots, t_n \, f(t_1, \ldots, t_n) \in (a)^{\mathcal{P}}\} & \text{for } f \in \Sigma \end{array}$$

$(\cdot)^{\mathcal{P}}$ is extended to set-based operators in the intuitive way.

2.2 CLP($\mathcal{SET}$)-Constraints

The notation and the definitions of this subsection can be found, e.g., in [16, 17].

Constant symbols: $\emptyset$, and, possibly, a set of first-order constant symbols (e.g., a, b).

Function symbols: $\{\cdot \,|\, \cdot\}$, and, possibly, a set of function symbols (e.g., f, g). We denote as Σ the set containing all constant and function symbols, including $\emptyset$ and $\{\cdot \,|\, \cdot\}$. We assume the following notational conventions:

$$\{t_0 \,|\, \{t_1 \,|\, \cdots \{t_n \,|\, X\} \cdots\}\}$$

is written as $\{t_0, \ldots, t_n | \, X\}$, and, when X is $\emptyset$, simply as $\{t_0, \ldots, t_n\}$.

Predicate symbols: $\doteq, \in, \cup_3, \|$ (used also in negative literals), where $\|$ is the binary predicate symbol stating whether two sets are disjoint, and $\cup_3$ is the predicative version of the union operator.

Interpretation domain: the set T_Σ of ground first-order terms over Σ modulo the least equivalence relation $\equiv$ obtained by the reflexive, symmetric, and transitive closure of the following axioms:

$$\begin{array}{rrcl} \forall r, s, t \in T_\Sigma & \{t \,|\, \{s \,|\, r\}\} & \equiv & \{s \,|\, \{t \,|\, r\}\} \\ \forall r, s \in T_\Sigma & \{s \,|\, \{s \,|\, r\}\} & \equiv & \{s \,|\, r\} \\ \forall s_1, \ldots, s_n, t_1, \ldots, t_n \in T_\Sigma & \bigwedge_{i=1}^n s_i \equiv t_i & \rightarrow & \\ & f(s_1, \ldots, s_n) & \equiv & f(t_1, \ldots, t_n), \quad f \in \Sigma \end{array}$$

Interpretation:

$$t^{\mathcal{T}} = [t]_{\equiv} \quad \text{for any term} \quad t \in T_{\Sigma}$$

where $[t]_{\equiv}$ is the canonical representative of the congruence class of the term t. In particular, each term without occurrences of $\{\cdot\,|\,\cdot\}$ is interpreted as the term itself—as usual in Herbrand models—, whereas the interpretation of terms whose outermost function symbol is $\{\cdot\,|\,\cdot\}$ allows them to be viewed as hereditarily finite hybrid sets. When $\Sigma = \{\emptyset, \{\cdot\,|\,\cdot\}\}$, the considered domain is isomorphic to the domain of the hereditarily finite and well-founded sets [8, 18].

Moreover, equality is the set-theoretic equality, implied by the two equational axioms for $\{\cdot\,|\,\cdot\}$ above (see [18]), i.e., $s \doteq t$ if and only if $[s]_{\equiv} = [t]_{\equiv}$; $s \in t$ holds if $[t]_{\equiv}$ is of the form $\{t_1, t_2, \ldots, [s]_{\equiv} \,|\, t'\}$; $\cup_3(r, s, t)$ holds, intuitively, when $[t]_{\equiv}$ represents the set $[r]_{\equiv} \cup [s]_{\equiv}$, while $\parallel (s,t)$ holds when $[s]_{\equiv}$ and $[t]_{\equiv}$ do not share elements.

Hereafter, the structure $\langle \wp(T_{\Sigma}), (\cdot)^{\mathcal{P}} \rangle$ used in Set constraints will be refered as $\mathcal{P}$, while the structure $\langle T_{\Sigma}/\equiv, (\cdot)^{\mathcal{T}} \rangle$ used for CLP($\mathcal{SET}$) constraints will be denoted by $\mathcal{T}$.

Example 2.1 *The following formulae are all CLP($\mathcal{SET}$)-constraints*

$$\begin{array}{ll} X \notin f(Y,Z), & \textit{always satisfiable in } \mathcal{T}; \\ \{a\,|\,X\} \doteq \{b\,|\,Y\}, & \textit{satisfiable in } \mathcal{T} \textit{ (by } [X/\{b\,|\,N\}, Y/\{a\,|\,N\}]); \\ a \in X \wedge X \neq \{Y\}, & \textit{satisfiable in } \mathcal{T} \textit{ (by } [X/\{a\,|\,N\}] \wedge N \neq \emptyset). \end{array}$$

2.3 The Satisfiability Problem

Several authors have investigated the problem of checking the satisfiability of different classes of Set constraints, using different techniques and methods for solving them ([3, 2, 22, 23]). In particular in [9] it has been shown that the satisfiability problem for Set constraints is decidable and NEXPTIME complete.

Also CLP($\mathcal{SET}$)-constraints have been proved to be always decidable and a constraint solver for them has been described in detail in [19, 17]. The technique used by the CLP($\mathcal{SET}$) constraint solver is based on a reduction of any given constraint to a normal form which is always satisfiable. The computational complexity of this problem—for the primitive constraints mentioned above—is shown to be NP-complete in [31]. The CLP($\mathcal{SET}$) constraint solver, moreover, always returns also the complete (though not always minimal) set of computed solutions by which the given constraint is satisfiable.

2.4 Extended CLP($\mathcal{SET}$)-Constraints

Other basic set-theoretical operations can be easily defined in terms of the existing ones in both languages. In particular, equality can be defined in the language of Set constraints as:

$$s \doteq t \quad \text{if and only if} \quad s \subseteq t \wedge t \subseteq s,$$

whereas $\subseteq$, $\not\subseteq$, and the ternary predicate $\cap_3$ (with the obvious interpretations) can be easily defined in CLP($\mathcal{SET}$) as follows:

$$\begin{array}{rll} s \subseteq t & \text{if and only if} & \cup_3(s,t,t) \\ s \not\subseteq t & \text{if and only if} & \exists Z (Z \in s \wedge Z \notin t) \\ \cap_3(r,s,t) & \text{if and only if} & \exists R, S (\cup_3(R,t,r) \wedge \cup_3(S,t,s) \wedge R \parallel S). \end{array}$$

Furthermore, being an instance of the general CLP scheme, CLP($\mathcal{SET}$) inherits all the computational capabilities of this scheme. In particular, it is possible to define *Restricted*

Universal Quantifiers (*RUQ*) by means of CLP($\mathcal{SET}$) clauses. *RUQ*s are formulae of the form $(\forall X \in s)G$, with G an arbitrary CLP($\mathcal{SET}$)-goal containing X, whose meaning is the quantified implication $\forall X((X \in s) \rightarrow G)$. Given a program P, any *RUQ*-atom $(\forall X \in s)G[X, Y_1, \ldots, Y_n]$ in P is replaced by a new atom $\mathsf{forall_G}(s, Y_1, \ldots, Y_n)$ which is defined by the following two clauses added to P:[2]

$$\mathsf{forall_G}(\emptyset, Y_1, \ldots, Y_n).$$
$$\mathsf{forall_G}(\{A \mid R\}, Y_1, \ldots, Y_n) \leftarrow A \notin R \wedge G^{[X/A]} \wedge \mathsf{forall_G}(R, Y_1, \ldots, Y_n).$$

This technique is easily generalized so as to implement by CLP($\mathcal{SET}$)-clauses also the more general form of RUQs

$$(\forall X_1 \in s_1) \cdots (\forall X_n \in s_n) \exists Z_1, \ldots, Z_m G[X_1, \ldots, X_n, Z_1, \ldots, Z_m]$$

where $X_1, \ldots, X_n$ are all distinct variables which do not occur in $s_1, \ldots, s_n$.

Hereafter, CLP($\mathcal{SET}$)-constraints augmented with Restricted Universal Quantifiers and the other basic set-theoretic operations $\subseteq$ and $\cap_3$ will be called *CLP($\mathcal{SET}$)$^+$-constraints.*

The problem with this extension of the language is that, while checking the satisfiability of CLP($\mathcal{SET}$)-constraints is always decidable (and in NP), checking the satisfiability of CLP($\mathcal{SET}$)$^+$-constraints may lead to non-termination, due to the presence of RUQ's atoms, whenever the input constraint admits solutions that involve infinite sets (see, e.g., Example 3.4). However, in Section 5 we identify a class of Set constraints whose satisfiability can be *finitely* checked testing a CLP($\mathcal{SET}$)$^+$-constraint obtained from it.

2.5 Comparing the two Languages

The following general notion of translation between two languages will be useful in the sequel to compare the two set-based languages we are interested in.

