Set-Based Constraints

Agostino Dovier* Carla Piazza* Gianfranco Rossi†

Abstract

The notion of *set constraint* has been presented in literature with two different meanings and aims. Each of them allows to deal with a particular class of *set based* formulae. We compare the two notions and present their satisfiability problem as instances of the more general framework of *Computable Set Theory*. We show that there are large classes of formulae for which both proposals provide suitable procedures for testing satisfiability with respect to a given privileged interpretation. We show examples of how these constraints can be used for set-based analysis and for problem solving in general.

Keywords: Set constraints, Constraint Logic Programming with Sets, Computable Set Theory

1 Introduction

Which kind of set-based constraints?

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, along with an interpretation structure $\mathcal{A}$ which provides a set-theoretical interpretation for these symbols.

A particular class of set-based constraints, usually simply called *set constraints*, have been extensively studied in recent years and used as a convenient formalism for program analysis (see [1, 2] for an introduction).

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 $e_1 \not\subseteq e_2$ [4, 24]. In this paper we will refer to this kind of set-based constraints as *Set Inclusion constraints* (abbreviated as *SI-constraints*).

Another different kind of set-based constraints is that described in [16]. These constraints have been introduced in the context of constraint logic programming (CLP) languages with sets, and are usually referred to as $\mathcal{SET}$-constraints.

A $\mathcal{SET}$-constraint is a conjunction of literals of the form

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

*Dipartimento di Matematica e Informatica, Università di Udine. Via delle Scienze 206, 33100 Udine (Italy). dovier|piazza@dimi.uniud.it

†Dipartimento di Matematica, Università di Parma. Via M. D'Azeglio 85/A, 43100 Parma (Italy). gianfr@prmat.math.unipr.it

where $\|$ 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 non-set terms).

In this paper we will refer to this kind of set-based constraints as *Set Membership constraints* (abbreviated as *SM-constraints*).

General aim

The aim of this paper is twofold:

(i) to show that set-based constraints are not necessarily limited to SI-constraints; other classes of decidable set-based formulae have been studied and can be used as well in a number of applications;

(ii) to show that set constraint-based program analysis is not necessarily the only possible application of set-based constraints; a number of different applications, ranging from combinatorial problem formulation to temporal logic satisfiability problems can take advantage from the availability of set-based constraint languages.

As regards aim (i), we describe precisely both SI- and SM-constraints, using uniform notation and style. We show that 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. Notwithstanding, other classes of set-based constraints can be dealt with in one proposal and not in the other and vice versa. As a non-trivial example to support this claim we show how using SM-constraints one can solve the *groundness analysis* problem for logic programs.

Furthermore, we observe that the study of set-based constraints is strongly related to work done in the so-called *Computable Set Theory* area (C.S.T.), a fruitful research stream born in the 1970's at the New York University thanks to the initial ideas and subsequent stimulus of J. T. Schwartz (see [8, 9] for a wide survey on the topic). C.S.T. was mainly developed in reply to the need to enhance the inferential engine for theorem provers and for the implementation of the imperative language with sets SETL [33]. So far, work in C.S.T. has lead to identify increasingly larger classes of computable formulae of suitable sub-theories of the general Zermelo-Fraenkel set-theory (i.e., set-based formulae) for which satisfiability is decidable. Further extensions of these classes are still under investigation at present. Although a lot of similarities exist between the field of C.S.T. and those dealing with set-based constraints (more evident for SM-constraints, not yet explored for SI-constraints) it is worth trying to involve also C.S.T. in our discussion. Precisely, the satisfiability problem for SI- and SM-constraints can both be seen as instances of C.S.T. problems.

Which kind of applications?

Advantages of using (set-based) constraint formulations in program analysis are well motivated in [2]. According to [2], the SI-constraint language "is rich enough to describe all of the data types commonly used in programming, and this is the property that makes set constraints a useful tool for program analysis".

For example, the set of all lists with elements drawn from the set X and built using the constructors `cons` and `nil` in the usual (LISP like) way can be described as the set assigned to Y by the solution of the following SI-constraint:

$$Y = \texttt{cons}(X, Y) \cup \texttt{nil}.$$

Thus, SI-constraints are often used to solve *type inference* problems in statically typed functional languages. As a matter of fact, a possible way to specify simple type inference uses systems of type equations; these, in turn, can be conveniently reformulated as SI-constraints, using inclusion instead of equality. In this way, if the SI-constraints are solvable, then the program under analysis is typable and the types are exhibited by the solutions of the constraints [2].

Similarly, also SM-constraints can be used to solve a number of program analysis problems. For example, many classical dataflow computations for imperative languages, such as live variable analysis, reaching definitions, and constant propagation, to mention those considered also in [2], can be formulated using SM-constraints, as well. In general, program analysis problems for which we can find a set-based abstraction which does not necessarily involve infinite sets (e.g., the groundness analysis problem) are likely to be solved using SM-constraints. Surely, this is not the case of type inference, since describing data types usually requires infinite sets, but for other properties, like those cited above, using SM-constraints represents a viable alternative.

A well-known criticism to using set-based constraints for program analysis is efficiency. Indeed, as stated in [2], "Beyond decidability, we would like efficient algorithms and algorithms that compute finite representations of solutions. In these areas the state of knowledge is incomplete. Currently, ... the complexity of solving general set constraints is high. Satisfiability of set constraints is NEXPTIME-complete;". Thus, "The complexity results strongly suggest that analysis based on solving set constraints in their full generality is infeasible. However, there are many very useful polynomial time fragments of full theory, and it is this tractable sub-theories that are our focus ..." [2].

However, set-based program analysis is absolutely not the only possible application of set-based constraints (and of Computable Set Theory in general). A further aim of this paper is to prove this claim by showing a number of simple examples relating set-based constraints to application areas other than program analysis. Further support to this claim comes from the various applications which are, at some extent, based on set-based (or even multiset-based) constraints and which are described in the literature. Among them we can mention decidability analysis for modal/temporal logic via translation to set constraints [12], uses of hypersets for solving automata problems [14], multiset rewriting programming [6].

Actually, most of these applications rely on the availability of a *Constraint Logic Programming* (CLP) language where set-based constraints can be embedded. As a matter of fact, both SI-constraints and SM-constraints—along with their constraint solving procedures—can be immediately used to obtain specific instances of CLP languages with sets. In particular, the two instances, CLP($\mathcal{SC}$) and CLP($\mathcal{SET}$), based on SI-constraints and SM-constraints, respectively, are described in the literature [29, 16].

