Narrowing the gap between Set-Constraints and CLP($\mathcal{SET}$)-Constraints

Agostino Dovier Carla Piazza Gianfranco Rossi

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 CLP($\mathcal{SET}$)-constraint; moreover, we show how the satisfiability check for some classes of set-constraints can be performed by the CLP($\mathcal{SET}$) constraint solver.

Keywords: Set Constraints, Constraint Logic Programming with Sets.

1 Introduction

Generally speaking, *set-based constraints* can be defined as formulae of a first-order language $\mathcal{L}$ whose literals make use of classical set-theoretic symbols, such as $\in, \subseteq, \cup \ldots$. The (set) constraint satisfiability problem amounts to finding an algorithm (a *set constraint solver*) which is able to decide the satisfiability of a constraint c in a privileged interpretation $\mathcal{D}$ of $\mathcal{L}$, that is, in a fixed constraint domain, which involves (possibly infinite) sets.

A well-known class of set-based constraints is the class of so called *set-constraints*. The set-constraint satisfiability problem was set first in the fields of program analysis and type inference, where, intuitively speaking, set-constraints are used to capture some properties of a programs ([16, 17], and [19] for an algebraic description). Various algorithms have been proposed for the satisfiability checking of sub-classes of the general set-constraint satisfiability problem (see, e.g., [3, 2, 14, 15]). Finally, in [7] the general problem has been proved to be decidable. However, in [7] and in other works (e.g., [1, 24]), the non-deterministic exponential complexity of the problem has been pointed out. Recently, a number of proposals have been put forward aimed at identifying useful polynomial sub-classes of the general problem (e.g., [22]).

Dovier: Dip. Scientifico-Tecnologico, Univ. di Verona. Strada Le Grazie 3, 37134 Verona (I). dovier@sci.univr.it

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

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

The work is partially supported by MURST project: *Tecniche formali per la specifica, l'analisi, la verifica, la sintesi e la trasformazione di sistemi software* and by CNR Grant 97.02426.CT12.

Another interesting approach to set-based constraints is that of constraint logic programming (CLP) [18]. A $CLP(\mathcal{X})$ scheme is a parametric CLP language whose parameter $\mathcal{X}$ can be instantiated to a generic domain of contraints and computation. Several proposals have been put forward for set-based instantiations of the parameter $\mathcal{X}$, to obtain *CLP languages with sets*. In particular, the languages CLPS [21] and CONJUNTO [13] are two of these proposals, aimed at developing a practical tool for programming with sets. Another—more general and theoretically sound—proposal following this approach is that of the language {log} and its CLP counter-part *CLP($\mathcal{SET}$)* (see [9, 12, 10]). CLP($\mathcal{SET}$) is a general-purpose CLP language dealing with sets that can be *nested* (sets of sets), *hybrid* (sets of terms, sets as arguments of standard terms, and so on) and possibly defined in an intensional way ([5, 11]). Observe that once a *set* instance of CLP has been defined, it is possible to combine it with other CLP languages (e.g., $CLP(\mathcal{R})$) to get a richer and powerful language [4].

It is worth mentioning that most of the results obtained for {log}, and for set-based languages in general, are strongly related to the results obtained in the *Computable Set Theory* area [6]. C.S.T. was mainly developed at the NYU 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 [23]. The general problem was that of identifying computable classes of formulae of suitable sub-theories of the general ZF 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.

Aim of this paper is the analysis and comparison—as concerns the language expressive power—of the two apparently distinct classes of set-based constraints, the class of *set-constraints* and the class of *CLP($\mathcal{SET}$)-constraints*.

We compare the two proposals by trying to reduce the constraint satisfiability problem for one class of constraints to the same problem for the other class (some work in this direction is also presented in [20], where, in particular, the possibility of using the language CLP($\mathcal{SET}$) is pointed out). The main contribution of this paper is to show that, in spite of their different origin and aims, there is a large class of constraint formulae for which both proposals provide suitable procedures for testing constraint satisfiability with respect to a given privileged interpretation. This comparison can serve to narrow the gap between two research areas that have proceeded quite separately so far. Such a result is obtained by:

- using a uniform approach and notation in presenting the two proposals;
- augmenting the CLP($\mathcal{SET}$) language of [12] by allowing it to deal with also the union and disjoint constraints as recently proposed in [10];
- developing a suitable rewriting procedure of set-constraints to CLP($\mathcal{SET}$)-constraints;
- characterizing sub-classes of set-constraints (which include, for instance, INES-constraints [22]) for which the rewriting preserves satisfiability.

The paper is organized as follows. In Sect. 2 and 3 we review the notions of set-constraint and the basic features of the language CLP($\mathcal{SET}$). In Sect. 4 we define the technique to be used for comparing the two languages, and in Sect. 5 we show how to translate a (general) set-constraint to a CLP($\mathcal{SET}$) program. Finally, in Sect. 6 we show how it is possible to use CLP($\mathcal{SET}$) to test satisfiability (even in domains with infinite sets) for a non-trivial sub-class of set-constraints.

2 Set-Constraints

In order to establish a common basis for comparing the two proposals, we present both of them as particular instances of the general constraint satisfiability problem for first-order languages. As concerns notation and terminology, we will keep to the syntactic conventions usually adopted in logic programming. In particular, capital letters X, Y, Z, etc. are used to represent variables; f, g, etc. stand for functional symbols. Moreover, if t is a term, then $vars(t)$ denotes the set of (free) variables occurring in t. With $\bar{t}$ we denote a list of terms $t_1, \ldots, t_n$.

Definition 1 *Given a signature Σ of functional and constant symbols (each with a fixed arity), consider the new signature:*

$$\Sigma' = \Sigma \cup \{\underline{0}, \underline{1}, \cup, \cap, \complement\} \cup \{f_k^{-1} \,:\, f \in \Sigma \quad and \quad 1 \leq k \leq ar(f)\},$$

the set of predicate symbols $\Pi = \{\subseteq\}$, and a denumerable set of variables $\mathcal{V}$. A set-constraint language *is a language $\mathcal{L} = \langle \Sigma', \Pi, \mathcal{V} \rangle$. A* set-constraint *is a finite conjunction of literals of $\mathcal{L}$.*

According to [16], we define the domain in which satisfiability of set-constraints has to be tested as follows.