Definition 2.2 *Given two first order languages $\mathcal{L}_1$ and $\mathcal{L}_2$, two interpretations $\mathcal{M}_1$ and $\mathcal{M}_2$, two classes of formulae $\mathcal{C}_1$ and $\mathcal{C}_2$, a* translation *of $\mathcal{C}_1$ to $\mathcal{C}_2$ is a function $\phi : \mathcal{C}_1 \rightarrow \mathcal{C}_2$. ϕ is a* conservative translation *under $\mathcal{M}_1$ and $\mathcal{M}_2$ if:*

$$\exists \sigma_1 \mathcal{M}_1 \models \varphi[\sigma_1] \quad \textit{if and only if} \quad \exists \sigma_2 \mathcal{M}_2 \models \phi(\varphi)[\sigma_2]$$

for each formula $\varphi \in \mathcal{C}_1$, where σ_i is a valuation for the variables into the domain $\mathcal{M}_i$, for $i = 1, 2$.

The main difficulties in defining such translations from (sub-classes of) Set constraints to CLP($\mathcal{SET}$)$^+$-constraints come from the following features of the two languages:

- Set constraints can be satisfied using *infinite* sets, whereas the domain of CLP($\mathcal{SET}$)-constraints $\mathcal{T}$ can have only finite sets;[3]
- the atoms $X \doteq f(X_1, \ldots, X_n)$ and $X \doteq f_k^{-1}(Y)$ of Set constraints, which represent a form of cartesian product and projection, are special cases of *intensionally defined sets*, that is, they define a set by stating a property that must be satisfied by all its elements rather than by explicitly enumerating all of them. This calls into play RUQs when moving to CLP($\mathcal{SET}$)$^+$-constraints and, hence, it requires to face adequately the problem of possible non-termination mentioned above.

[2] A procedure that transforms CLP($\mathcal{SET}$)-clauses with *RUQ*s to equivalent CLP($\mathcal{SET}$)-clauses without *RUQ*s is described in detail in [16].

[3] This is also a main feature of Computable Set Theory results. In this context, set theories have the property to "reflect on the finite".

As far as the reverse translation is concerned, namely the problem of encoding CLP($\mathcal{SET}$)-constraints by Set constraints, we wish to point out that CLP($\mathcal{SET}$) can be used also to decide formulae which are not expressible as Set constraint formulae. The problem arises with the encoding of the membership predicate. If sets are "flat" (i.e., sets of elements that are not themselves sets) there are no problems. Otherwise, one can try to translate CLP($\mathcal{SET}$)-constraints into Set constraints using a unary function symbol f to denote the singleton set constructor and mapping $t \in s$ into $f(t) \subseteq s$. Assume, for the sake of simplicity, that Σ for CLP($\mathcal{SET}$) is simply $\{\emptyset, \{\cdot \,|\, \cdot\}\}$. First we note that we can not use the constant $\underline{0}$ to encode $\emptyset$: $\emptyset \in \emptyset$, which is false, would be mapped to $f(\underline{0}) \subseteq \underline{0}$, which is true. So we need at least a constant symbol a such that $\emptyset$ is translated into a. In this way, the translation I of CLP($\mathcal{SET}$)-terms would be:

$$\left\{\begin{array}{rcll} I(X) & = & X & X \text{ a variable} \\ I(\emptyset) & = & a & \\ I(\{s\}) & = & f(I(s)) & \\ I(\{s \,|\, t\}) & = & f(I(s)) \cup I(t) & t \not\equiv \emptyset \end{array}\right.$$

The translation of $s \in t$ would be $I(\{s\}) \subseteq I(t)$.

A translation of this kind however does not preserve satisfiability, since the two set terms $\{\{s,t\}\}$ and $\{\{s\},\{t\}\}$ have the same translation $f(f(I(s)) \cup f(I(t))) = f(f(I(s))) \cup f(f(I(t)))$. Thus, for instance, the CLP($\mathcal{SET}$)-constraint $\{\emptyset\} \in \{\{\emptyset, \{\emptyset\}\}\}$, which is false, is translated to $f(f(a)) \subseteq f(a \cup f(a))$, which is satisfiable as a Set constraint.

The problem is the "rigid" interpretation of function symbols for Set constraints and it is related to the loosing of interdependence of variables. Sets having a level of nesting greater than one can be "confused" with other sets of the same level of nesting.

3 Translating Set Constraints to CLP($\mathcal{SET}$)$^+$-constraints

In this section we show a possible translation—though not a conservative translation—from the whole class of Set constraints to CLP($\mathcal{SET}$)$^+$-constraints is feasible. The translation can be split into two parts: first we reduce the given Set constraint C to an equivalent Set constraint C' in flat form (actually, an extension of the flat form of INES-constraints [30]); then we translate C' to the corresponding CLP($\mathcal{SET}$)$^+$-constraint.

Definition 3.1 *A Set constraint in* flat form *is a conjunction of literals of the form:*

$$\left\{\begin{array}{lll} Z \doteq \underline{0} & Z \doteq \underline{1} & Z \doteq \complement(X) \\ Z \;\pi\; f(X_1,\ldots,X_n) & Z \;\pi\; f_k^{-1}(X) & \textit{where } \pi \textit{ in } \{\subseteq, \supseteq\} \\ Z \doteq X \cup Y & Z \doteq X \cap Y & \\ X \subseteq Y & X \not\subseteq Y & \end{array}\right.$$

where X, X_i, Y, and Z are variables and f is a function or a constant symbol.

Given a Set constraint C (possibly including $\not\subseteq, \complement, f_i^{-1}$), it is straightforward (and always possible) to transform C into an equi-satisfiable Set constraint in flat form C' by using standard techniques.

The following is the definition of a function ϕ that translates any Set constraint C to the corresponding CLP($\mathcal{SET}$)$^+$-constraint. Observe that function symbols $\cup$ and $\cap$ for Set constraints are replaced by ternary predicate symbols $\cup_3$ and $\cap_3$. Moreover, as shown in Section 2, in CLP($\mathcal{SET}$) $s \subseteq t$ stands for $\cup_3(s,t,t)$ and $s \not\subseteq t$ stands for $\exists Z\,(Z \in s \wedge Z \notin t)$.

$$
\begin{array}{r|c|ll}
X \; \pi \; Y & \stackrel{\phi}{\mapsto} & X \; \pi \; Y & \pi \text{ in } \{\subseteq, \not\subseteq\} \\
Z \doteq \underline{0} & \stackrel{\phi}{\mapsto} & Z \doteq \emptyset & \\
Z \doteq \underline{1} & \stackrel{\phi}{\mapsto} & Z \doteq U & \\
Z \doteq X \; \pi \; Y & \stackrel{\phi}{\mapsto} & \pi_3(X, Y, Z) & \pi \text{ in } \{\cup, \cap\} \\
Z \doteq \complement(X) & \stackrel{\phi}{\mapsto} & \cup_3(X, Z, U) \wedge X \parallel Z & \\
\hline
f(X_1, \ldots, X_n) \subseteq Y & \stackrel{\phi}{\mapsto} & (\forall Z_1 \in X_1) \cdots (\forall Z_n \in X_n)(f(Z_1, \ldots, Z_n) \in Y) & \\
Y \subseteq f(X_1, \ldots, X_n) & \stackrel{\phi}{\mapsto} & (\forall Z \in Y) \exists Z_1, \ldots, Z_n & \\
 & & \qquad (Z \doteq f(Z_1, \ldots, Z_n) \wedge Z_1 \in X_1 \wedge \ldots \wedge Z_n \in X_n) & \\
f_k^{-1}(X) \subseteq Z & \stackrel{\phi}{\mapsto} & (\forall Y \in X) \forall Y_1 \ldots Y_n \, (Y \doteq f(Y_1, \ldots, Y_k, \ldots, Y_n) \rightarrow Y_k \in Z) & \\
Z \subseteq f_k^{-1}(X) & \stackrel{\phi}{\mapsto} & (\forall Y \in Z) \exists Y_1 \ldots Y_n \, (f(Y_1, \ldots, Y_{k-1}, Y, Y_{k+1}, \ldots, Y_n) \in X) &
\end{array}
$$

Figure 1: Rewriting rules for ϕ

Definition 3.2 *Let C be a Set constraint and C' be the equivalent constraint in flat form. Let U be a new variable not occurring in C. The translation for each literal of the Set constraint C is defined by the rewriting rules of Fig. 1.*
Moreover, if $C' = C'_1 \wedge \ldots \wedge C'_n$ and $vars(C') = \{Z_1, \ldots, Z_m\}$, then $\phi(C') = \phi(C'_1) \wedge \ldots \wedge \phi(C'_n) \wedge (Z_1 \subseteq U) \wedge \ldots \wedge (Z_m \subseteq U) \wedge U \not\doteq \emptyset$. If Σ is composed simply by a finite set of constant symbols $\{a_1, \ldots, a_n\}$, then add also the constraint $U \doteq \{a_1, \ldots, a_n\}$ to $\phi(C')$. Finally, put $\phi(C) = \phi(C')$.