Contents

The paper is organized as follows. In Sections 2 and 3 we briefly present the two set-based constraint languages, namely SI-constraints and SM-constraints. Some hints on the satisfiability problem for set-based constraints are provided in the subsequent section (Section 4). In the same section we also provide an overview of the Computable Set Theory area. Section 5 is devoted to compare the two constraint languages by precisely identifying which kind of constraints can be dealt with in each of them. A non-trivial example in the domain of set-based program analysis using SM-constraints is shown in Section 6. A few other examples using SM-constraints in different application domains are shown in the next section (Section 7). Section 8 addresses the CLP issue, referring in particular to CLP($\mathcal{SC}$) and CLP($\mathcal{SET}$), the CLP instances based on the domain of SI- and SM-constraints, respectively. Some discussion about the translation of (subclasses of) SI-constraints into CLP($\mathcal{SET}$) programs is also provided. Finally, some conclusions are drawn in Section 9 along with some hints about possible future work.

2 SI-Constraints

The notation and definitions sketched here can be found, e.g., in [29, 2].

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. We will use Σ to denote the set of constant and function symbols save $\underline{0}, \underline{1}, \cup, \cap, \complement$ and $ar : \Sigma \longrightarrow \mathbb{N}$ to denote the associated arity function. T_Σ denotes the set of ground first-order terms over Σ.

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

Example 2.1 *(SI-constraints) The following set-based formulae are SI-constraints:*

i. $a \subseteq X \wedge b \subseteq Y \wedge c \subseteq Y \wedge f(X, Y) \subseteq Z$

ii. $a \subseteq X \wedge s(X) \subseteq X$

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: $(\cdot)^{\mathcal{P}}$ is used to denote the interpretation function for the language of SI-constraints, defined as:

$$\begin{array}{rcll} (0)^{\mathcal{P}} & = & \emptyset & \\ (1)^{\mathcal{P}} & = & T_\Sigma & \\ (f(a_1, \ldots, a_n))^{\mathcal{P}} & = & \{f(t_1, \ldots, t_n) \,:\, \bigwedge_{i=1}^{n} t_i \in (a_i)^{\mathcal{P}}\} & \text{for } f \in \Sigma \\ (f_i^{-1}(a))^{\mathcal{P}} & = & \{t_i \,:\, \exists t_1, \ldots, t_n\, f(t_1, \ldots, t_n) \in (a)^{\mathcal{P}}\} & \text{for } f \in \Sigma \end{array}$$

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

Example 2.2 *The interpretation of*

$$f(a \cup b, c)$$

is the set

$$\{f(a, c), f(b, c)\}$$

while the interpretation of

$$f_1^{-1}(f(a \cup b, c))$$

is the set

$$\{a, b\}$$

(note that the interpretation of the constant a is $\{a\}$).

Example 2.3 *(Satisfiable SI-constraints) Constraint* (i) *in Example 2.1,*

$$a \subseteq X \wedge b \subseteq Y \wedge c \subseteq Y \wedge f(X,Y) \subseteq Z$$

has the (least) solution

$$X/\{a\}, Y/\{b,c\}, Z/\{f(a,b), f(a,c)\}.$$

The SI-constraint

$$a \subseteq X \wedge s(X) \subseteq X$$

has the (least) infinite solution

$$X/\{a, s(a), s(s(a))), \ldots\}.$$

3 SM-Constraints

The notation and definitions of this section can be found, e.g., in [15, 16].

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). Σ denotes 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: $\doteq, \in, \cup_3, \|$ (all 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 $T_\Sigma/\!\equiv$ of ground first-order terms over Σ modulo the least congruence relation $\equiv$ obtained by the reflexive, symmetric, and transitive closure of the following axioms:

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

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, 19].

Moreover,

- *equality* is the set-theoretic equality, implied by the two equational axioms for $\{\cdot \mid \cdot\}$ given above (see [19]), i.e., $s \doteq t$ if and only if $[s]_\equiv = [t]_\equiv$;
- $s \in t$ holds if $[t]_\equiv$ is of the form $\{t_1, t_2, \ldots, [s]_\equiv \mid t'\}$;
- $\cup_3(r, s, t)$ holds, intuitively, when $[t]_\equiv$ represents the set $[r]_\equiv \cup [s]_\equiv$;
- $\parallel (s, t)$ holds when $[s]_\equiv$ and $[t]_\equiv$ do not share elements.

Following [29] and [16] we will refer to the constraint domain of SI-constraints and to that of SM-constraints as $\mathcal{SC}$ and $\mathcal{SET}$, respectively. Moreover, the structure $\langle \wp(T_\Sigma), (\cdot)^{\mathcal{P}} \rangle$ used for SI-constraints will be referred to as $\mathcal{A}_{\mathcal{SC}}$, while the structure $\langle T_\Sigma/\equiv, (\cdot)^{\mathcal{T}} \rangle$ used for SM-constraints will be referred to as $\mathcal{A}_{\mathcal{SET}}$.

Example 3.1 *(Satisfiable SM-constraints) The following formulae are all SM-constraints:*

i. $\emptyset \in X \wedge X \in Y \wedge Y \in Z$
satisfiable in $\mathcal{A}_{\mathcal{SET}}$*, e.g., by* $X/\{\emptyset\}, Y/\{\{\emptyset\}\}, Z/\{\{\{\emptyset\}\}\}$;

ii. $X \notin \emptyset$
satisfiable in $\mathcal{A}_{\mathcal{SET}}$ *by any valuation for the variable* X;

iii. $\{a \mid X\} \doteq \{b \mid Y\}$
satisfiable in $\mathcal{A}_{\mathcal{SET}}$*, e.g., by* $X/\{b\}, Y/\{a\}$;

iv. $a \in X \wedge X \neq \{Y\}$
satisfiable in $\mathcal{A}_{\mathcal{SET}}$*, e.g., by* $X/\{a, b\}, Y/a$.

The SM-constraint solver transforms any given SM-constraint into an equi-satisfiable disjunction of constraints in a special form, called the *solved form*, which is proved to be always satisfiable in $\mathcal{A}_{\mathcal{SET}}$. A solved form constraint can be used as a finite representation of the possible infinite set of solutions of the given constraint. From a solved form constraint C it is possible to assign terms to the variables occurring in it in order to obtain a solution. Moreover, all the solutions of C can be obtained in this way.

Example 3.2 *(Computed answers) The following answers are computed for the constraints in Example 3.1*

i. $X \doteq \{\emptyset \mid N_1\} \wedge Y \doteq \{\{\emptyset \mid N_1\} \mid N_2\} \wedge Z \doteq \{\{\{\emptyset \mid N_1\} \mid N_2\} \mid N_3\}$;

ii. `true`;

iii. $X \doteq \{b \mid N\} \wedge Y \doteq \{a \mid N\}$;

iv. $(X \doteq \{a \mid N\} \wedge Y \neq a) \vee (X \doteq \{a, M \mid N\} \wedge Y \neq M)$.

For instance, the solution to constraint (*iv*) *shown in Example 3.1 is obtained from the second disjunct of its computed answer by assigning* b *to* M, $\emptyset$ *to* N *and* a *to* Y.