Definition 2 *Given a set-constraint language $\mathcal{L}$, the privileged interpretation $\mathcal{P}$ for $\mathcal{L}$ is $\mathcal{P} = \langle \wp(T_\Sigma), (\cdot)^{\mathcal{P}} \rangle$, where T_Σ is the set of ground first-order terms built form the signature Σ (assume that there is at least one constant symbol in Σ) and $\wp(T_\Sigma)$ is its powerset. Moreover, the interpretation function $(\cdot)^{\mathcal{P}}$ for the symbols of Σ' is defined as follows:*[1]

$$\begin{array}{rcll}
\underline{0}^{\mathcal{P}} & = & \emptyset & \qquad \underline{1}^{\mathcal{P}} \;=\; T_\Sigma \\
(f(a_1, \ldots, a_n))^{\mathcal{P}} & = & \{f(t_1, \ldots, t_n) \,:\, \forall\, i \leq n, n \geq 0, t_i \in a_i^{\mathcal{P}}\} & \\
(f_k^{-1}(a))^{\mathcal{P}} & = & \{t_k \,:\, k \leq n, n \geq 1, f(t_1, \ldots, t_n) \in a^{\mathcal{P}}\} & \\
(a \cup b)^{\mathcal{P}} & = & a^{\mathcal{P}} \cup b^{\mathcal{P}} & \qquad (a \cap b)^{\mathcal{P}} \;=\; a^{\mathcal{P}} \cap b^{\mathcal{P}} \\
(\complement(a))^{\mathcal{P}} & = & T_\Sigma \setminus a^{\mathcal{P}} &
\end{array}$$

while the interpretation of the predicate symbol $\subseteq$ is the obvious: $(a \subseteq b)^{\mathcal{P}}$ iff $a^{\mathcal{P}} \subseteq b^{\mathcal{P}}$.

$\mathcal{P}$ is the domain used in most of the papers dealing with set-constraints. Replacing T_Σ with T_Σ^∞, namely the set of infinite trees over Σ, in the definition above, one obtains the domain $\wp(T_\Sigma^\infty)$. This is the domain analyzed in [8]. In [22] the domains used are $\wp(T_\Sigma^+) = \wp(T_\Sigma) \setminus \{\emptyset\}$ and $\wp(T_\Sigma^\infty)^+ = \wp(T_\Sigma^\infty) \setminus \{\emptyset\}$.

Example 3 *Let $\Sigma = \{a, s\}$, $ar(a) = 0$ and $ar(s) = 1$. Then the set-constraint $X \subseteq s(a)$ is satisfiable (two possible evaluations are admitted: $[X/\emptyset]$ and $[X/\{s(a)\}]$). The set-constraint $s(X) \subseteq X \wedge X \not\subseteq \underline{0}$ is also satisfiable by the evaluation $[X/T_\Sigma]$ (notice that without$X \not\subseteq \underline{0}$, also $[X/\emptyset]$ would be a solution). The evaluation $[X/\{s(s(s(\cdots)))\}]$ is a solution over $\wp(T_\Sigma^\infty)$.*

[1] Equality ($\doteq$) is implicitly included in Π, with the usual interpretation $(s \doteq t)^{\mathcal{P}}$ iff $s^{\mathcal{P}} \equiv t^{\mathcal{P}}$. As a matter of fact, it holds that $s \doteq t$ iff $s \subseteq t \wedge t \subseteq s$.

Several authors have investigated the problem of checking the satisfiability of set-constraints, using different techniques and presenting different methods for solving contraints ([3, 2, 14, 15]). In particular, Charatonik and Pacholski [7] have shown that the set-constraint satisfiability problem is decidable and NEXPTIME complete.

Remark 4 *The interpretation of the function application* $f(a_1, \ldots, a_n)$ *and of the* inverse *function application* $f_k^{-1}(a)$ *are a* colored *version of the standard set-theoretic operation of cartesian product and projection, respectively. As a matter of fact, if* $A_1, \ldots, A_n, A$ *are the sets associated with* $a_1, \ldots, a_n, a$*, then* $f(a_1, \ldots, a_n)$ *denotes* $A_1 \times \cdots \times A_n$ *colored by* f *and* $f_k^{-1}(a)$ *is the projection on the direction* k *of the tuples of* A *colored by* f*.*

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

CLP($\mathcal{SET}$) [12, 10] is an instance of the general CLP scheme [18].

Definition 5 *Consider a signature* Σ *of functional symbols such that* $\{\emptyset, \{\cdot \mid \cdot\}\} \subseteq \Sigma$*, a numerable set of variables* $\mathcal{V}$*, and a set of predicate symbols* $\Pi = \{\doteq, \in, \cup_3, ||, \mathsf{c}^{of}\}$*. A* CLP($\mathcal{SET}$)-language *is a language* $\mathcal{L} = \langle \Sigma, \Pi, \mathcal{V} \rangle$*. A* CLP($\mathcal{SET}$)-constraint *is a finite conjunction of literals of* $\mathcal{L}$*.*

The domain where to test the satisfiability of CLP($\mathcal{SET}$)-constraints is defined by taking a quotient of the set of ground terms T_Σ.

Definition 6 *The relation* $\equiv \subseteq T_\Sigma \times T_\Sigma$ *is the least congruence such that:*

$$\begin{array}{rrcl} \forall r, s, t \in T_\Sigma & \{t|\{s|r\}\} & \equiv & \{s|\{t|r\}\} \\ \forall r, s \in T_\Sigma & \{s|\{s|r\}\} & \equiv & \{s|r\} \end{array}$$

and it is closed under the functor application (i.e., if $t_i \equiv t_i'$ *for each* i*, then* $f(\bar{t}) \equiv f(\bar{t'})$*).*

In this way, $\{\cdot \mid \cdot\}$ is interpreted as a set-constructor symbol, not as a list constructor, while free functional symbols are interpreted as usual in Herbrand models. A detailed presentation of the interpretation of the language can be found in [9, 12] and, for this extended case, in [10]. However, denoting as $[t]_\equiv$ the canonical representative (no matter what algorithm is used to choose it) of the congruence class of t, we define:

Definition 7 *Given a CLP(*$\mathcal{SET}$*)-language* $\mathcal{L}$*, the privileged interpretation* $\mathcal{T}$ *for* $\mathcal{L}$ *is* $\mathcal{T} = \langle T_\Sigma/\equiv; (\cdot)^{\mathcal{T}} \rangle$*, where the interpretation function* $(\cdot)^{\mathcal{T}}$ *is defined by* $t^{\mathcal{T}} = [t]_\equiv$ *for any term* $t \in T_\Sigma$*, while the interpretation of the predicate symbols is:*

$$
\begin{array}{rcl}
(s \doteq t)^{\mathcal{T}} & \textit{iff} & s^{\mathcal{T}} = t^{\mathcal{T}} \\
(t \in s)^{\mathcal{T}} & \textit{iff} & \{t \mid s\}^{\mathcal{T}} = s^{\mathcal{T}} \\
(\mathsf{c}^{of}(f(\bar{s}), g(\bar{t}))^{\mathcal{T}} & \textit{iff} & (f(\bar{s}))^{\mathcal{T}} = (g(\bar{t}))^{\mathcal{T}} \\
(\mathsf{c}^{of}(\{t \mid s\}, k))^{\mathcal{T}} & \textit{iff} & (\mathsf{c}^{of}(s, k))^{\mathcal{T}} \\
\textit{for all } m, n, k \textit{ and for all } x_0 \cdots x_m y_0 \cdots y_n z_0 \cdots z_k\textit{:} & & \\
(\cup_3(s_1, s_2, s_3))^{\mathcal{T}} & \textit{iff} & \forall v((v \in s_3)^{\mathcal{T}} \leftrightarrow (v \in s_1)^{\mathcal{T}} \vee (v \in s_2)^{\mathcal{T}}) \wedge \\
 & & \exists k((\mathsf{c}^{of}(s_1, k))^{\mathcal{T}} \wedge (\mathsf{c}^{of}(s_2, k))^{\mathcal{T}} \wedge (\mathsf{c}^{of}(s_3, k))^{\mathcal{T}}) \\
(||(s_1, s_2))^{\mathcal{T}} & \textit{iff} & \forall v((v \in s_1)^{\mathcal{T}} \rightarrow (v \notin s_2)^{\mathcal{T}}) \wedge \\
 & & \exists k((\mathsf{c}^{of}(s_1, k))^{\mathcal{T}} \wedge (\mathsf{c}^{of}(s_2, k))^{\mathcal{T}})
\end{array}
$$

The *CLP* language presented here extends that of [12] with the introduction of the constraint predicate symbols $\cup_3, ||$, and c^{of}, as done in [10]. This extension is justified by the need of augmenting the expressive power of the constraint language as concerns expressivity of the computed solutions. Actually, one could define $\subseteq, \cap, \cup$ and other basic set-theoretical operations also by CLP($\mathcal{SET}$) programs (see [12]). However, it can be proved[2] that given a set-theoretic model M of the language, there is no *open* formula φ in the language based on $\{\emptyset, \{\cdot \mid \cdot\}, \doteq, \in\}$, s.t. $\mathcal{T} \models \forall XY\,(X \subseteq Y \leftrightarrow \exists Z_1 \ldots Z_n\,\varphi)$. Thus, for instance, if $\subseteq$ is programmed, the solutions to the goal $X \subseteq Y$ are the (infinite) $[X = \emptyset], [X = \{Z_1\}, Y = \{Z_1 \mid N\}], [X = \{Z_1, Z_2\}, Y = \{Z_1, Z_2 \mid N\}], \ldots, [Y = X], [Y = \{Z_1 \mid X\}], [Y = \{Z_1, Z_2 \mid X\},], \ldots$. The same holds also for union and intersection, since $X \subseteq Y$ is equivalent to both $X \cup Y = Y$ and $X \cap Y = X$.

When dealing with *hybrid* entities, such as, for instance, the term $\{\emptyset \mid a\}$, a functional or predicate symbol concerning with the *color* of a set is required. In [12] a functional symbol *ker* is adopted. However, a functional symbol affects the interpretation domain. The use of a predicate symbol, c^{of}, allows a clean treatment of such entities (cf. also [9]).

In the same spirit, we have introduced $\cup_3$ as a ternary predicate symbol, rather than a binary functional symbol. Moreover, the predicate $||$, that states the disjointness of two sets, is introduced as a necessary tool for implementing intersection (see [10]):[3]

$$
\begin{array}{rcl}
s \subseteq t & \text{iff} & \cup_3(s, t, t) \\
\cap_3(r, s, t) & \text{iff} & \exists R, S(\cup_3(R, t, r) \wedge \cup_3(S, t, s) \wedge R||S).
\end{array}
$$

A constraint solver for CLP($\mathcal{SET}$)-constraints is described in detail in [12, 10].

Example 8 *Here are some examples of constraints in the CLP($\mathcal{SET}$)-language, along with substitutions that make them satisfiable in the CLP($\mathcal{SET}$)-interpretation $\mathcal{T}$:*

[2] The proof is very technical and outside the scope of this paper.

[3] Strictly speaking, also literals based on predicate symbols $\in$, and $\doteq$ could be equivalently replaced by literals based on $\cup_3$: $s \in t$ iff $\cup_3(t, t, \{s \mid t\})$, $s \doteq t$ iff $\cup_3(s, s, t)$. Therefore, we could avoid considering these symbols as part of the set Π. Notwithstanding, we prefer letting the constraint solver to deal with this kind of operations as primitive constraints, both for the sake of simplicity, and for efficiency reasons.

• $X \notin \{\emptyset|X\} \wedge \{a|Y\} \doteq \{b|Z\}$ *satisfied by the substitution* $[X/\{\emptyset\}, Y/\{b\}, Z/\{a\}]$;
• $\cup_3(X,Y,Z) \wedge \cup_3(X,Y,W) \wedge Z \neq W$ *which is unsatisfiable;*
• $\cup_3(X,Y,\{a|Z\}) \wedge X||\{a|V\} \wedge a \notin Z$ *satisfied by* $[X/\emptyset, Z/\emptyset, V/\emptyset, Y/\{a\}]$.

4 How to compare the two proposals

The main differences between the two classes of languages presented so far are summarized in the table below.