The new variable U represents a universe to which we refer to for testing the satisfiability of the constraint after the translation.

The translation of $f_k^{-1}(X) \subseteq Z$ seems to cross the syntactical limits of CLP($\mathcal{SET}$)$^+$-constraints, hiding a form of negation. As a matter of fact, the formula on the right-hand part of this rule is equivalent to $(\forall Y \in X) \neg p(Y, Z)$, with the predicate p defined as $p(Y, Z) \leftarrow Y = f(Y_1, \ldots, Y_n) \wedge Y_k \notin Z$. Unfortunately, negation can not be well supported in CLP($\mathcal{SET}$), due to the undecidability of any model for a set theory.

Notwithstanding, we can avoid the hindrances of negation in this special case by assuming that the signature Σ for the language of Set constraints is finite and, hence, we can replace the constraint in the right-hand part of the rule by the CLP($\mathcal{SET}$)$^+$-constraint $(\forall Y \in X)\mathsf{negation\_of\_p}(Y, Z)$ where the predicate $\mathsf{negation\_of\_p}$ is defined as:

$$
\begin{array}{ll}
\mathsf{negation\_of\_p}(g(Y_1, \ldots, Y_m), Z). & \text{for all } g \in \Sigma, g \not\equiv f, ar(g) = m \\
\mathsf{negation\_of\_p}(f(Y_1, \ldots, Y_n), Z) \leftarrow Y_k \in Z. &
\end{array}
$$

We have defined a procedure for translating (general) Set constraints to CLP($\mathcal{SET}$)$^+$-constraints. Now we can prove that:

Theorem 3.3 *Let C be a Set constraint in flat form*

(1) If the constraint solver of CLP($\mathcal{SET}$)$^+$ terminates with failure (i.e., $\phi(C)$ is unsatisfiable), then C is unsatisfiable as a Set constraint;

(2) If the constraint solver of CLP($\mathcal{SET}$)$^+$ terminates with success (i.e., $\phi(C)$ is satisfiable), then C is satisfiable as a Set constraint.

Proof. (Sketch) First notice that each rewriting rule respects the (set theoretic) interpretation $(\cdot)^{\mathcal{P}}$ of Set constraints.

For (1) observe that any solution σ of C in $\mathcal{P}$ is a solution for $\phi(C)$ over a structure $\mathcal{T}'$ that extends $\mathcal{T}$ with all denumerable infinite and well-founded sets built from Σ. Now, the CLP$(\mathcal{SET})^+$ constraint solver tries to build a solution by adding elements to sets. It is a blind search strategy that tries to incrementally collect (possibly infinite) sets. If all the non-deterministic computations eventually halt with failure, then there are no solutions (neither finite nor infinite). Thus, if an infinite solution for C exists, the constraint solver can not halt with failure.

For (2) given a solution σ of $\phi(C)$ over $\mathcal{T}$, it is possible to compute a solution σ' of $\mathcal{C}$ over $\mathcal{P}$. In the proof of Theorem 4.4 the method is made explicit for a subclass of formulae. □

The problem with this translation emerges when (due to the presence of RUQs) the execution of the constraint solver of CLP$(\mathcal{SET})^+$ does not terminate. In this case the constraint solver is trying to build, element by element, an infinite solution for the given Set constraint C.

Example 3.4 *Let $\Sigma = \{a, f\}$. The Set constraint $X \subseteq f_1^{-1}(X) \wedge X \doteq \underline{1}$ is translated into*

$$(\forall W \in X)(f(X) \in X) \wedge X \doteq U \wedge X \subseteq U \wedge U \neq \emptyset$$

After a simple simplification step it becomes: $(\forall W \in U)(f(X) \in U) \wedge U = \{N \mid N'\}$ where N, N' are two new variables. The constraint solver will loop while trying to collect the answer: $U = \{N, f(N), f(f(N)), \ldots\}$.

In the following sections we will identify three subclasses of Set constraints for which the satisfiability test is decidable. In particular, for two of them, we will find a bound on the number of the computation steps of the constraint solver that, if reached, ensures infinite satisfiability of the Set constraint.

4 The Class $\mathcal{S}_0$

One class of Set constraints for which the translation ϕ is trivially a conservative one is the class $\mathcal{S}_f$ of *finitely satisfiable Set constraints* (i.e., the class of Set constraints for which we can find a solution in $\wp_{fin}(T_\Sigma)$). Although there are applications for which finitely satisfiable Set constraints (or, even, the larger class of them including membership) may turn out to be sufficiently expressive in set-based program analysis [25, 26] the ability to deal with also infinite solutions is a crucial requirement.

In this section we syntactically identify another very simple class of set constraints—we will call $\mathcal{S}_0$—which is not limited to finite solutions and for which we can prove that the translation ϕ is conservative. Moreover, we prove that the satisfiability test for $\mathcal{S}_0$ is NP-complete.

Definition 4.1 *Let $\mathcal{S}_0$ be the class of Set constraints composed only by set operators (no function symbols or projections are used). $\mathcal{S}_0$ is described by the following grammar (X denotes a generic variable):*

$$\begin{array}{rcl} \mathcal{S}_0 & ::= & C \mid C \wedge \mathcal{S}_0 \\ C & ::= & E \subseteq E \mid E \not\subseteq E \\ E & ::= & \underline{0} \mid \underline{1} \mid X \mid E \cap E \mid E \cup E \mid \complement(E) \end{array}$$

Notice that $\mathcal{S}_0 \not\subseteq \mathcal{S}_f$ as shown by the following example.

Example 4.2 *Consider the $\mathcal{S}_0$-constraint $\underline{1} \subseteq X$ when $\Sigma = \{a, f/1\}$. It is satisfiable with the infinite solution $X/\{a, f(a), f(f(a)), \ldots\}$. Its translation (starting from its flat form $Y \doteq \underline{1} \wedge Y \subseteq X$) is*

$$Y = U \wedge Y \subseteq X \wedge X \subseteq U \wedge Y \subseteq U \wedge U \neq \emptyset$$

which is satisfiable in CLP($\mathcal{SET}$) (CLP($\mathcal{SET}$)$^+$ is not needed). The solution returned is: $U = \{A \mid B\}, Y = U, X = U$ where A, B are two new variables.

The following results hold for this class of Set constraints.

Proposition 4.3 *Let $\Sigma = \{a\}$. Checking satisfiability of a $\mathcal{S}_0$-constraint over T_Σ is NP-hard.*

Proof. Any instance of 3-SAT, e.g. $(X_1 \vee \neg X_2 \vee X_3) \wedge (\neg X_1 \vee X_2 \vee \neg X_3)$ can be transformed into the $\mathcal{S}_0$-constraint:

$$\underline{1} \subseteq (X_1 \cup \mathsf{C}(X_2) \cup X_3) \cap (\mathsf{C}(X_1) \cup X_2 \cup \mathsf{C}(X_3))$$

It is immediate to prove that any solution of the former is a solution of the latter and *vice versa*, provided `false` is interpreted with $\underline{0}$ and `true` is interpreted with $\{a\}$. This encoding can be easily generalized to any instance of 3-SAT. $\square$

Theorem 4.4 *Let C be a $\mathcal{S}_0$-constraint. C is satisfiable in $\wp(T_\Sigma)$ if and only if $\phi(C)$ is satisfiable in $\mathcal{T} = \langle T_{\Sigma'}/\equiv, (\cdot)^{\mathcal{T}}\rangle$, where $\Sigma' = \Sigma \cup \{\emptyset, \{\cdot \mid \cdot\}\}$.*

Proof. If Σ contains only constant symbols, the result is trivial. The two solutions are the same.

Assume Σ contains at least a function symbol (a constant symbol must be present in Σ by definition) and let C be a $\mathcal{S}_0$-constraint. This implies that $\phi(C)$ is a CLP($\mathcal{SET}$)-constraint not involving RUQs and based only on the (set-theoretic) language $\emptyset, \{\cdot \mid \cdot\}, =, \in, \cup_3, \|$. Thus, $\phi(C)$ belongs to a class of set-theoretic formulae "reflecting on the finite". In other words, $\phi(C)$ is satisfiable if and only if it is satisfiable over the universe of hereditarily finite and well-founded sets [8].