4 The Satisfiability Problem

SI- and SM-constraints define classes of decidable set-based formulae. In fact, given a SI-constraint (resp., a SM-constraint) C, it is always decidable whether $\mathcal{A}_{\mathcal{SC}} \models \vec{\exists} C$ (resp., $\mathcal{A}_{\mathcal{SET}} \models \vec{\exists} C$).

Satisfiability of SI-constraints has been investigated by several authors ([5, 4, 24, 25]), taking into account different (sub-)classes of SI-constraints, and using different techniques and methods to solve them (mostly based on the use of *tree-automata* [22]). In particular, [11] shows that the satisfiability problem for general SI-constraints is decidable and NEXPTIME complete.

A complete constraint solver for SM-constraints is described in detail in [16]. The technique used to check satisfiability is based on a reduction of any given constraint to a normal form (the *solved form*) which is always satisfiable. The computational complexity of this problem—for the primitive constraints mentioned above—is shown to be NP-complete (see [9]). The constraint solver for SM-constraints, moreover, always returns the complete (though not always minimal) set of solutions for the given constraint. This set of solutions is implicitly and precisely characterized by the finite set of solved form constraints returned by the constraint solver.

Observe that decidable set-theoretical operations are not limited to those provided as primitives by the constraint language. As a matter of fact, other basic set-theoretical operations can be easily defined in terms of the existing ones as (complex) constraints. In particular, equality can be defined in the language of SI-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 predicates $\cap_3$ and $\setminus_3$ (with the obvious interpretations) can be easily defined in the language of SM-constraints as follows:

$$\begin{array}{rll} s \subseteq t & \text{if and only if} & \cup_3(s,t,t) \\ s \not\subseteq t & \text{if and only if} & \exists Z(Z \in s \wedge Z \notin t) \\ \cap_3(r,s,t) & \text{if and only if} & \exists R, S(\cup_3(R,t,r) \wedge \cup_3(S,t,s) \wedge R \parallel S) \\ \setminus_3(r,s,t) & \text{if and only if} & \exists W(\cap_3(r,s,W) \wedge \cup_3(W,t,r) \wedge W \parallel t) \end{array}$$

The right-hand sides of these definitions are SM-constraints.

As mentioned in the introduction, studying solvers for set-based constraints, or, more generally, decision algorithms for classes of set-based formulae, is the main activity of the *Computable Set Theory* area [8, 9]. All classes of formulae studied in C.S.T. allow to employ:

- the constant symbol $\emptyset$
- variables
- equality $=$ and membership $\in$
- logical connectives $\wedge, \vee, \neg$

A first taxonomy of decidable classes is concerned with quantifier free formulae and it is based on the form of the theory which describes the behaviour of the operators involved in the formulae. Some of the theories which have been extensively studied in C.S.T. are the following:

(i) The theory obtained by adding $\cup$ and the finite set constructor $\{\_, \ldots, \_\}$ to the initial endowment (observe that, in absence of free constant and function symbols, SM-constraints are included in this class, since $\{t \mid s\}$ can be expressed as $\{t\} \cup s$).

(ii) The *Multilevel Syllogistic (MLS)* theory obtained by adding $\cap, \subseteq, \setminus, \cup$ to the initial endowment.

(iii) The *Multilevel Syllogistic with Singleton (MLSS)* theory obtained by adding the finite set constructor $\{\_, \ldots, \_\}$ to MLS.

(*iv*) The theory obtained by adding the powerset operator $\wp$ ($\wp(x) = \{y : y \subseteq x\}$) to MLSS.

(*v*) The theory obtained by adding the unary union $\bigcup$ ($\bigcup x = \{y : (\exists z \in x)(y \in z)\}$) to the theory cited in (*iv*).

All the theories from (*i*) to (*iv*) are decidable and, moreover, a formula is satisfiable if and only if it is satisfiable in the model of hereditarily finite sets, whose domain is $T_{\{\emptyset,\{\cdot\,|\,\cdot\}\}}/\equiv$ as in Section 3. Theory (*v*) instead is still decidable but the finite model property does not hold anymore.

A second taxonomy of decidable classes is based on the form of quantifiers which can occur in front of a quantifier free formula of the initial endowment described above. Set operators can be mimicked by formulae of this kind and their definition can be explicitly attached to a formula. For instance $x \subseteq y$ can be expressed by $(\forall z \in x)(z \in y)$.

(*i*) The class of formulae of the form $Q_1 \cdots Q_n p$ where p a quantifier free formula and all Q_is are either of the form $(\forall y_i \in z_i)$ or of the form $(\exists y_i \in z_i)$, and moreover no z_j is also a y_i.

(*ii*) The class briefly referred as $\exists^*\forall$ of the formulae of the form $\exists x_1 \cdots \exists x_n \forall y p$ where p a quantifier free formula and $n \geq 0$.

A detailed review of the main results of C.S.T. can be found in [9], §. 6.4.

5 Comparison

In spite of their different origins and aims, the two constraint languages described so far present many differences but also many similarities. Thus, we can identify classes of set-based constraints for which both proposals provide suitable procedures for testing constraint satisfiability with respect to a given privileged interpretation, as well as classes of set-based constraints which can be expressed in one of the two languages but not in the other. This section is devoted to identify more precisely such classes.

5.1 Finite vs. infinite sets

A first notable difference between SI- and SM-constraints concerns finite and infinite satisfiability. We have already seen (Example 2.3) that there are simple SI-constraints that admit solutions in models with infinite sets but not in models of finite sets: for example, the SI-constraint

$$a \subseteq X \wedge s(X) \subseteq X \tag{1}$$

admits only infinite sets as its solutions, the least one being

$$X/\{a, s(a), s(s(a))), \ldots\}.$$

On the other hand it can be proved (see, e.g., [9], § 6.4), that all SM-constraints, if satisfiable, are satisfiable also in a model of finite sets (this feature is called the property of "reflection on the finite"). This fact does not forbid the existence of infinite set solutions for SM-constraints. For instance, the constraint

$$X \in Y \wedge X \neq \emptyset$$

admits the finite set solution $X/\{\emptyset\}, Y/\{\{\emptyset\}\}$ but also the (infinite set) solution

$$X/\{\emptyset, \{\emptyset\}, \{\{\emptyset\}\}, \ldots, \}, Y/\{\{\emptyset, \{\emptyset\}, \{\{\emptyset\}\}, \ldots, \}, \{\emptyset\}, \{\{\emptyset\}\}, \ldots\}$$

As a matter of fact, the theory underlying the SM-constraint domain (cf. [15, 16]) does not prohibit sets to be infinite. However, the selected interpretation domain is limited to contain only finite sets, but this is not a restriction for the satisfiability of a given constraint. Thus, for instance, the equation $X \doteq \{a \,|\, Y\}$ represents, in our domain, any finite set containing the element a, but it could be used, as well, to represent an infinite set containing the element a.