	Set-Constraints	CLP($\mathcal{SET}$)-Constraints
Primitive constraints	$\subseteq, \cup, \cap, \complement$	$\doteq, \in, \cup_3, \|\|$
Domain	Flat hybrid sets	Hereditarily finite hybrid sets
Int. of functional symbols	$\mathcal{P} \models \exists \bar{X}\ f(X_1,\ldots,X_n) \neq \underline{0}$	$\mathcal{T} \models \forall \bar{X}\ X_0 \notin f(X_1,\ldots,X_n)$

Hybrid means that set elements can be (also) non-set objects (those denoted by terms whose outermost functional symbol is a free Herbrand functor). *Flat* means that sets can not be nested, whereas *hereditarily finite* means that sets can contain a *finite* number of elements that can be (also) other *hereditarily finite* sets. Note that the interpretation of the functional symbols is very different in the two domains as pointed out in the table.

In order to compare the two proposals, we need to define a general notion of translation between two languages.

Definition 9 *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 *from* $\mathcal{C}_1$ *into* $\mathcal{C}_2$ *is a function* $\phi : \mathcal{C}_1 \to \mathcal{C}_2$*.* ϕ *is a* conservative translation *under* $\mathcal{M}_1$ *and* $\mathcal{M}_2$ *if for each formula* $\varphi \in \mathcal{C}_1$*:*

$$\exists \sigma_1 \mathcal{M}_1 \models \varphi[\sigma_1] \quad \textit{iff} \quad \exists \sigma_2 \mathcal{M}_2 \models \phi(\varphi)[\sigma_2]\,.$$

The comparison between the language of set-constraints and the language of CLP($\mathcal{SET}$)-constraints is obtained by defining suitable conservative translations from increasingly larger sub-classes of the former to the latter. At the end, we will prove that the language of CLP($\mathcal{SET}$)-constraints is at least as powerful as a non-trivial sub-class of the language of set-constraints, akin to the class of co-definite set-constraints.

As concerns the inverse translation, the problem is the coding of the membership symbol. As a matter of fact, $x \in y$ is set-theoretically equivalent to $\{x\} \subseteq y$. However, it is not possible to express the nesting of a set $\{x\}$ in the language of set constraints. The inability to deal with the predicate $\in$ in the set-constraint domain $\mathcal{P}$ would require us either to restrict to a smaller sub-class of CLP($\mathcal{SET}$)-constraints or to try to extend the language of set-constraints. While the former seems to turn out to impose too severe limitations, the latter would require a non-trivial extension of the set-constraint language which is out of the scope of this paper.

On the other hand, the main difficulties in defining translations from the language of set-constraints to that of CLP($\mathcal{SET}$)-constraints come from the following two features:

• set-constraints can be satisfied using *infinite* sets, whereas $\mathcal{T}$ can have only finite sets;

- as shown in Remark 4, the atoms $X \doteq f(X_1, \ldots, X_n)$ and $X \doteq f_k^{-1}(Y)$ represent a form of cartesian product and projection and are a special case 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 the elements).

As concerns the last point above, it is important to note that in CLP($\mathcal{SET}$) 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. This form stands for the quantified implication $\forall X((X \in s) \rightarrow G)$.

Given a program P, CLP($\mathcal{SET}$) replaces a *RUQ*-goal $(\forall X \in s)G[X, \bar{Y}]$ by the new goal $\mathtt{forall_G(s, \bar{Y})}$ which is defined by the following two clauses added to P:[4]

$\mathtt{forall_G}(\emptyset, \bar{\mathtt{Y}})$.

$\mathtt{forall_G}(\{\mathtt{A} \mid \mathtt{R}\}, \bar{\mathtt{Y}}) \leftarrow \mathtt{A} \notin \mathtt{R} \wedge \mathtt{G}^{[\mathtt{X/A}]} \wedge \mathtt{forall_G}(\mathtt{R}, \bar{\mathtt{Y}})$.

Furthermore, it is easy to generalize such a technique in order to implement by CLP($\mathcal{SET}$) clauses also the more general form of RUQs $(\forall X_1 \in s_1) \cdots (\forall X_n \in s_n) \exists \bar{Z}$ $G[\bar{X}, \bar{Z}]$ where $X_1, \ldots, X_n$ are all distinct variables which do not occur in $s_1, \ldots, s_n$.

Hereafter, we call CLP($\mathcal{SET}$)-constraints, enriched with *RUQ*s, *enriched CLP($\mathcal{SET}$)-constraints*. Enriched CLP($\mathcal{SET}$)-constraints will be used in the next section to represent functional set-expressions and projections of set-constraints.

5 Translating set-constraints to CLP($\mathcal{SET}$)-constraints

In this section we show how to transform a set-constraint into enriched CLP($\mathcal{SET}$)-constraints (CLP($\mathcal{SET}$)-constraints of Def. 5—that are always decidable—plus restricted universal quantifiers applied to them). If the starting constraint admits infinite solutions, the constraint solver of CLP($\mathcal{SET}$) could enter into infinite computation during the satisfiability checking. However, we will identify two sub-classes of set-constraints that can be decided using this method.

Given a set-constraint C, the translation of C can be split into two parts: first we reduce C to an equivalent set-constraint C' in flat form (actually, an extension of the flat form of INES-constraints [22]); then we translate C' to the corresponding CLP($\mathcal{SET}$)-constraints.

Definition 10 *A* **flat-constraint** *is a conjunction of literals of the form:*

$$\begin{array}{lll} Z \doteq \underline{0} & Z \doteq \underline{1} & Z \doteq f(X_1, \ldots, X_n) \\ Z \doteq f_k^{-1}(X) & Z \doteq X \cup Y & Z \doteq X \cap Y \\ Z \doteq \complement(X) & X \subseteq Y & X \not\subseteq Y \end{array}$$

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

Given a set-constraint C, it is straightforward (and always possible) to transform C into an equi-satisfiable flat-constraint C'.

The following is the definition of a function ϕ that translates any atom of a constraint in flat form C' to extended CLP($\mathcal{SET}$)-constraints.

[4] A procedure that transforms {*log*} clauses with *RUQ*s to equivalent CLP($\mathcal{SET}$)-clauses without *RUQ*s is described in detail in [9].