It is easy to see that any solution of C involving finite (infinite) sets of finite terms has a corresponding solution for $\phi(C)$ involving finite (infinite) sets (in the case of infinite, consider a set-theoretic universe, extending $T_{\Sigma'}/\equiv$). From the property above, there is a solution for $\phi(C)$ over $\mathcal{T}$.

It remains to prove that if $\phi(C)$ is satisfiable over $\mathcal{T}$ then C is satisfiable over $\wp(T_\Sigma)$. Let σ be a solution of $\phi(C)$ over $\mathcal{T}$, and let $\eta : T_{\Sigma'}/\equiv \longrightarrow T_\Sigma$, be an arbitrarily bijective function. Using η we build the function ρ from $U\sigma$ (viewed as a nonempty set of terms) to $\wp(T_\Sigma)$: fix arbitrarily $r \in U\sigma$ and define

$$\begin{cases} \rho(t) & = & \{\eta(t)\} & \text{if } r \neq t \\ \rho(t) & = & \eta(r) \cup (T_\Sigma \setminus \{\eta(s) : s \in U\sigma\}) & \text{otherwise} \end{cases}$$

By case analysis, it is easy to prove that the valuation computed on each variable as: $X/\bigcup_{s \in \sigma(X)} \rho(s)$ is a solution of C over T_Σ. $\square$

Corollary 4.5 *Let $\Sigma = \{a\}$. Checking satisfiability of a $\mathcal{S}_0$-constraint over T_Σ is NP-complete.*

Proof. Immediate from the two above results and [31]. □

In practical applications, in particular in set-based program analysis, the availability of function symbol strongly enhances the expressive power of the language. Therefore in the next sections we will introduce new sub-classes of Set constraints which overcome these limitations of $\mathcal{S}_0$-constraints. We will show how to decide Set constraints in these classes—whose solutions are possibly infinite sets—using the language CLP($\mathcal{SET}$) which computes over finite sets only.

5 The Class $\mathcal{S}_1$

In this section we identify a class of Set constraints such that when a constraint of this class is translated to a CLP$(\mathcal{SET})^+$ constraint, RUQs can occur, but their presence does not affect the feasibility of the satisfiability test with a CLP($\mathcal{SET}$) based technique. Such a class ($\mathcal{S}_1$) will help in solving the analogous problem for the class $\mathcal{S}_2$ presented in the next section that is akin to a well-known class of Set constraints.

Definition 5.1 *We call $\mathcal{S}_1$ the class of Set constraints generated by the following grammar:*

$$\begin{array}{rcl} \mathcal{S}_1 & ::= & C \mid C \wedge \mathcal{S}_1 \\ C & ::= & X \subseteq T \mid T \subseteq X \mid X_1 \subseteq X_2 \cup X_3 \\ T & ::= & \underline{0} \mid X \mid f(X_1, \ldots, X_n) \end{array}$$

where $X, X_1, \ldots, X_n$ are variables and f is a function or a constant symbol.

Intuitively, this class has been chosen so as to guarantee that there is a constructive way to define a (minimal) solution. Constraints of the form $X_1 \subseteq X_2 \cup X_3$ will introduce non-determinism forcing the non uniqueness of a least solution.

The following notion of sequence of evaluations will help us in finding a model (actually, a minimal model) of a $\mathcal{S}_1$-constraint.

5.1 Restricted $\mathcal{S}_1$

Let us begin by proving a property for the class of *restricted* $\mathcal{S}_1$- composed by conjunctions of constraints of the form:

$$X \subseteq e \qquad\qquad e \subseteq X$$

where X is a generic variable and e a set-expression of the form $\underline{0}$, Y (a generic variable), or $f(Y_1, \ldots, Y_n)$, Y_i all variables. We can also assume that $\underline{0} \subseteq X$ and $X \subseteq X$ do not appear in C (such constraints are always satisfiable and can be safely removed). We recall from [3] that a satisfiable constraint of this form admits always a minimum solution.

Let C be a restricted $\mathcal{S}_1$-constraint, we build a directed labeled graph $\mathcal{G}_C$ as follows:

nodes: add one node for each variable X and for each constant c occurring in C and one node identified by $\underline{0}$;

edges: The edges are of the form: $\langle m, \mathit{label}, n\rangle$, where *label* is a triple $\langle f, \mathsf{cost}, \mathsf{id}\rangle$ such that:

- $f \in \Sigma$,
- $\mathsf{cost} \in \{-1, 0, +1\}$, and
- $1 \leq \mathsf{id} \leq |C|$ is an identifier of an atom of C.

For each atomic constraint η of C, add edges according to the rules:

$$\begin{array}{rcl}
X \subseteq c & \mapsto & \langle X, \langle c, -1, \eta \rangle, c \rangle \\
X \subseteq \underline{0} & \mapsto & \langle X, \langle \underline{0}, -1, \eta \rangle, \underline{0} \rangle \\
X \subseteq f(Y_1, \ldots, Y_n) & \mapsto & \langle X, \langle f, -1, \eta \rangle, Y_1 \rangle, \ldots, \langle X, \langle f, -1, \eta \rangle, Y_n \rangle, \\
c \subseteq X & \mapsto & \langle c, \langle c, +1, \eta \rangle, X \rangle \\
f(Y_1, \ldots, Y_n) \subseteq X & \mapsto & \langle Y_1, \langle f, +1, \eta \rangle, X \rangle, \ldots, \langle Y_n, \langle f, +1, \eta \rangle, X \rangle \\
X \subseteq Y & \mapsto & \langle X, \langle \subseteq, 0, \eta \rangle, Y \rangle
\end{array}$$

Given a path $\gamma = \mu_0 \overset{f_1, c_1, \eta_1}{\to} \mu_1 \overset{f_2, c_2, \eta_2}{\to} \mu_2 \overset{f_3, c_3, \eta_3}{\to} \cdots \mu_{k-1} \overset{f_k, c_k, \eta_k}{\to} \mu_k$ in the graph, we say that the *cost* of the path γ is $cost(\gamma) = \sum_{j=1}^{k} c_j$. A *cycle* is a path such that $\mu_0 \equiv \mu_k$.

A variable X is $\mathcal{G}_C$*-nonempty* if there is:

- an edge $c \overset{c,+1,\eta}{\to} X$, or
- an edge $Y \overset{\subseteq,0,\eta}{\to} X$ and Y is $\mathcal{G}_C$-nonempty, or
- an edge $Y \overset{f,-1,\eta}{\to} X$ and Y is $\mathcal{G}_C$-nonempty, or
- a situation of the form depicted in Table 1, where $Y_1, \ldots, Y_n$ are all $\mathcal{G}_C$-nonempty and there are no other edges labeled $\overset{f,+1,\eta}{\to}$.

$$\begin{array}{ccc}
Y_1 & & \\
 & \overset{f,+1,\eta}{\searrow} & \\
\vdots & \vdots & X \\
 & \overset{f,+1,\eta}{\nearrow} & \\
Y_n & &
\end{array}$$

Table 1: A condition for $\mathcal{G}_C$-nonemptyness

Lemma 5.2 *Let C be a satisfiable restricted $\mathcal{S}_1$-constraint, X a variable which occurs in C, and θ the minimum solution of C. Then X is $\mathcal{G}_C$-nonempty if and only if $\theta[X] \neq \emptyset$.*

Proof. We first prove the $(\rightarrow)$ implication by induction on the definition of $\mathcal{G}_C$-nonempty.

- If there is an edge $c \overset{c,+1,\eta}{\to} X$, then in C there is the atom $c \subseteq X$, so $c \in \theta[X]$.
- If there is an edge $Y \overset{\subseteq,0,\eta}{\to} X$ (i.e., $Y \subseteq X$ is in C) and Y is $\mathcal{G}_C$-nonempty, then, by inductive hypothesis, $\theta[Y] \neq \emptyset$. Thus, $\theta[X] \neq \emptyset$.
- If there is an edge $Y \overset{f,-1,\eta}{\to} X$ (i.e., $Y \subseteq f(X_1, \ldots, X, \ldots, X_n)$ is in C), by inductive hypothesis $\theta[Y] \neq \emptyset$. Since C is satisfiable this means that all the terms in $\theta[Y]$ are of the form $f(t_1, \ldots, t, \ldots, t_n)$ (and, in particular, there is at least one term of this form in $\theta[Y]$). This means that $t \in \theta[X]$, hence $\theta[X] \neq \emptyset$.
- If we are in the situation depicted in Table 1, then in C there is an atom of the form $f(Y_{j_1}, \ldots, Y_{j_m}) \subseteq X$, $n \leq m$, Y_i not necessarily distinct, and such that $\{j_1, \ldots, j_m\} = \{1, \ldots, n\}$. By inductive hypothesis, for all $i \in \{1, \ldots, n\}$ it holds that $\theta[Y_i] \neq \emptyset$. Thus, $\theta[X] \neq \emptyset$.