It is proved (see [8], p. 204) that the simplest first-order formula in the language of SM-constraints that forces infinite solutions requires restricted universal quantification with a clever nesting. Why are SI-constraints able to force infiniteness? Consider the constraint (1) above. The two constraints $a \subseteq X$ and $s(X) \subseteq X$ could be expressed using SM-constraints as well: we could write $a \in X$ and $\cup_3(s(X), X, X)$. What is not expressible in SM-constraints is the meaning of $s(X)$: $s(X) = \{s(t) : t \in X\}$. This is a (weak) form of intensional set former, a very strong constructor of set theory justified by the *Comprehension axiom* (see [30]), the axiom that originally led to the Russell's paradox. This intensional definition of sets—that is, the possibility of defining a set by stating a property that must be satisfied by all its elements rather than by explicitly enumerating them—is not available in the language of SM-constraints.

5.2 Nested vs. flat sets

On the opposite side, there are SM-constraints which are not expressible as SI-constraints. The problem arises with SM-constraints involving membership in conjunction with nested sets (i.e., sets whose elements can be themselves sets). Remember that SI-constraints do not include the membership predicate and they allow sets to be only flat (namely, subsets of the standard Herbrand Universe), that is elements of a set can not be sets themselves. To express membership in SI-constraints one could

- use a unary function symbol f to denote the singleton set constructor
- mapping $t \in s$ into $f(t) \subseteq s$.

Assume, for the sake of simplicity, that Σ for the language of SM-constraints is simply $\{\emptyset, \{\cdot \,|\, \cdot\}\}$. First we note that we can not use the constant $\underline{0}$ of SI-constraints 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 encoded as a. In this way, we could define a translation I from terms of the language of SM-constraints into terms of the language of SI-constraints as follows:

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

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 SM-constraint

$$\{\emptyset\} \in \{\{\emptyset, \{\emptyset\}\}\},$$

which is false in the domain of SM-constraints, would be translated into

$$f(f(a)) \subseteq f(a \cup f(a)),$$

which is satisfiable as a SI-constraint.

5.3 A common decidable class of set-based constraints

In spite of the differences highlighted above, we can identify classes of set-based formulae which can be dealt with equally well in both languages. The following general notion of *translation* between two languages will be useful in the sequel to precisely single out cases in which the two languages can be considered "equivalent".

Definition 5.1 *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 $\mathcal{C}_1$ into $\mathcal{C}_2$, $\phi : \mathcal{C}_1 \to \mathcal{C}_2$, is a* conservative translation *under $\mathcal{M}_1$ and $\mathcal{M}_2$ if for each formula $\varphi \in \mathcal{C}_1$:*

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

where σ_i is a valuation for the variables into the domain $\mathcal{M}_i$, for $i = 1, 2$.

Our goal here is to define a conservative translation from SI-constraints into SM-constraints. The main difficulties in defining one such translation come from the fact that the atoms $X \doteq f(X_1, \ldots, X_n)$ and $X \doteq f_k^{-1}(Y)$ of SI-constraints, which represent a form of cartesian product and projection, are special cases of *intensionally defined sets*. As noted above, this form of set designation is not available in the language of SM-constraints. We restrict, therefore, our attention to a simple (sub-)class of SI-constraints—here called $\mathcal{S}_0$—composed only of set operators, that is, not including any function symbol nor projections.

Definition 5.2 *The class of* $\mathcal{S}_0$-constraints *is described by the following grammar (X denotes a generic variable):*

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

For this restricted class of SI-constraints (which, however, is not limited to finite solutions) we can find a translation into SM-constraints which can be proved to be conservative. Moreover, we prove, using C.S.T. methods, that the satisfiability test for $\mathcal{S}_0$ is NP-complete. A similar result concerning only constraints of the form $s \subseteq t$ have been proved in [3]. However, the proof using C.S.T. is new and promising.

The translation can be split into two parts: first we reduce the given $\mathcal{S}_0$-constraint C to an equivalent $\mathcal{S}_0$-constraint C' in a flat form, then we translate C' into the corresponding SM-constraint.

Definition 5.3 *A SI-constraint in* flat form *is a conjunction of literals of the form:*

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

where π is a predicate symbol in $\{\subseteq, \supseteq\}$, X, X_i, Y, and Z are variables, and f is a function or a constant symbol.

The flat form is chosen in such a way to simplify the successive translation into SM-constraints.[1] The flat form of a given SI-constraint C can be obtained by repeatedly applying to any of its conjuncts the following rewriting rules:

1. if $c \equiv s \; op \; t$, op in $\subseteq, \not\subseteq$, replace c with $M \doteq s \wedge N \doteq t \wedge M \; op \; N$, M, N new variables;

2. if $c \equiv X \doteq h(s_1, \ldots, s_m)$, h in $f, f_i^{-1}, \cup, \cap, \complement$, replace c with $X \doteq h(N_1, \ldots, N_m) \wedge N_1 \doteq s_1 \wedge \ldots \wedge N_m \doteq s_m$.

Polynomial execution time and correctness of the rewriting procedure are immediate to verify.

In Figure 1 we show the definition of a function ϕ that translates any literal of a flat $\mathcal{S}_0$-constraint C into the corresponding SM-constraint. Observe that the function symbols $\cup$ and $\cap$ in SI-constraints are replaced by the ternary predicate symbols $\cup_3$ and $\cap_3$ in SM-constraints. Moreover, as shown in Section 4, in the language of SM-constraints we can write $s \subseteq t$ for $\cup_3(s, t, t)$ and $s \not\subseteq t$ for $\exists Z \, (Z \in s \wedge Z \notin t)$.

$X \; \pi \; Y$	$\overset{\phi}{\mapsto}$	$X \; \pi \; Y$	π in $\{\subseteq, \not\subseteq\}$
$Z \doteq \underline{0}$	$\overset{\phi}{\mapsto}$	$Z \doteq \emptyset$	
$Z \doteq \underline{1}$	$\overset{\phi}{\mapsto}$	$Z \doteq U$	
$Z \doteq X \; \pi \; Y$	$\overset{\phi}{\mapsto}$	$\pi_3(X, Y, Z)$	π in $\{\cup, \cap\}$
$Z \doteq \complement(X)$	$\overset{\phi}{\mapsto}$	$\cup_3(X, Z, U) \wedge X \parallel Z$	

Figure 1: Translation for flat $\mathcal{S}_0$-literals

Function ϕ can be immediately extended to any $\mathcal{S}_0$-constraint C as follows.