Definition 11 *Let C be a set-constraint and C' be the corresponding constraint in flat form. Let U be a new variable not occurring in C. The translation ϕ is defined as follows: for every literal of C':*

$$
\begin{array}{rcll}
X\pi Y & \stackrel{\phi}{\mapsto} & X\pi Y & \pi \in \{\subseteq, \not\subseteq\} \\
Z \doteq \underline{0} & \stackrel{\phi}{\mapsto} & Z \doteq \emptyset & Z \doteq \underline{1} \stackrel{\phi}{\mapsto} Z \doteq U \\
Z \doteq X\pi Y & \stackrel{\phi}{\mapsto} & \pi(X, Y, Z) & \pi \in \{\cup_3, \cap_3\} \\
Z \doteq \complement(X) & \stackrel{\phi}{\mapsto} & \cup_3(X, Z, U) \wedge X||Z & \\
f(X_1, \ldots, X_n) \subseteq Y & \stackrel{\phi}{\mapsto} & (\forall Z_1 \in X_1) \cdots (\forall Z_n \in X_n)(f(Z_1, \ldots, Z_n) \in Y) & \\
Y \subseteq f(X_1, \ldots, X_n) & \stackrel{\phi}{\mapsto} & (\forall Z \in Y)\exists Z_1, \ldots, Z_n\, (Z \doteq f(Z_1, \ldots, Z_n) \wedge \bigwedge_{i=1}^{n} Z_i \in X_i) & \\
Y \doteq f(X_1, \ldots, X_n) & \stackrel{\phi}{\mapsto} & \phi(Y \subseteq f(X_1, \ldots, X_n)) \wedge \phi(f(X_1, \ldots, X_n) \subseteq Y) & \\
Z \doteq f_k^{-1}(X) & \stackrel{\phi}{\mapsto} & \phi(Z \subseteq f_k^{-1}(X)) \wedge \phi(f_k^{-1}(X) \subseteq Z) & \\
Z \subseteq f_k^{-1}(X) & \stackrel{\phi}{\mapsto} & (\forall Y \in Z)\exists Y_1 \ldots Y_n\, (f(Y_1, \ldots, Y_{k-1}, Y, Y_{k+1}, \ldots, Y_n) \in X) & \\
f_k^{-1}(X) \subseteq Z & \stackrel{\phi}{\mapsto} & (\forall Y \in X)\forall Y_1 \ldots Y_n\, (Y \doteq f(Y_1, \ldots, Y_k, \ldots, Y_n) \rightarrow Y_k \in Z) &
\end{array}
$$

Moreover, if $C' = C'_1 \wedge \ldots \wedge C'_n$ and $vars(C') = \{Z_1, \ldots, Z_m\}$, then $\phi(C') = \phi(C'_1) \wedge \ldots \wedge \phi(C'_n) \wedge (Z_1 \subseteq U) \wedge \ldots \wedge (Z_m \subseteq U) \wedge U \neq \emptyset$. Finally, we set $\phi(C) = \phi(C')$.

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

The last rule seems to cross the syntactical limits of extended CLP($\mathcal{SET}$)-constraints, hiding a form of negation. As a matter of fact, it is equivalent to $(\forall Y \in X)\neg p(Y, Z)$, where the predicate p is defined as: $p(Y, Z) \leftarrow Y = f(Y_1, \ldots, Y_n) \wedge Y_k \notin Z$. If Σ is finite, then we can use a constructive approach to rewrite the RUQ constraint as $(\forall Y \in X) not_p(Y, Z)$ and the predicate not_p is defined as:

$$
\begin{array}{rcll}
not_p(g(\bar{Y}), Z). & & & g \in \Sigma, g \not\equiv f \\
not_p(f(Y_1, \ldots, Y_n), Z) & \leftarrow & Y_k \in Z. &
\end{array}
$$

To avoid the drawbacks of negation, we will make the assumption that, whenever the last rewriting rule needs to be employed, Σ is finite.

We would like to prove that C is satisfiable iff $\phi(C)$ is satisfisfiable (i.e., ϕ is a conservative translation). This is not an immediate result because in the CLP($\mathcal{SET}$) domain there are only finite sets of trees while in the set-constraints domain there are also *infinite sets* (of course, there are set-constraints that are satisfiable only using infinite sets of terms); for instance:

Example 12 *The set-constraint $f(X) \subseteq X \wedge a \subseteq X$ is satisfiable in $\mathcal{P}$ but its least solution is $[X/\{a, f(a), f(f(a)), \ldots\}]$, which clearly involves an infinite set.*

On these constraints CLP($\mathcal{SET}$) generates a computation which never ends (for each variable it collects iteratively the elements of the solution). Of course, we can nevertheless use CLP($\mathcal{SET}$) for testing satisfiability of special sub-classes of set-constraints.

Definition 13 *Let $\mathcal{S}_0$ be the class of set-constraints composed only by set operators (no functional symbols or projections are used).*

For this class it holds the following result (that contains, in the proof, a conservative translation result):

Theorem 14 *Let Σ contain (at least) a constant symbol and a functional symbol. Then every $\mathcal{S}_0$-set-constraint is satisfiable in $\wp(T_\Sigma)$ iff it is satisfiable in $\wp(T_\Sigma)$ using only finite sets of terms.*

Proof. (sketch) Let C be a $\mathcal{S}_0$-set-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-thoretic formulas 'reflecting on the finite'. In other words, $\phi(C)$ is satisfiable iff it is satisfiable over the universe of hereditarily finite and well-founded sets (isomorphical to the universe of $\mathcal{T}$, when no functional symbols are involved), (c.f. [6]).

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 $\mathcal{T}$) of terms. ¿From the property above, there is a solution on hereditarily finite sets for $\phi(C)$. ¿From that solution it is possible to use the constant symbol and the functional symbol of Σ to compute a finite solution for C. □

A larger class for which CLP($\mathcal{SET}$)-constraint solver could be used safely is the class $\mathcal{S}_f$ of finitely satisfiable set-constraint (which includes $\mathcal{S}_0$). It is easy to prove that the translation of constraints of this class using ϕ generates finitely satisfiable CLP($\mathcal{SET}$)-constraints.