We now prove the ($\leftarrow$) implication. To start, we can assume in C there are only unary function symbols.

If $t \in \theta[X]$, then there must be a chain of atoms of C of the form

$$e_1 \subseteq e'_1 \wedge e_2 \subseteq e'_2 \wedge \cdots \wedge e_n \subseteq e'_n$$

such that e_1 is a constant, the variable occurring in e'_i is the same that occurs in the term e_{i+1}, and the variable occurring in e'_n is X. From the first case of the definition we have that the variable in e'_1 is $\mathcal{G}_C$-nonempty. Hence, by induction on the length of the chain, using the other cases, we obtain that X is $\mathcal{G}_C$-nonempty.

If in C there are also function symbols of arity greater than one, the proof remains similar, with chains replaced by trees. $\square$

We now "close" C using the rewriting rules:

$$\begin{array}{rcl} \left.\begin{array}{c} f(X_1,\ldots,X_n) \subseteq X \wedge X \subseteq f(Y_1,\ldots,Y_n) \\ \text{for all } i=1,\ldots,n,\ X_i \text{ is } \mathcal{G}_C\text{-nonempty} \end{array}\right\} & \mapsto & X_1 \subseteq Y_1 \wedge \cdots \wedge X_n \subseteq Y_n \\ X \subseteq Y \wedge Y \subseteq e & \mapsto & X \subseteq e \\ e \subseteq X \wedge X \subseteq Y & \mapsto & e \subseteq Y \end{array}$$

Again, we can assume that $\underline{0} \subseteq X$ and $X \subseteq X$ do not appear in C.

In the following lemmata we will make use of the evaluation σ on the variables of a restricted $\mathcal{S}_1$-constraint C and defined as follows:

$$\left\{\begin{array}{rcl} \sigma_0[X] & = & \emptyset \\ \sigma_{i+1}[X] & = & \sigma_i[X] \cup \displaystyle\bigcup_{e \subseteq X \text{ in } C} \sigma_i[e] \cup \bigcup_{Y \subseteq f(\underbrace{\ldots,X}_{k},\ldots) \text{ in } C} f_k^{-1}(\sigma_i[Y]) \\ \sigma[X] & = & \displaystyle\bigcup_{i \in \omega} \sigma_i[X] \end{array}\right.$$

σ is extended to non-variable terms using the interpretation properties defined in Sect. 2.

Lemma 5.3 *If C is satisfiable, then σ satisfies C and it is the minimum solution of C.*

Proof. Immediate from the definition of σ and from the results in [3]. $\square$

Given a Set constraint C, with v and m we denote the number of distinct variables and the number of atoms in it, respectively. With $|C|$ we indicate the global number of (occurrences of) symbols in C. Of course, $|C| \leq v + m$. Let α be the maximum arity of function symbols of C.

The cardinality of $\sigma_i[X]$ ($\#(\sigma_i[X])$) can grow in a double exponential way with i. As a matter of fact, at each step of the computation of σ, cartesian products (of power $\leq \alpha$) must be explicitly computed. For instance, if C is $a \subseteq X \wedge f(X,X) \subseteq X$, then:

i	$\sigma_i[X]$	$\#(\sigma_i[X])$
0	$\emptyset$	0
1	$\{a\}$	1
2	$\{a, f(a,a)\}$	2
3	$\{a, f(a,a), f(a,f(a,a)), f(f(a,a),a), f(f(a,a),f(a,a))\}$	5
$\vdots$	$\vdots$	$\vdots$

More precisely,

Lemma 5.4 *Let C be a restricted $\mathcal{S}_1$-constraint. Then, for each variable X of C, and for $i \geq 1$:*

$$\#(\sigma_i[X]) = O(m^{\frac{\alpha^i-1}{\alpha-1}}) .$$

Proof. By induction on $i \geq 1$.

If $i = 1$, then $\#(\sigma_1[X]) \leq m$, since there are at most m atoms of the form $c \subseteq X$ with c constant.

Let us suppose that for each X in C $\#(\sigma_i[X]) \leq m^{\frac{\alpha^i-1}{\alpha-1}}$.

We have that:

$$\#(\sigma_{i+1}[X]) \leq m(\#(\sigma_i[X]))^\alpha ,$$

since each atom $f(Y_1, \ldots, Y_n) \subseteq X$ introduces $\#(\sigma_i[Y_1]) \times \cdots \times \#(\sigma_i[Y_n])$ elements in $\sigma_{i+1}[X]$, with $n \leq \alpha$, and at most m of these equations are in C. Thus, by inductive hypothesis,

$$\#(\sigma_{i+1}[X]) \leq m(m^{\frac{\alpha^i-1}{\alpha-1}})^\alpha = m^1 \cdot m^{\alpha \cdot \frac{\alpha^i-1}{\alpha-1}} = m^{\frac{\alpha-1+\alpha(\alpha^i-1)}{\alpha-1}} = m^{\frac{\alpha^{i+1}-1}{\alpha-1}}$$

□

Given a closed restricted $\mathcal{S}_1$-constraint C, we say that in the graph $\mathcal{G}_C$ there is a *critical situation* if there is a subgraph of $\mathcal{G}_C$ of the form:

$$\begin{array}{ccccc}
\mu_0^1 & & & & \\
 & \overset{f,+1,\eta_1}{\searrow} & & & \\
\vdots & \vdots & \mu_1 & \overset{g,-1,\eta_2}{\rightarrow} & \mu_2 \\
 & \overset{f,+1,\eta_1}{\nearrow} & & & \\
\mu_0^n & & & &
\end{array}$$

where $n \leq ar(f)$, $f \not\equiv g$, there are no other edges in $\mathcal{G}_C$ labeled by $\overset{f,+1,\eta_1}{\rightarrow}$, and for $i = 1, \ldots, n$ μ_0^i is $\mathcal{G}_C$-nonempty.

Lemma 5.5 *A closed restricted $\mathcal{S}_1$-constraint C is unsatisfiable if and only if there is a critical situation in $\mathcal{G}_C$.*

Proof. We first prove that if there is a critical situation on the graph $\mathcal{G}_C$, then the constraint is unsatisfiable.

Since $\mu_0^1, \ldots, \mu_0^n$ are all $\mathcal{G}_C$-nonempty, then μ_1 is $\mathcal{G}_C$-nonempty by definition. Moreover, any solution for the variable associated to node μ_1 must contain a term of the form $f(t_1, \ldots, t_n)$. Thus, the atom $\mu_1 \subseteq g(\ldots, \mu_2, \ldots)$ can not be fulfilled, since $f \not\equiv g$.

It remains to prove that if no critical situations occur in $\mathcal{G}_C$, then C is satisfiable. Suppose, by contradiction, that C is unsatisfiable. By Lemma 5.3, this implies that σ does not satisfy C. All the atoms of C that are not satisfied from σ have to be of the form $X \subseteq e$, with $e = \underline{0}$, c, or $f(Y_1, \ldots, Y_n)$ (it is immediate to observe that every atom of C of the form $e \subseteq X$ is always satisfied by σ).

The following property on closed $\mathcal{S}_1$-constraints holds: (1) if $f(t_1, \ldots, t_n) \in \sigma[X]$, then there are $X_1, \ldots, X_n$ such that $f(X_1, \ldots, X_n) \subseteq X$ is in C and for all $i \in \{1, \ldots, n\}$ $t_i \in \sigma[X_i]$.

We consider an atom $X \subseteq e$ not satisfied by σ. This means that there exists a term t such that $t \in \sigma[X]$, but $t \notin \sigma[e]$. The construction of σ implies that $t = f(\ldots)$ and $e = g(\ldots)$ with $f \not\equiv g$.

By the above property (1), it emerges that in the graph there must be a critical situation. □

We can state that:

Theorem 5.6 *The satisfiability of a restricted $\mathcal{S}_1$-constraint can be checked in time $k = O(|C|^3)$.*

Proof. First compute for each variable X of C, if X is $\mathcal{G}_C$-nonempty or not. This task can be performed in time $O(v^2 \times m) = O(|C|^3)$. For instance, define a procedure $\mathsf{expand}(\mu)$ that marks the node μ, and (a) deletes each edge outgoing from μ of the form $\langle \subseteq, 0 \rangle$, $\langle c, +1 \rangle$, or $\langle f, -1 \rangle$, and (b) recursively calls expand to all nodes reached from that edges. A procedure of this kind (since it destroys edges) can be globally applied m times. Then define a boolean function $\mathsf{markable}(\mu)$ that checks if a not yet marked node μ is made $\mathcal{G}_C$-nonempty by the condition of Table 1 (it takes time depending on m). Verifying if there is a markable node takes $O(v \cdot m)$. The main algorithm is as follows:

for $c \in N_\mathcal{G}$ do expand(c);
while there is $X \in N_\mathcal{G}$ s.t. markable(X) do expand(X);

Then compute a new constraint C' whose size is bounded by:

$$|C'| \leq v(v-1) + mv \leq |C|^2$$

Notice that the nodes of $\mathcal{G}_{C'}$ are the same as those of $\mathcal{G}_C$.