Definition 5.4 *Let C be a $\mathcal{S}_0$-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 C' is defined by the rewriting rules of Figure 1. Let C' be of the form $C'_1 \wedge \ldots \wedge C'_n$ and $vars(C') = \{Z_1, \ldots, Z_m\}$.[2] Then $\phi(C) = \phi(C'_1) \wedge \ldots \wedge \phi(C'_n) \wedge (Z_1 \subseteq U) \wedge \ldots \wedge (Z_m \subseteq U) \wedge U \neq \emptyset$. 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 $\phi(C)$.*

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

Example 5.5 *Consider the $\mathcal{S}_0$-constraint*

$$X_1 \cup X_2 \subseteq X_2 \wedge X_1 \not\subseteq \underline{0} \wedge X_2 \cap X_3 \not\subseteq \underline{0}$$

and assume $\Sigma = \{a, b\}$. This constraint can be translated into the corresponding flat form (rule 1 is sufficient in this case), introducing new variables A, B, $\ldots$, F:

$$\begin{array}{l} A \doteq X_1 \cup X_2 \wedge B \doteq X_2 \wedge A \subseteq B \wedge \\ C \doteq X_1 \wedge D \doteq \underline{0} \wedge C \not\subseteq D \wedge \\ E \doteq X_2 \cap X_3 \wedge F \doteq \underline{0} \wedge E \not\subseteq F \end{array}$$

[1]Note that we consider here a definition of flat form which is more general than what is strictly needed. Indeed, when we restrict our attention to $\mathcal{S}_0$-constraints, the cases that deal with function symbols and projections are not required at all.

[2]Given a sequence of terms $t_1, \ldots, t_n$, $vars(t_1, \ldots, t_n)$ denotes the set of variables which occur in at least one of the terms t_i. If φ is a first-order formula, $vars(\varphi)$ denotes the set of free variables in φ.

Then, by applying the translation ϕ of Definition 5.4 we get the SM-constraint:

$$\begin{array}{l} \cup_3(X_1, X_2, A) \wedge B \doteq X_2 \wedge A \subseteq B \wedge \\ C \doteq X_1 \wedge D \doteq \emptyset \wedge C \not\subseteq D \wedge \\ \cap_3(X_2, X_3, E) \wedge F \doteq \emptyset \wedge E \not\subseteq F \wedge \\ X_1 \subseteq U \wedge \ldots \wedge F \subseteq U \wedge U \neq \emptyset \wedge U \doteq \{a, b\} \end{array}$$

This constraint can be solved by the SM-constraint solver. One of its possible solutions is:

$$X_1/\{a\}, X_2/\{a\}, X_3/\{a, b\}$$

plus, for the new variables: $A/\{a\}, B/\{a\}, C/\{a\}, D/\emptyset, E/\{a, b\}, F/\emptyset, U/\{a, b\}$.

Constraints in the $\mathcal{S}_0$ class are not limited to finitely satisfiable constraints as shown by the following example.

Example 5.6 *Consider the $\mathcal{S}_0$-constraint $\underline{1} \subseteq X$ with $\Sigma = \{a, f/1\}$. 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*

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

which is satisfiable in $\mathcal{A}_{\mathcal{SET}}$. The computed answer returned by the SM-constraint solver is: $U \doteq \{A \,|\, B\} \wedge Y \doteq U \wedge X \doteq U$ where A, B are two new variables.

The theorem below states that the satisfiability of a $\mathcal{S}_0$-constraint C can be easily tested in C.S.T.. Moreover, the test is in the NP class and, whenever C is satisfiable, then it is *finitely* satisfiable.

Theorem 5.7 *Let C be a $\mathcal{S}_0$-constraint. C is satisfiable in $\mathcal{A}_{\mathcal{SC}} = \langle \wp(T_\Sigma), (\cdot)^{\mathcal{P}} \rangle$ if and only if $\phi(C)$ is satisfiable in $\mathcal{A}_{\mathcal{SET}} = \langle T_{\Sigma'}/\equiv, (\cdot)^{\mathcal{T}} \rangle$, where $\Sigma' = \Sigma \cup \{\emptyset, \{\cdot \,|\, \cdot\}\}$.*

Proof. If Σ contains only constant symbols, the result is trivial. The solution in $\mathcal{A}_{\mathcal{SC}}$ and that in $\mathcal{A}_{\mathcal{SET}}$ are the same (see also Example 5.5). 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 SM-constraint based only on the (set-theoretic) language consisting of $\emptyset, \{\cdot \,|\, \cdot\}, \doteq, \in, \cup_3, \|$. 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 $\phi(C)$ in $\mathcal{A}_{\mathcal{SET}}$.

It remains to prove that if $\phi(C)$ is satisfiable in $\mathcal{A}_{\mathcal{SET}}$ then C is satisfiable in $\mathcal{A}_{\mathcal{SC}}$. Let σ be a solution of $\phi(C)$ in $\mathcal{A}_{\mathcal{SET}}$, 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

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

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 $\wp(T_\Sigma)$. □

Furthermore, we can prove the following result concerning complexity of the satisfiability check for $\mathcal{S}_0$-constraints.

Proposition 5.8 *Checking satisfiability of a $\mathcal{S}_0$-constraint is NP-complete.*

Proof. The class of constraints which are obtained from $\mathcal{S}_0$-constraints by applying to them the translation ϕ of Definition 5.4 can be solved by NP-algorithms ([9], § 8.1). This ensures that solving $\mathcal{S}_0$-constraints is in NP. It remains to prove that $\mathcal{S}_0$-constraints are NP-hard. The Boolean form of $\mathcal{S}_0$-constraints suggests a reduction of 3-SAT to it. We reason on an instance of 3-SAT, but the method is general. Consider the formula (2) to be tested for satisfiability:

$$(X_1 \vee \neg X_2 \vee X_3) \wedge (\neg X_1 \vee X_2 \vee \neg X_3) \tag{2}$$

Its natural reduction to a $\mathcal{S}_0$-constraint is:

$$\underline{1} \subseteq (X_1 \cup \mathsf{C}(X_2) \cup X_3) \cap (\mathsf{C}(X_1) \cup X_2 \cup \mathsf{C}(X_3)) \tag{3}$$

Now, if σ is a truth assignment for (2), it is immediate to see that σ' defined as: $\sigma'(X) = T_\Sigma$ if $\sigma(X) = \mathsf{true}$, $\sigma'(X) = \emptyset$ otherwise, is a solution of the $\mathcal{S}_0$-constraint (3).

Conversely, let σ be a solution of (3) and let a be an arbitrarily chosen element of T_Σ. Define $\sigma'(X) = \mathsf{true}$ if $a \in \sigma(X)$, $\sigma'(X) = \mathsf{false}$ otherwise. σ' is a successful assignment for the formula (2): assume, by contradiction, that σ' does not fulfill (2). This means that there is a conjunct in (3) in which a does not belong to $\sigma(\ell_1) \cup \sigma(\ell_2) \cup \sigma(\ell_3)$. As a consequence, σ cannot be a solution of (3): a contradiction. □

6 Set-based program analysis using SM-constraints

Set-based constraints were originally introduced in the field of program analysis [26]. Their original intended use is to interpret the solution of a set constraint as the approximation of the set of possible outputs of a program. Set-based constraints can be used to encode specific properties of programs, so that, by checking satisfiability of the constraint, one can check those properties.