However, it is not immediate to find a syntactic characterisation of $\mathcal{S}_f$ and moreover the restriction to this class seems to be too strong (for the applications). So we try to analyze the output of CLP($\mathcal{SET}$) also over set-constraints which are infinitely satisfiable. In next section we will prove that, for the class $\mathcal{S}_2$ (akin to the class of co-definite set-constraints [8]) we can test satisfiability using CLP($\mathcal{SET}$) both for finite and infinite solutions, by checking the partial solution evaluated at a certain step (which depends on the given constraint).

6 Using CLP($\mathcal{SET}$) to decide the class $\mathcal{S}_2$

Now we deal with the class $\mathcal{S}_2$ of set-constraints of the form: $\bigwedge_{i=1}^{n}(\ell_i \subseteq r_i)$ in which:
- $\complement$ and $\underline{1}$ do not appear,
- ℓ_i's contain only variables, constant and function symbols, and the union operator, and
- the projections in r_i's are only applied to unary functional symbols.

Observe that there are two differences between $\mathcal{S}_2$ and the class of so-called *co-definite* set-constraints ([8]). The former enlarges the latter in that in $\mathcal{S}_2$ there are no restrictions about the functional symbols in ℓ_i; on the other hand, in $\mathcal{S}_2$ projections are allowed only if applied to unary functional symbols: a condition not required by co-definite set-constraints. This restriction ensures that the equivalence $t \subseteq f_1^{-1}(s) \quad \text{iff} \quad f(t) \subseteq s$ holds.

Unless explicitly specified, all results will hold both on the interpretation domain $\wp(T_\Sigma)$ and on $\wp(T_\Sigma^\infty)$. As usual, we define a canonical form for constraints that allows an easiest satisfiability analysis.

$\underline{0} \subseteq t$	$\mapsto$	`true` (or omitted)
$t \subseteq s$ s and t ground, $t \not\equiv s$, and (s is a constant or $\underline{0}$ and t is a constant or a term not containing $\underline{0}$)	$\mapsto$	`false`
$f(t_1, \ldots, t_n) \subseteq f(s_1, \ldots, s_n)$	$\mapsto$	$(\bigwedge_{i=1}^{n} t_i \subseteq s_i \vee \bigvee_{i=1}^{n} (t_i \subseteq \underline{0})$
$f(t_1, \ldots, t_n) \subseteq g(s_1, \ldots, s_m)$	$\mapsto$	$\bigvee_{i=1}^{n} t_i \subseteq \underline{0}$
$X \subseteq f(t_1, \ldots, t_n)$ there is $i = 1, \ldots, n$ s.t. $nonvar(t_i)$	$\mapsto$	$\bigwedge_{i=1}^{n} (X_i \subseteq t_i) \wedge X \subseteq f(X_1, \ldots, X_n)$
$f(t_1, \ldots, t_n) \subseteq X$ there is $i = 1, \ldots, n$ s.t. $nonvar(t_i)$	$\mapsto$	$\bigwedge_{i=1}^{n} t_i \subseteq X_i \wedge f(X_1, \ldots, X_n) \subseteq X) \vee$ $\bigvee_{i=1}^{n} (t_i \subseteq \underline{0})$
$t \subseteq g_1^{-1}(s)$ $ar(f) = 1$	$\mapsto$	$f(t) \subseteq s$
$e_1 \cup e_2 \subseteq e$	$\mapsto$	$e_1 \subseteq e \wedge e_2 \subseteq e$
$e \subseteq e_1 \cap e_2$	$\mapsto$	$e \subseteq e_1 \wedge e \subseteq e_2$
$f(t_1, \ldots, t_n) \subseteq$ $f(s_1, \ldots, s_n) \cup f(r_1, \ldots, r_n)$	$\mapsto$	$\bigvee_{i=1}^{n} (t_i \subseteq \underline{0}) \vee (\bigwedge_{i=1}^{n} (t_i \subseteq s_i)) \vee$ $(\bigwedge_{i=1}^{n} (t_i \subseteq r_i))$
$f(t_1, \ldots, t_n) \subseteq f(s_1, \ldots, s_n) \cup g(r_1, \ldots, r_m)$	$\mapsto$	$f(t_1, \ldots, t_n) \subseteq f(s_1, \ldots, s_n)$
$f(t_1, \ldots, t_n) \subseteq g(s_1, \ldots, s_m) \cup h(r_1, \ldots, r_k)$	$\mapsto$	$\bigvee_{i=1}^{n} t_i \subseteq \underline{0}$
$f(t_1, \ldots, t_n) \subseteq X \cup t$	$\mapsto$	$f(X_1, \ldots, X_n) \subseteq X \wedge$ $f(t_1, \ldots, t_n) \subseteq f(X_1, \ldots, X_n) \cup t$
$f(t_1, \ldots, t_n) \subseteq g_1^{-1}(r) \cup s$	$\mapsto$	$f(t_1, \ldots, t_n) \subseteq X \cup s \wedge X \subseteq g_1^{-1}(r)$
$X \subseteq f(t_1, \ldots, t_n) \cup t$	$\mapsto$	$X \subseteq Y \cup t \wedge Y \subseteq f(t_1, \ldots, t_n)$
$X \subseteq g_1^{-1}(r) \cup s$	$\mapsto$	$X \subseteq Y \cup s \wedge Y \subseteq g_1^{-1}(r)$

Figure 1: Rules for the procedure *simpl*

Definition 15 *A constraint C is in* canonical form *(or in the class $\mathcal{S}_1$) if it is a conjunction of atoms of the form: $e \subseteq X$, or $X \subseteq e$, or $X \subseteq Y \cup Z$, where X, Y, Z are variables, and e can be $\underline{0}$, a variable, or a term of the form $f(X_1, \ldots, X_n)$, $n \geq 0$, and X_is are variables. With $|C|$ we denote the number of atoms in C.*

In Fig. 1 we define a set of transformation rules to reduce every $\mathcal{S}_2$-constraint in a disjunction of $\mathcal{S}_1$ constraints or `false`. We call *simpl* the procedure that applies, as much as possible, one of the rules in Fig. 1. It holds that:

Lemma 16 *Given a $\mathcal{S}_2$-constraint C, simpl(C) always terminates and the result is a disjunction of $\mathcal{S}_1$-constraints or* `false`.