To discover the critical situations of the Lemma 5.5 we analyze the graph $\mathcal{G}_{C'}$. For each node, we need to check the intersections between the list of ingoing edges ($\leq m$) and of the outgoing edges. This can be done in time $O(v \cdot m^2) = O(|C|^3)$. □

The critical situations can also be discovered by computing the assignment σ. It is sufficient to build σ to mimic the paths from constants to variables that force variables to be $\mathcal{G}_{C'}$-nonempty.More in detail, if there is $X \subseteq g(\ldots, Y, \ldots)$ in C and there is $f(\ldots)$ in $\sigma_i[X]$, then we say σ_i creates a critical situation and we are sure that C is unsatisfiable.

Corollary 5.7 *A closed restricted $\mathcal{S}_1$-constraint C is unsatisfiable if and only if σ_k does not create critical situations, where k is the same as in Theorem 5.6.*

Proof. Immediate. □

The result can now be extended to (not necessarily closed) $\mathcal{S}_1$-constraints. Basically, closed constraints generate graphs with shorter paths between two variables. In any case, the number of variables, say v, remains unchanged in the two graphs. Thus, basically, the idea is that any critical situation discovered by σ computed on C' can be discovered working on Σ in a number of steps multiplied by v.

Corollary 5.8 *A restricted $\mathcal{S}_1$-constraint C is unsatisfiable if and only if σ_h does not create critical situations, where $h = k \cdot v$, namely $O(|C|^4)$.*

Proof. If C is a $\mathcal{S}_1$-constraint and C' is its closure, then every edge of $\mathcal{G}_{C'}$ can replace a path of $\mathcal{G}_C$ of length at most v. It follows that σ_i computed on C is included into σ_j computed on C', where $i = v * j$. Thus, since we only need to check critical situations, if we compute something "more" in σ_j computed on C', we will discover this fact anyway. From Corollary 5.7 we obtain the thesis. □

Corollary 5.9 *The satisfiability check of a restricted $\mathcal{S}_1$-constraint C using the σ can be performed in time $O(m^{\frac{\alpha^{|C|^4}-1}{\alpha-1}})$.*

Proof. From Lemma 5.4 and Corollary 5.8. $\square$

Remark 5.10 *Observe that if we are interested in checking the satisfiability of restricted $\mathcal{S}_1$-constraints, a polynomial procedure is sufficient (Theorem 5.6). If, otherwise, we are interested in finding a solution (or a part of it), (double) exponential time algorithms are needed (Corollary 5.9).*

5.2 Unrestricted $\mathcal{S}_1$-constraints

We need now to extend the result to the case of S_1-constraints containing also literals of the form $X \subseteq Y \cup Z$.

Definition 5.11 *Given a $\mathcal{S}_1$-constraint C we can non-deterministically define an evaluation σ as follows:*

$$\left\{\begin{array}{rcl} \sigma_0[X] & = & \emptyset \\ \sigma_{i+1}[X] & = & \sigma_i[X] \cup \bigcup_{e \subseteq X \text{ in } C} \sigma_i[e] \cup \bigcup_{Y \subseteq f(\underbrace{\ldots, X, \ldots}_{k}) \text{ in } C} f_k^{-1}(\sigma_i[Y]) \cup \bigcup_{Y \subseteq X \cup Z} \eta(\sigma_i[Y]) \\ & & \\ \sigma[X] & = & \bigcup_{i \,\in\, \omega} \sigma_i[X] \end{array}\right.$$

where $\eta(S)$ returns, non-deterministically, a subset of the set S such that $\sigma_i[Y] \subseteq \sigma_{i+1}[X] \cup \sigma_{i+1}[Z]$ is fulfilled.

The next results allow us to find a precise upper bound to the cardinality of sets that need to be computed in order to decide the (finite or infinite) satisfiability of the constraint:

Corollary 5.12 *An $\mathcal{S}_1$-constraint C is unsatisfiable if and only if all the possible non-deterministic σ_h generate critical situations, where $h = O(|C|^4)$.*

Proof. It derives from Corollary 5.8 and from the fact that any minimal (not necessarily, minimum, since the property of having a least solution does not hold for $\mathcal{S}_1$-constraints) solution is described by one of the σ_h. $\square$

Corollary 5.13 *Let C be a $\mathcal{S}_1$-constraint. Then, for each variable X of C, and for $i \geq 1$ the cardinality of $\sigma_i[X]$ is not greater than $m^{\frac{\alpha^i-1}{\alpha-1}}$.*

Proof. It is an immediate consequence of the Lemma 5.4 since the occurrences of $\cup$ do not force the sets to be wider. $\square$

Now we can move to CLP($\mathcal{SET}$). To do this, we can exploit the strong connection between the way σ is computed and the way CLP($\mathcal{SET}$) builds (finite) sets when trying to find a solution to a given constraint. Sets are built element by element by solving the RUQs formulae possibly occurring in the CLP$(\mathcal{SET})^+$-constraint generated by ϕ. At each *computation step* a new element is added to each such sets, and the whole constraint is checked to test its satisfiability.

Therefore, we can state the following:

Theorem 5.14 *Let C be a $\mathcal{S}_1$-constraint, C' be the CLP($\mathcal{SET}$)-constraint resulting from the translation of C using ϕ, and p be an integer constant whose value depends on the (syntactic) form of C. Then, if the non-deterministic computation of the CLP($\mathcal{SET}$) constraint solver applied to C':*

- *terminates* with success *in less than p steps in at least one of its branches, then C is satisfiable in $\wp_{fin}(T_\Sigma)$ (hence in $\wp(T_\Sigma)$);*
- *terminates* with failure *in less than p steps in all its branches, then C is unsatisfiable in $\wp(T_\Sigma)$ (hence in $\wp_{fin}(T_\Sigma)$);*
- *does not terminate in less than p steps, then C is satisfiable in $\wp(T_\Sigma)$ (but not necessarily in $\wp_{fin}(T_\Sigma)$).*

Proof. The theorem follows from the analogy between the computation of the σs and a CLP($\mathcal{SET}$) computation and Corollaries 5.12 and 5.13, with $p = m^{\frac{\alpha^h - 1}{\alpha - 1}}$. □

Note that, the fact that the CLP($\mathcal{SET}$) constraint solver always explicitly constructs a solution for the constraint it has to check means that, in the case the solution involves an infinite set, what is obtained at the p-step is just a partial representation of this infinite set, which can be (finitely) represented in CLP($\mathcal{SET}$) as a partially specified set, $\{t_1, \ldots, t_h \mid X\}$, X variable.

6 The Class $\mathcal{S}_2$

In this section we show how to move to a larger class of Set constraints—called $\mathcal{S}_2$—which strictly contains $\mathcal{S}_1$ and nevertheless can be decided using CLP($\mathcal{SET}$).

Definition 6.1 *$\mathcal{S}_2$ is the class of Set constraints generated by the following grammar:*

$$\begin{array}{lcl} \mathcal{S}_2 & ::= & C \mid C \wedge \mathcal{S}_2 \\ C & ::= & L \subseteq R \\ L & ::= & \underline{0} \mid X \mid f(L, \ldots, L) \mid L \cup L \\ R & ::= & \underline{0} \mid X \mid f(R, \ldots, R) \mid g_1^{-1}(R) \mid R \cup R \mid R \cap R \end{array}$$

where X denotes a variable, f is any function or a constant symbol, while a function symbol g can be used in projection constructors (g_1^{-1}) only if $ar(g) = 1$.

Intersection in L and the full power of projections make difficult to develop a constructive approach based on the CLP($\mathcal{SET}$)$^+$ constraint solver.

Every $\mathcal{S}_2$-constraint can be translated automatically into a $\mathcal{S}_1$-constraint by a procedure, called *flat*, which introduce new variables to obtain the necessary flattening.

The translation rules for the procedure *flat* are shown in Figure 2:

It can be proved that *flat*(C) always terminates. Moreover, it holds that:

Proposition 6.2 *C is satisfiable if and only if flat(C) is satisfiable.*

Proof. The proof is a consequence of the correctness and completeness of each transformation rule with respect to models in $\wp(T_\Sigma)$. □

Finally, by Proposition 6.2 and Theorem 5.14, we obtain an algorithm for checking the satisfiability of any constraint in the wider class $\mathcal{S}_2$.