Set-based program analysis has been usually carried on in the context of SI-constraints, using a few fixed (decidable) abstractions of the program to be analyzed. However, different abstractions, and different set-based constraint languages associated with them, could be used as well. In this section we prove this claim by considering the analysis of a specific property—the groundness of program variables in logic programs—and showing how it can be abstracted using SM-constraints.

The *groundness analysis* of a logic program P consists in identifying those variables of P which are instantiated to ground terms. Groundness analysis is a central component of logic programming compilers. If we can prove for instance that a variable X is surely bound to ground terms then we can exclude for that variable any possibility of aliasing (i.e., the situation in which X is bound to a term which shares some variable with a term bound to another variable Y). Moreover, a sure groundness information for X allows the occur-check test for X to be useless. The groundness property is, of course, in general undecidable and thus, only an approximation of the analysis can be performed.

Let P be the logic program to be analyzed. Our analysis aims at determining for each argument j of any predicate p in P whether it is surely ground for any successful execution of P. The starting point for our set-based analysis is the notion of *set variable*. For each predicate p in P we associate a set variable V_{pj} with the j-th argument of p and a set variable W_{pij} with the j-th argument of p in the i-th clause defining p.

Moreover, we associate a SM-constraint $\mathcal{V} \doteq e$ with each set variable $\mathcal{V}$ generated for the program P, in the following way:

1. For each set variable W_{pij}, e will be either
   (a) The singleton set $\{c\}$ when W_{pij} is associated with a ground argument of p
   (b) The set $vars(t) = \{X_1, \ldots, X_n\}$ when W_{pij} is associated with a non-ground argument t of p, and $vars(t) \cap \mathsf{Bodyvars} = \emptyset$, where $\mathsf{Bodyvars}$ is the set of variables occurring in the body of the clause considered (the i-th clause defining p)
   (c) The *intersection* of the set variables associated with those atoms in the body of the i-th clause which share some variables with the j-th argument of p, otherwise.
2. For each set variable V_{pj}, instead, e will be the *union* of the set variables associated with the j-th argument in the various clauses defining p.[3]

Finally, the constraint associated with the whole program is obtained by taking the conjunction of all these constraints.[4] Observe that the use of variables instead of constants distinct from c to represent the case of non-ground arguments is intended to allow bindings which are possibly created during constraint solving to be easily propagated throughout the whole constraint.

For the purpose of our analysis, however, we need only to encode the boolean condition "to be ground" (represented by the constant c) or "to be possibly not ground". The latter can be simply represented by the occurrence of an unbound variable X in the set associated with a set variable. Therefore, we can conveniently restrict—by adding further constraints—these sets to have only one of the following forms:

- $\{c\}$
- $\{X\}$, with the constraint that $X \neq c$
- $\{X, c\}$, with the constraint that $X \neq c$

for some variable X.

Furthermore, we need to guarantee that the new bindings for (program) variables possibly created by applying $\cap$ to set variables associated with clause bodies remain local to the clause itself. To this purpose, we assume that a "copy" of the set term possibly bound to set variables associated with atoms in the body of a clause is created whenever it is necessary. This task, however, is strongly simplified by the fact that the considered set terms can have only one of the simple forms mentioned above.

In order to force solutions to be of the desired forms and, simultaneously, to obtain a copy of a set term when needed, we introduce a new predicate, called δ, which is defined as follows:

$$\begin{array}{lll} \delta(s,t,N) & \text{if and only if} & (s \doteq \{c\} \wedge t \doteq \{c\}) \vee \\ & & \exists X (s \doteq \{X\} \wedge t \doteq \{N\} \wedge X \neq c \wedge X \neq N) \vee \\ & & \exists X (s \doteq \{X, c\} \wedge t \doteq \{N, c\} \wedge X \neq c \wedge X \neq N) \end{array}$$

$\delta(A, B, N)$ is true if A is either $\{c\}$, or $\{X\}$, or $\{X, c\}$, and B is the same as A but X is replaced by N, with N a newly introduced variable distinct from X. The presence of

[3]SM-constraints use $\cup_3$ and $\cap_3$ to represent $\cup$ and $\cap$; however, we prefer here to use the standard functional notation for the sake of readability.

[4]We assume, w.l.o.g., that all atoms in clause bodies have the form $q(X_1, \ldots, X_n)$, with $X_1, \ldots, X_n$ variables: if this is not the case, a simple program transformation can be applied to reach this form. For instance, the clause $p(X,Y) :- q(f(Z),X), r(g(Z),Y,a)$ can be rewritten as:
$p(X,Y) :- q(A,X), r(B,Y,C), aux_1(A,Z), aux_2(B,Z), aux_3(C).\ aux_1(f(Z),Z).\ aux_2(g(Z),Z).\ aux_3(a).$

N in δ allows copies of different sets (those concerning the same clause body) to be built using the same new variable. δ is defined as a disjunction of SM-constraints, hence the SM-constraint solver is always able to decide its satisfiability.

The technique for generating the SM-constraint associated with a program P and how to use this constraint for checking groundness of predicates in P are informally introduced by a number of simple examples hereafter.

Example 6.1 *Consider the following Prolog program P*

```
r(a).
r(f(H)).
q(Z,a).
p(X,Y) :- q(X,X),r(Y).
```

The SM-constraint $\Gamma(P)$ associated with P is:

$$\begin{array}{l} W_{r11} \doteq \{c\} \ \wedge \ W_{r21} \doteq \{H\} \wedge \\ W_{q11} \doteq \{Z\} \ \wedge \ W_{q12} \doteq \{c\} \wedge \\ W_{p11} \doteq V'_{q1} \cap V'_{q2} \ \wedge \ \delta(V_{q1}, V'_{q1}, N) \ \wedge \ \delta(V_{q2}, V'_{q2}, N) \ \wedge \ \delta(W_{p11}, W'_{p11}, N) \wedge \\ W_{p12} \doteq V'_{r1} \ \wedge \ \delta(V_{r1}, V'_{r1}, M) \wedge \\ V_{r_1} \doteq W_{r11} \cup W_{r21} \ \wedge \ V_{q1} \doteq W_{q11} \ \wedge \ V_{q2} \doteq W_{q12} \ \wedge \ V_{p_1} \doteq W_{p11} \ \wedge \ V_{p2} \doteq W_{p12} \end{array}$$

Computing the solutions for the SM-constraint associated with a program P allows us to draw some conclusions about the groundness property of the predicates in P. Intuitively, to deduce whether an argument j of a predicate p is ground or not one can proceed as follows: collect *all* possible solutions for the set variable V_{pj}; if the collection contains only (ground) sets of the form $\{c\}$ then we can conclude that the groundness property for argument j of p holds; otherwise nothing can be said about its groundness.