Lemma 17 *C is satisfiable iff at least one of the disjoints returned by simpl(C) is satisfiable.*

The rewriting procedure allows to reduce the general satisfiability problem for $\mathcal{S}_2$ to the problem of the satisfiability of $\mathcal{S}_1$-constraints. To face this problem, we first assume the restriction that in C there are no atoms of the form $X \subseteq Y \cup Z$, and state the Theorem 24. Then we show how to extend such a result to the whole class $\mathcal{S}_2$.

We will make use of the following sequence of evalutations that, intuitively, will help us in finding the least model of a $\mathcal{S}_1$-constraint (with restriction):[5]

$$\left\{\begin{array}{rcl} \sigma_0[X] & = & \emptyset \\ \sigma_{n+1}[X] & = & \sigma_n[X] \cup \bigcup_{e \subseteq X \text{ in } C} \sigma_n[e] \cup \bigcup_{Y \subseteq f(\ldots,X,\ldots) \text{ in } C} f_k^{-1}(\sigma_n[Y]) \end{array}\right.$$

and we define the limit of such a succession as $\sigma[X] = \bigcup_{n\in\omega} \sigma_n[X]$. The definition is for variables, but using the interpetation defined in Sect. 2 it can be extended to all terms.

The following technical lemmata is crucial for the successive results:

Lemma 18 *Let C be a $\mathcal{S}_1$-constraint (with restriction). 1) If σ satisfies C and $X \subseteq e$ is in C, then for all integer k, and for each term t $(t \in \sigma_k(X) \rightarrow t \in \sigma_{k+1}(e))$. 2) If σ does not satisfy C, then C is unsatisfiable.*

We introduce the notion of a graph $G(C)$ obtained from a $\mathcal{S}_1$-constraint C.

Definition 19 *Let C be a $\mathcal{S}_1$-constraint. $G(C)$ is the directed labeled multi-graph (i.e., the set of edges is in fact a* multiset*) such that:*
the nodes of $G(C)$ are the variables and the constant occurring in C, and $\underline{0}$;
if $X \subseteq a$ is in C and a is a constant symbol or $\underline{0}$, then $\langle X, a\rangle$ is an edge of $G(C)$, labeled by a_d;
if $X \subseteq f(X_1, \ldots, X_n)$ is in C, then $\langle X, X_1\rangle, \ldots, \langle X, X_n\rangle$ are edges of $G(C)$ labeled by f_d^i;
if $a \subseteq X$ is in C and a is a constant symbol, then $\langle a, X\rangle$ is an edge of $G(C)$, labeled by a_s;
if $f(X_1, \ldots, X_n) \subseteq X$ is in C then $\langle X_1, X\rangle, \ldots, \langle X_n, X\rangle$ are edges of $G(C)$ labeled by f_s;
if $X \subseteq Y \cup Z$ is in C, then $\langle X, Y\rangle, \langle X, Z\rangle$ are edges of C labeled by XYZ.

The graph $G(C)$ is a sort of automaton whose initial nodes are those associated to constants. They start a flow of tokens in the graph that will originate the possible terms that must belong to any solution set of the various variables. This flow of information is a graphical counter-part of the construction of the evaluation σ.

Example 20 *If C is $a \subseteq X \wedge f(X) \subseteq Y \wedge g(Y) \subseteq Z \wedge Z \subseteq g(X)$, then $G(C)$ is:*

$$\begin{array}{ccc} X & \xrightarrow{f_s} & Y \\ {\scriptstyle a_s}\uparrow & \nwarrow^{g_d} & \downarrow{\scriptstyle g_s} \\ a & & Z \end{array} \qquad \begin{array}{ccc} \sigma_1(X) = \{a\} & \sigma_1(Y) = \emptyset & \sigma_1(Z) = \emptyset \\ \sigma_2(X) = \{a\} & \sigma_2(Y) = \{f(a)\} & \sigma_2(Z) = \emptyset \\ \sigma_3(X) = \{a\} & \sigma_3(Y) = \{f(a)\} & \sigma_3(Z) = \{g(f(a))\} \\ \sigma_4(X) = \{a, f(a)\} & \sigma_4(Y) = \{f(a)\} & \sigma_4(Z) = \{g(f(a))\} \\ \ldots & \ldots & \ldots \end{array}$$

The following lemma (whose proof is omitted due to lack of space) reminds (also in the proof) the *pumping lemma* for finite state automata:

[5] The absence of atoms of the form $X \subseteq Y \cup Z$ ensures the existence of a least model

Lemma 21 *Let C be a $\mathcal{S}_1$-constraint (with restriction). If σ does not satisfies C, then there is $X \subseteq e$ in C and a term t such that $t \in \sigma_k[X]$ and $t \notin \sigma_{k+1}[e]$, where $k = 4|C|(|C|+1)$.*

We are ready for the announced theorem:

Theorem 22 *Let C be a $\mathcal{S}_1$-constraint (with restriction). C is unsatisfiable iff there is $X \subseteq e$ in C and a term t such that $t \in \sigma_k[X]$ and $t \notin \sigma_{k+1}[e]$, where $k = 4|C|(|C|+1)$.*

Proof. If C is unsatisfiable, then by definition, σ does not satisfy C. Lemma 21 ensures that there are k and t fulfilling the required property.

If C is satisfiable, then by Lemma 18 (2), σ satisfies C. The claim follows from Lemma 18 (1). □

In Sect. 5 we have shown how to convert a set-constraint into a CLP($\mathcal{SET}$) constraint. The translation introduces constraints based on restricted universal quantifiers. The technique of CLP($\mathcal{SET}$) for solving a constraint of this form is that to *build* a set one element per time. Such a (blind) search strategy is highly non-deterministic. However, what we argue here is that, provided such non-determinism is controlled (fairness) there is a strong connection between the search strategy of CLP($\mathcal{SET}$) and the construction of σ using the graph $G(C)$.

Definition 23 *Given an enriched CLP($\mathcal{SET}$) constraint $C_1 \wedge \ldots \wedge C_n$, a computation of the satisfiability algorithm is* fair *if between two consecutive insertions of elements in the set built in a constraint C_i involving RUQs, all the sets that must be built to solve constraints $C_1 \wedge \ldots \wedge C_{i-1} \wedge C_{i+1} \wedge \ldots \wedge C_n$, have been updated.*

Any step of the construction of σ is simulated by at most a number α of actions of the satisfiability algorithm (it depends from the granularity of the analysis of the satisfiability algorithm implementation). The following theorem is extremely interesting, since it connects the two worlds of set-constraints and of logic programming with sets. Moreover, it allows the use of a satisfiability algorithm developed for finite terms (and sets) to check also the satisfiability of a set constraint also in the universe of infinite terms.