$\left.\begin{array}{r} s \subseteq t \\ nonvar(s) \text{ and } nonvar(t) \end{array}\right\}$	$\mapsto$	$s \subseteq X \wedge X \subseteq t$
$\left.\begin{array}{r} f(s_1,\ldots,s_n) \subseteq X \\ \text{there is } i = 1,\ldots,n \text{ s.t. } nonvar(s_i) \end{array}\right\}$	$\mapsto$	$\bigwedge_{i=1}^n s_i \subseteq X_i \wedge f(X_1,\ldots,X_n) \subseteq X$
$s_1 \cup s_2 \subseteq X$	$\mapsto$	$s_1 \subseteq X \wedge s_2 \subseteq X$
$\left.\begin{array}{r} X \subseteq f(t_1,\ldots,t_n) \\ \text{there is } i = 1,\ldots,n \text{ s.t. } nonvar(t_i) \end{array}\right\}$	$\mapsto$	$X \subseteq f(X_1,\ldots,X_n) \wedge \bigwedge_{i=1}^n X_i \subseteq t_i$
$\left.\begin{array}{r} X \subseteq t_1 \cup t_2 \\ nonvar(t_1) \text{ or } nonvar(t_2) \end{array}\right\}$	$\mapsto$	$X \subseteq X_1 \cup X_2 \wedge X_1 \subseteq t_1 \wedge X_2 \subseteq t_2$
$X \subseteq t_1 \cap t_2$	$\mapsto$	$X \subseteq t_1 \wedge X \subseteq t_2$
$X \subseteq g_1^{-1}(t)$	$\mapsto$	$g(X) \subseteq Y \wedge Y \subseteq t$

Figure 2: Rules for the procedure *flat*

7 Set Constraint classes

In Section 2.1 we have quoted the general definition of Set constraints. Set constraints are decidable in this general setting ([9]). However, due to complexity bounds, for an effective applicability of Set-based program analysis, restricted subclasses of the set constraints have been singled out. In this section we recall some of these subclasses and we relate them with the classes $\mathcal{S}_1$ and $\mathcal{S}_2$ analyzed in this paper.

Definition 7.1 *The class $\mathcal{D}ef$ of* definite *Set constraints is the class of Set constraints generated by the grammar:*

$$\begin{array}{rcl} \mathcal{D}ef & ::= & C \mid C \wedge \mathcal{D}ef \\ C & ::= & L \subseteq R \\ L & ::= & \underline{0} \mid X \mid f(L,\ldots,L) \mid f_k^{-1}(L) \mid L \cup L \mid L \cap L \\ R & ::= & \underline{0} \mid X \mid f(R,\ldots,R) \mid g_1^{-1}(R) \mid R \cap R \end{array}$$

where X denotes a variable, f is any function or a constant symbol, with $ar(f) \geq k$, and g is a unary function symbol.

The restrictions on the expressions on the l.h.s. for $\mathcal{D}ef$ are stronger than those required for the class $\mathcal{S}_1$. As a matter of fact, the expression $R \cup R$, R a variable, is allowed as l.h.s. of a constraint in $\mathcal{S}_1$ but not in $\mathcal{D}ef$. On the other hand, $f_k^{-1}(L)$ and $L \cap L$ are not allowed in $\mathcal{S}_1$ as l.h.s., while their use is accepted in $\mathcal{D}ef$. However, notice that when the signature Σ is finite (this is often true in real applications) we can safely enlarge the class $\mathcal{S}_1$ to allow $f_k^{-1}(L)$ on the l.h.s., since a constraint of this form can be rewritten using:

$$f_k^{-1}(s) \subseteq t \mapsto s \subseteq f(X_1,\ldots,X_{k-1},t,\ldots,X_n) \cup \bigcup_{g \in \Sigma, g \not\equiv f} g(Y_1,\ldots,Y_m).$$

The class $\mathcal{D}ef$ of definite set constraints has been presented in [24] and used for the set-based analysis of logic programs in [25].

Definition 7.2 *The class $\mathcal{P}os$ of* positive *Set constraints is the class of Set constraints generated by the grammar:*

$$\begin{array}{rcl} \mathcal{P}os & ::= & C \mid C \wedge \mathcal{P}os \\ C & ::= & T \subseteq T \\ T & ::= & \underline{0} \mid X \mid f(T,\ldots,T) \mid T \cup T \mid T \cap T \mid \complement(T) \end{array}$$

where X denotes a variable and f is any function or a constant symbol.

This class is a superclass of the class $\mathcal{S}_2$ presented in this paper (g_1^{-1} when $ar(g) = 1$ can be easily removed). More precisely, $\mathcal{P}os$ enlarges $\mathcal{S}_2$ with the possibility of writing the complement on the l.h.s. ($\complement(R)$) and that of writing the complement and the intersection on the r.h.s. ($\complement(L), L \cap L$).

This class has been introduced in [3], while its relations with the monadic first-order formulae has been pointed out in [5].

Definition 7.3 *The class $\mathcal{I}nt$ of Set constraints with intersection is the class of Set constraints generated by the grammar:*

$$\begin{array}{rcl}
\mathcal{I}nt & ::= & C \mid C \wedge \mathcal{I}nt \\
C & ::= & T \subseteq T \mid T \not\subseteq T \\
T & ::= & \underline{0} \mid X \mid f(T, \dots, T) \mid T \cap T
\end{array}$$

where X denotes a variable, f is any function or a constant symbol.

As mentioned before, we can skip negative set constraint ($t \not\subseteq s$) since they can previously be mapped into CLP($\mathcal{SET}$) constraints of the form $X \in t \wedge X \notin s$ that do not introduce problems in the satisfiability checking algorithms (e.g., non-termination). Thus, to obtain from the class $\mathcal{S}_1$ (and hence also from $\mathcal{S}_2$) a superclass of $\mathcal{I}nt$ we have to add in the grammar the production $L \cap L$.

On the other side, to obtain a superclass of $\mathcal{S}_2$ from the class $\mathcal{I}nt$ it is necessary to allow the production $T \cup T$.

The class of set constraints with intersection has been defined in [10], together with the development of a satisfiability test.

Definition 7.4 *The class $\mathcal{C}o\text{-}def$ of* Co-definite *Set constraints is the class of Set constraints generated by the following grammar:*

$$\begin{array}{rcl}
\mathcal{C}o\text{-}def & ::= & C \mid C \wedge \mathcal{C}o\text{-}def \\
C & ::= & L \subseteq R \\
L & ::= & \underline{0} \mid X \mid g(L) \mid L \cup L \\
R & ::= & \underline{0} \mid X \mid f(R, \dots, R) \mid f_k^{-1}(R) \mid R \cap R \mid R \cup R
\end{array}$$

where X denotes a variable, f is any function or a constant symbol, with $ar(f) \geq k$, and g is a unary function symbol.

Differences between the class $\mathcal{S}_2$ and the co-definite Set constraints are that, on the one hand, $\mathcal{S}_2$ enlarges the class of co-definite Set constraints in that it does not put any restriction on the function symbols in L, and, on the other hand, in $\mathcal{S}_2$ projections are allowed only if applied to unary function symbols: a condition not required by co-definite Set constraints. This restriction ensures that the equivalence $t \subseteq f_1^{-1}(s)$ if and only if $f(t) \subseteq s$ holds.

This class has been introduced and used for program analysis in [11, 12, 15].

Definition 7.5 *The class $\mathcal{I}nes$ of Ines constraints is the class of Set constraints generated by the following grammar:*

$$\begin{array}{rcl}
\mathcal{I}nes & ::= & C \mid C \wedge \mathcal{I}nes \\
C & ::= & T \subseteq T \\
T & ::= & \underline{0} \mid X \mid f(T, \dots, T)
\end{array}$$

where X denotes a variable, f is any function or a constant symbol.

Clearly, $\mathcal{S}_2$ is a superclass of the class of $\mathcal{I}nes$-constraints, which do not allow any set operator. $\mathcal{I}nes$ constraints have been defined in [30], where a satisfiability procedure of complexity $O(n^3)$ for this class is presented, and the applications to program analysis are described.

We have pointed out the relationships between some classes of set constraints and our class $\mathcal{S}_2$. To conclude this section we can notice that constraints in many of the subclasses presented here can contain constraints of $\mathcal{S}_0$. Thanks to the simplicity of this class, a preliminary test over the part of the constraints that are also $\mathcal{S}_0$-constraints can be performed to cut the search tree.