Example 6.2 *Consider the program P of Example 6.1 and the SM-constraint $\Gamma(P)$ associated with it. By applying all possible immediate substitutions, $\Gamma(P)$ becomes:*

$$\begin{array}{l} V_{p1} \doteq V'_{q1} \cap V'_{q2} \ \wedge \ \delta(\{Z\}, V'_{q1}, N) \ \wedge \ \delta(\{c\}, V'_{q2}, N) \ \wedge \ \delta(V_{p1}, W'_{p11}, N) \wedge \\ V_{r_1} \doteq \{c\} \cup \{H\} \ \wedge \ V_{q1} \doteq \{Z\} \ \wedge \ V_{q2} \doteq \{c\} \wedge \\ V_{p_2} \doteq V'_{r1} \ \wedge \ \delta(V_{r1}, V'_{r1}, M) \end{array}$$

A possible computed answer for $\delta(\{Z\}, V'_{q1}, N)$ is $V'_{q1} \doteq \{N\} \wedge Z \doteq c \wedge N \neq Z$ while the only answer for $\delta(\{c\}, V'_{q2}, N)$ is $V'_{q2} \doteq \{c\}$. From these, we get as part of constraint $\Gamma(P)$,

$$V_{p1} \doteq \{N\} \cap \{c\} \ \wedge \ Z \doteq c \wedge N \neq Z \ \wedge \ \delta(V_{p1}, W'_{p11}, N)$$

whose only solution is

$$V_{p1} \doteq \{c\} \ \wedge \ N \doteq c.$$

From the part of $\Gamma(P)$ dealing with V_{r_1} and V_{p_2}

$$V_{r_1} \doteq \{c\} \cup \{H\} \ \wedge \ V_{p_2} \doteq V'_{r1} \ \wedge \ \delta(V_{r1}, V'_{r1}, M)$$

one of the possible computer answer we get is

$$V_{r_1} \doteq \{c, H\} \ \wedge \ H \neq c \ \wedge \ V_{p_2} \doteq \{c, M\} \ \wedge \ M \neq c$$

Thus, one of the computed answers for the variables of interest is

$$V_{p1} \doteq \{c\} \wedge V_{p2} \doteq \{c, M\} \wedge V_{q_1} \doteq \{Z\} \wedge V_{q_2} \doteq \{c\} \wedge V_{r1} \doteq \{c, H\}.$$

All other answers for $\Gamma(P)$ are more specific than this one (in particular, they require that H is instantiated to c). Therefore, we can conclude that the first argument of p and the second of q are surely ground, whereas the others are not ensured to be ground.

Observe that, in general, all computed answers for a given SM-constraint must be considered. In particular, it is important to consider more general answers in order to be able to establish groundness. Furthermore, observe that also creating copies of sets of program variables (predicate δ) is necessary, in general. With reference to Example 6.2, without creating a copy of V_{q_1} and V_{q_2}, variable Z would be instantiated to c and thus also the first argument of p would be (incorrectly) considered surely ground.

The same technique works correctly also for *recursive* clauses, as shown by the following simple example.

Example 6.3 *Consider the recursive Prolog program*

$$\begin{array}{l}\mathtt{nat}(0).\\ \mathtt{nat}(s(X))\ :-\ \mathtt{nat}(X).\end{array}$$

The SM-constraint associated with it is:

$$\begin{array}{l}W_{\mathtt{nat}11} \doteq \{c\} \ \wedge\ W_{\mathtt{nat}21} \doteq V'_{\mathtt{nat}1}\ \wedge \delta(V_{\mathtt{nat}1}, V'_{\mathtt{nat}1}, N) \wedge\\ V_{\mathtt{nat}1} \doteq W_{\mathtt{nat}11} \cup W_{\mathtt{nat}21}\end{array}$$

By applying immediate substitutions we get the constraint

$$V_{\mathtt{nat}1} \doteq \{c\}\ \cup\ V'_{\mathtt{nat}1} \wedge \delta(V_{\mathtt{nat}1}, V'_{\mathtt{nat}1}, N).$$

The (only) computed answer returned by the constraint solver for this constraint is $V_{\mathtt{nat}1} \doteq \{c\}$*. From this answer, it is immediate to conclude (correctly) that* `nat` *is ground. Note that if the first clause of the sample program above is replaced by* $\mathtt{nat}(Z)$ *the computed answer for the corresponding SM-constraint is* $V_{\mathtt{nat}} = \{Z\}$*, indicating that in this case* `nat` *may be not ground.*

6.1 Correctness of the method

Given a program P, we have shown how to associate a formula $\alpha(P)$ involving SM-constraints and atoms based on the predicate δ to the program P. Each occurrence of δ in P can be replaced (through unfolding) by the disjunction of SM-constraints $\gamma_1 \vee \gamma_2 \vee \gamma_3$ defining it (assuming that new instances of X are created for each occurrence of δ). Let $\alpha'(P)$ be the disjunction of SM-constraints obtained from $\alpha(P)$ by applying unfolding over δ and distributivity on the resulting formula. The constraint solver for SM-constraints can be applied to each disjunct of $\alpha'(P)$. We use the notation

$$\mathcal{A}_{\mathcal{SET}} \models \alpha'(P) \rightarrow V_{pj} \doteq \{c\} \tag{4}$$

to represent the property that "at least one of the constraints in solved form returned by the constraint solver contains $V_{pj} \doteq \{c\}$ and all others are either false or contain $V_{pj} \doteq \{c\}$". Correctness of the constraint solving procedure w.r.t. the model $\mathcal{A}_{\mathcal{SET}}$ (c.f. Section 3) ensures that the model-theoretic reading of formula (4) corresponds exactly to the property at hand.

Theorem 6.4 *If formula* (4) *holds for some predicate* p *and integer* j*, then for all successful SLD-derivations of* $\leftarrow p(X_1, \ldots, X_j, \ldots, X_{ar(p)})$*, if* σ *is the computed answer substitution, then* $\sigma(X_j)$ *is ground.*

Proof. By contradiction, let k be the minimum integer s.t. there exist p and j s.t. i) there is a successful SLD-derivation of $\leftarrow p(X_1,\ldots,X_j,\ldots,X_{ar(p)})$ of length k, ii) $\sigma(X_j)$ is not ground, and iii) formula (4) holds for V_{pj}.

If $k=1$ this means that a unit clause $p(t_1,\ldots,t_{ar(p)})$ belongs to P and t_j is not ground. Let us assume that this is the i-th clause defining p. This means that the constraint $W_{pij} \doteq vars(t_j)$ belongs to each disjunct of $\alpha'(P)$. The variable W_{pij} occurs in exactly one equation of the form $V_{pj} \doteq W_{p11} \cup \cdots \cup W_{pij} \cup \cdots \cup W_{phar(p)}$, with h the number of clauses defining p in P. The variable V_{pj} can occur in the r.h.s. of other equations. If this is not the case, then it is immediate to see that it is impossible for W_{pij} to be forced to $\{c\}$ in all solutions and, consequently, this holds also for V_{pj}. Otherwise, the unfolding of the predicates δ forces V_{pj} (and hence, W_{pij}) in some disjunct to be of the form $\{c\}$, $\{X\}$, or $\{X,c\}$ for some variable X. Since the variable W_{pij} does not depend on any other variable, by constructing a dependency graph among variables, one can notice that if a solution holds in the first disjunct ($V_{pj} \doteq \{c\}$), also a solution in the third disjunct ($V_{pj} \doteq \{X,c\}$), is admitted (the extra variable can be added to each variable reachable from W_{pij}). This is a contradiction with assumption (4).