Theorem 24 *Let C be a $\mathcal{S}_1$-constraint (with restriction), C' the result of its translation as CLP($\mathcal{SET}$) constraint and $h = \alpha k$. Then, if a fair computation of the constraint solver of CLP($\mathcal{SET}$) on C':*

- *terminates with success in less than h steps, then C is satisfiable in $\wp(T_\Sigma)$ (hence in $\wp(T_\Sigma^\infty)$);*
- *terminates definitely with failure in less than h steps, then C is unsatisfiable in $\wp(T_\Sigma^\infty)$ (hence in $\wp(T_\Sigma)$);*
- *does not terminate in less than h steps, then C is satisfiable in $\wp(T_\Sigma^\infty)$ (but not necessarily in $\wp(T_\Sigma)$).*

As described in [22], the most interesting satisfiability result concerns interpretations on $\wp(T_\Sigma^\infty)$, thus the CLP($\mathcal{SET}$) approach solves exactly that problem.

What remains to do here is to extend the results of Theorems 22 and 24 to the full class of $\mathcal{S}_1$-constraints (without restrictions). As a corollary, and using Lemmata 16 and 17, this result will hold also on the (apparently) wider class of $\mathcal{S}_2$-constraints.

For doing that, we need to consider atoms of the form: $X \subseteq Y \cup Z$. Reasoning on graphs, this kind of atoms induces a form of non-determinism. Intuitively, this fact reflects in the definition of σ in such a way that to compute $\sigma_{k+1}[Y]$ and $\sigma_{k+1}[Z]$ we need to split (non-deterministically) $\sigma_k[X]$. Therefore, the definition of σ must be modified; with this definition, however, Theorems 22 and 24 can be proved without the restriction on C with the only difference in computing the number k used in the statements, that now grows exponentially w.r.t. $|C|$.[6]

7 Conclusions

In this paper we have tried to establish a link between the two research areas of set-constraints and constraint logic programming with sets. In particular, we have shown how set-constraints can be rewritten in the language of CLP($\mathcal{SET}$) and how some classes of set-constraints can be decided using the CLP($\mathcal{SET}$) constraint solver. The proposed rewriting procedure is indeed quite complex; however, its aim is mainly that of proving the expressivity of the CLP($\mathcal{SET}$)constraint language. For this reason, the complexity of the whole process has not been analyzed in detail, yet. It can be a good starting point for the continuation of the work.

References

[1] A. Aiken, D. Kozen, M. Vardi, and E. Wimmers. The complexity of set constraints. In E. Börger, Y. Gurevich, and K. Meinke, eds., *Proc. 1993 Conf. on CSL*, vol. 832 of *LNCS*, pp. 1–17. Springer-Verlag, Berlin, 1993.

[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. LICS*, pp. 329–340. IEEE, 1992.

[4] F. Baader and K. Schulz. Combination of constraint solvers for free and quasi-free structures. *Theoretical Computer Science*, 192:107–161, 1998.

[5] P. Bruscoli, A. Dovier, E. Pontelli, and G. Rossi. Compiling Intensional Sets in CLP. In Pascal Van Entenryck, ed., *Proc. Eleventh ICLP*, pp. 647–661. The MIT Press, Cambridge, Mass., 1994.

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

[7] W. Charatonik and L. Pacholski. Negative set constraints with equality. In *Proc. 9th Symp. LICS*. IEEE, 1994.

[6] Actually, we are looking for a simpler proof that, making use of classical results on automata, becomes simpler than the current one that analyzes a great variety of cases. The problem is to map our graphs to (standard) automata.

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

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

[10] A. Dovier, C. Piazza, E. Pontelli, and G. Rossi. On the Representation and Management of Finite Sets in CLP-languages. NMSU-CSTR-97-18, Dept. of Computer Science, New Mexico State University, USA, December 1997.

[11] A. Dovier, E. Pontelli, and G. Rossi. The CLP language {log}, and the relation between Intensional sets and Negation. NMSU-CSTR-9503, Dept. of Computer Science, New Mexico State University, USA, March 1995.

[12] A. Dovier and G. Rossi. Embedding Extensional Finite Sets in CLP. In D. Miller, ed., *Proc. of ILPS'93*, pp. 540–556. The MIT Press, Cambridge, Mass., 1993.

[13] C. Gervet. Conjunto: Constraint Logic Programming with Finite Set Domains. In M. Bruynooghe, ed., *Proc. of ILPS'94*, pp. 339–358. The MIT Press, Cambridge, Mass., 1994.

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

[15] R. Gilleron, S. Tison, and M. Tommasi. Solving systems of set constraints using tree automata. In *Proc. Symp. Theoretical Aspects of Computer Science*, vol. 665 of *LNCS*, pp. 505–514. Springer-Verlag, Berlin, 1993.

[16] N. Heintze and J. Jaffar. A finite presentation theorem for approximating logic programs. In *Proc. 17th POPL*, pp. 197–209. ACM, 1990.

[17] N. Heintze and J. Jaffar. Set Constraints and Set-Based Analysis. Technical report, Carnegie Mellon University, 1994.

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

[19] D. Kozen. Logical Aspects of Set Constraints. In E. Börger, Y. Gurevich, and K. Meinke, eds., *Proc. 1993 Conf. on CSL*, vol. 832 of *LNCS*, pp. 175–188. Springer-Verlag, Berlin, 1993.

[20] D. Kozen. Set Constraints and Logic Programming. Technical report 94-1467, Computer Science Department, Cornell University, 1994.

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

[22] M. Müller, J. Niehren, and A. Podelski. Inclusion constraints over non-empty sets of trees. In M. Dauchet, ed., *Proc. 9th TAPSOFT*, vol. 1214 of *LNCS*, pp. 345–356. Springer-Verlag, Berlin, 1997.

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

[24] K. Stefánsson. Systems of set constraints with negative constraints are NEXPTIME-complete. In *Proc. 9th LICS*. IEEE, 1994.