8 Conclusions and future work

We have shown that the whole class of Set constraints can be translated to CLP($\mathcal{SET}$)$^+$-constraints for which we can use the CLP($\mathcal{SET}$)-interpreter to obtain a partial test for Set constraint satifiability: if the CLP($\mathcal{SET}$)-interpreter terminates with success (resp., with failure) we can conclude that the original Set constraint is satisfiable (resp., unsatisfiable), also getting, in the case of success, the possible solutions for it. There are cases, however, in which CLP($\mathcal{SET}$) may fail to test satisfiability of CLP($\mathcal{SET}$)$^+$-constraints due to non-termination. Therefore we have identified three sub-classes of Set constraints for which CLP($\mathcal{SET}$) can be used to obtain a complete satifiability test. For constraints of these classes we can either exclude a-priori non-termination (class $\mathcal{S}_0$) or we can fix a bound to the number of steps of the corresponding CLP($\mathcal{SET}$) computation which allows us to check anyhow satisfiability of the Set constraint using CLP($\mathcal{SET}$) (classes $\mathcal{S}_1$ and $\mathcal{S}_2$).

As future work, one could try to characterize larger classes of Set constraints whose satisfiability can be always checked by using CLP($\mathcal{SET}$), by adapting to these classes the techniques already used for $\mathcal{S}_1$ and $\mathcal{S}_2$. Also, extending CLP($\mathcal{SET}$), e.g., by moving to the domain of rational trees and hypersets [4], could help us to deal with larger classes of Set constraints.

On the other hand, a possible future line of work, could be to extend the notion of Set constraints to a wider class of decidable set-based constraints, possibly by making use of Computable Set Theory results, and by exploiting other already available, as well as future, facilities of CLP($\mathcal{SET}$).

Acknowledgements

The authors wish to thank the anonymous referees of preliminary versions of this paper that helped us to improve the presentation of the work.

A. Dovier and C. Piazza are partially supported by MURST project: *Tecniche formali per la specifica, l'analisi, la verifica, la sintesi e la trasformazione di sistemi software.*

References

[1] A. Aiken. Set constraints: Results, Applications, and Future Directions. In A. Borning editor, *Principles and Practice of Constraint Programming, Second Int'l Workshop, PPCP'94*, LNCS Vol. 874, pp. 326-335, Springer-Verlag, 1994.

[2] A. Aiken, D. Kozen, and E. Wimmers. Decidability of systems of set constraints with negative constraints. Technical report, Computer Science Department, Cornell University, June 1993.

[3] A. Aiken and E. Wimmers. Solving systems of set constraints. In *Proc. 7th Symp. Logic in Computer Science*, pages 329–340. IEEE, 1992.

[4] D. Aliffi, A. Dovier, and G. Rossi. From Set to Hyperset Unification. *Journal of Functional and Logic Programming* (MIT Press), to appear.

[5] L. Bachmair, H. Ganzinger, and U. Waldmann. Set constraints are the monadic class. In *Proc. 8th Symp. Logic in Computer Science*, pages 75–83. IEEE, 1993.

[6] P. Bruscoli, A. Dovier, E. Pontelli, and G. Rossi. Compiling Intensional Sets in CLP. In Pascal Van Entenryck, editor, *Proc. 11th Int'l Conf. on Logic Programming*, pages 647–661. The MIT Press, Cambridge, Mass., June 1994.

[7] F. Bueno, M. Codish, and V. Lagoon. Sharing Analysis Using Set Logic Programs In Lee Naish ed., *Proc. 14th International Conference on Logic Programming* MIT Press, 1997, pag. 417.

[8] D. Cantone, A. Ferro, and E. G. Omodeo. *Computable Set Theory, Vol. 1.* Int'l Series of Monographs on Computer Science. Clarendon Press, Oxford, 1989.

[9] W. Charatonik and L. Pacholski. Set constraints with projections are in NEXPTIME. In *Proc. 35th Symp. Foundations of Computer Science*, pages 642-653, IEEE, 1994.

[10] W. Charatonik and A. Podelski. Set constraints with intersection. In *Proc. 12th Symp. Logic in Computer Science*, pages 362–372, IEEE, 1997.

[11] W. Charatonik and A. Podelski. Set constraints for greatest models. Technical report mpi-i-97-2-004, Max-Planck-Institut für Informatik, April 1997.

[12] A. Podelski, W. Charatonik, and M. Müller II. Set-Based Failure Analysis for Logic Programs and Concurrent Constraint Programs. In S. Doaitse Swierstra ed., *Programming Languages and Systems, 8th European Symposium on Programming*, 177-192, 1998.

[13] M. Codish and V. Lagoon. Type Dependencies for Logic Programs Using ACI-Unification. Technical report, Department of Mathematics and Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel, 1996.

[14] P. Cousot and R. Cousot. Formal language, grammar and set-constraint-based program analysis by abstract interpretation. In *Conference on Functional Programming Languages and Computer Architecture*, pages 170–181. ACM, 1995.

[15] G. Delzanno and A. Podelski. Model Checking in CLP. In R. Cleaveland ed., *Tools and Algorithms for Construction and Analysis of Systems, 5th International Conference,* 223-239, 1999.

[16] A. Dovier, E. G. Omodeo, E. Pontelli, and G. Rossi. {log}: A Language for Programming in Logic with Finite Sets. *J. of Logic Programming*, 28(1):1–44, 1996.

[17] A. Dovier, C. Piazza, E. Pontelli, and G. Rossi. On the Representation and Management of Finite Sets in CLP-languages. In J. Jaffar, editor, *Proc. 1998 Joint Int'l Conference and Symposium on Logic Programming*, pages 40–54. The MIT Press, Cambridge, Mass., June 1998.

[18] A. Dovier, A. Policriti, and G. Rossi. A uniform axiomatic view of lists, multisets, and sets, and the relevant unification algorithms *Fundamenta Informaticae*, 36(2/3):201-234, 1998.

[19] A. Dovier and G. Rossi. Embedding Extensional Finite Sets in CLP. In D. Miller, editor, *Proc. Int'l Logic Programming Symposium, ILPS'93*, pages 540–556. The MIT Press, Cambridge, Mass., October 1993.

[20] J. Foster. CLP(SC): Implementations and Efficiency Considerations. In *Workshop on Set Constraints at CP'96*, Cambridge, MA, August 1996.

[21] C. Gervet. Interval Propagation to Reason about Sets: Definition and Implementation of a Practical Language. *Constraints* 1(3):191–244, 1997.

[22] R. Gilleron, S. Tison, and M. Tommasi. Solving system of set constraints with negated subset relationships. In *Proc. 34th Symp. Foundations of Computer Science*, pages 372–380. IEEE, 1993.

[23] R. Gilleron, S. Tison, and M. Tommasi. Solving systems of set constraints using tree automata. In *Proc. Symp. Theoretical Aspects of Computer Science*, volume 665 of *L. Notes in Computer Science*, pages 505–514. Springer-Verlag, Berlin, 1993.

[24] N. Heintze and J. Jaffar. A decision procedure for a class of set constraints. In *Proc. 5th Symp. Logic in Computer Science*, pages 42–51. IEEE, 1990.

[25] N. Heintze and J. Jaffar. A finite presentation theorem for approximating logic programs. In *Proc. 17th Symp. Principles of Programming Languages*, pages 197–209. ACM, 1990.

[26] N. Heintze and J. Jaffar. Set Constraints and Set-Based Analysis. In A. Borning editor, *Principles and Practice of Constraint Programming, Second Int'l Workshop, PPCP'94*, LNCS Vol. 874, pp.281-298, Springer-Verlag, 1994.

[27] J. Jaffar and M. J. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 19–20:503–581, 1994.

[28] D. Kozen. Set Constraints and Logic Programming. *Information and Computation* 142(1):2–25, 1998.

[29] B. Legeard and E. Legros. Short overview of the CLPS system. In J. Maluszynsky and M. Wirsing, eds., *Proc. 3rd Int'l PLILP*, vol. 528 of *LNCS*, pp. 431–433. Springer-Verlag, Berlin, 1991.

[30] M. Müller, J. Niehren, and A. Podelski. Inclusion constraints over non-empty sets of trees. In M. Dauchet, editor, *Proc. 9th TAPSOFT*, volume 1214 of *Lecture Notes in Computer Science*, pages 345–356. Springer-Verlag, Berlin, 1997.

[31] E. G. Omodeo and A. Policriti. Solvable set/hyperset contexts: I. Some decision procedures for the pure, finite case. *Communication on Pure and Applied Mathematics.* 9–10:1123–1155, 1995.

[32] J. T. Schwartz, R. B. K. Dewar, E. Dubinsky, and E. Schonberg. *Programming with sets, an introduction to SETL.* Springer-Verlag, Berlin, 1986.