If $k>1$ this means that a clause C of the form $p(t_1,\ldots,t_{ar(p)}) \leftarrow q_1(\bar{Z}_1),\ldots,q_m(\bar{Z}_m)$ belongs to P, and it is used in the first step of the SLD derivation. Let C be the i-th clause defining p. If $vars(t_j) \cap \mathsf{Bodyvars} = \emptyset$ we are in the same situation as the previous case. Otherwise, $vars(t_j) \cap \bar{Z}_\ell \neq \emptyset$ for at least one ℓ in $\{1,\ldots,m\}$. For the sake of simplicity, assume that the intersection is not empty exactly for the first argument of q_1 and the second argument of q_2. This means that $W_{pij} \doteq V'_{q_1 1} \cap V'_{q_2 2}$ is in $\alpha'(P)$.

Since $V_{pj} \doteq \{c\}$ in all solutions, this fact holds also for $W_{pij}, V'_{q_1 1}$, and $V'_{q_2 2}$. Thus, there is another predicate (q_1) and j (1) s.t. (4) holds and, using the lifting lemma, there exists a SLD derivation for it, returning a non-ground value for the j-th argument in less than k steps. A contradiction with the minimality of k. □

The set-based analysis described in this section can be completely implemented using SM-constraints and its constraint solver. A number of working examples of SM-constraints generated according to this analysis technique, running on the CLP language with sets CLP($\mathcal{SET}$), can be found at www.math.unipr.it/~gianfr/SETLOG/APPLICATIONS/analysis.html

7 Examples

In this section we present a number of simple examples to show the expressivity of SM-constraints. We will also show how using two slightly different definitions of SM-constraints for encoding directly finite automata and temporal logic problems.

7.1 Encoding Problems into SM-constraints

3-SAT. Consider the instance of 3-SAT:

$$(X_1 \vee \neg X_2 \vee X_3) \wedge (\neg X_1 \vee \neg X_2 \vee X_3) \wedge (X_1 \vee X_2 \vee \neg X_3)$$

It can be expressed using the following SM-constraint:

$$\begin{array}{l} \{\{X_1,Y_1\},\{X_2,Y_2\},\{X_3,Y_3\}, \\ \quad \{X_1,Y_2,X_3,0\},\{Y_1,Y_2,X_3,0\},\{X_1,X_2,Y_3,0\}\} \quad \doteq \quad \{\{0,1\}\} \end{array}$$

where the variables Y_i take the place of $\neg X_i$. Each solution for this equation is also a solution for the given 3-SAT problem and, vice versa, from each solution for the SAT problem it is immediate to assign values to the variables Y_i and R_i in order to obtain a solution for the equation. One such solution is, for instance:

$$X_1/1, X_2/1, X_3/1$$

(and $Y_1/0, Y_2/0, Y_3/0$).

Map Coloring. Consider the following instance of the well-known problem of the coloring of a geographical map. Assume that we wish to use the four colors `white`, `yellow`, `red`, and `blue` to color the countries `Austria`, `France`, `Italy`, `Slovenia`, and `Swiss`. This problem can be rendered by the following SM-constraint

$$\begin{array}{ll} \{ & \{\texttt{Italy}, \texttt{Austria}\}, \{\texttt{Italy}, \texttt{France}\}, \{\texttt{Italy}, \texttt{Slovenia}\}, \\ & \{\texttt{Italy}, \texttt{Swiss}\}, \{\texttt{Austria}, \texttt{Swiss}\}, \{\texttt{Austria}, \texttt{Slovenia}\}, \{\texttt{France}, \texttt{Swiss}\} \,|\, \mathsf{R}\} \quad \doteq \\ \{ & \{\texttt{white}, \texttt{yellow}\}, \{\texttt{white}, \texttt{red}\}, \{\texttt{white}, \texttt{blue}\}, \\ & \{\texttt{yellow}, \texttt{red}\}, \{\texttt{yellow}, \texttt{blue}\}, \{\texttt{red}, \texttt{blue}\}\} \end{array}$$

where names with capital initial are variables and all other names are constants. Observe that the right-hand side of the equation is a set containing all possible unordered pairs of colors which can be formed using the given four colors. The variable R in the left-hand side is introduced for allowing solutions using less than four colors. One of the (many) solutions for this constraint is the one given by the solved form constraint:

$$\begin{array}{c} \texttt{Italy} \doteq \texttt{white} \ \wedge \ \texttt{Swiss} \doteq \texttt{red} \ \wedge \ \texttt{France} \doteq \texttt{yellow} \ \wedge \\ \texttt{Austria} \doteq \texttt{yellow} \ \wedge \ \texttt{Slovenia} \doteq \texttt{red} \ \wedge \ \mathtt{R} \doteq \emptyset \end{array}$$

Observe that, in general, equality constraints (hence, set unification [16]) can be conveniently exploited in those problems—e.g., resource allocation problems—whose solution can be naturally expressed as a *generate & test* scheme. Solving the set equations allows one to non-deterministically generate all possible combinations of values for the given problem, and set constraints will select only those combinations which are satisfactory.

7.2 Other Examples

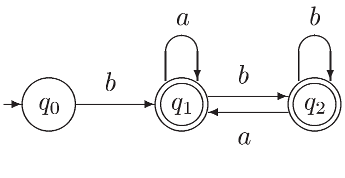
In the first two examples of this section (drawn from [13]) we will make use of constraints on non-well founded sets (or hypersets).

Automata equivalence. A *deterministic finite automaton* (DFA for short) consists of a set $Q = \{Q_0, \ldots, Q_n\}$ of *states*, a set $S = \{s_1, \ldots, s_k\}$ of *symbols*, and a *transition function* $d : Q \times S \to Q \cup \{\perp\}$. One of the states—say Q_0—is called *initial state*, and there is a set $F \subseteq Q$ of *accepting states*.

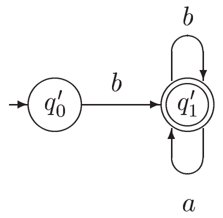
Given a DFA A, one may define a SM-constraint in the signature $\Sigma = \{\emptyset, \{\cdot \,|\, \cdot\}, \perp, \delta, \delta'\}$, where $\perp$ is a constant symbol and δ, δ' are function symbols of arity k, as follows:

$$\begin{array}{rcl} \mathcal{E}^A & = & A \doteq \{Q_0, \ldots, Q_n\} \wedge \\ & & \bigwedge_{Q_i \in Q \setminus F} Q_i \doteq \delta(d(Q_i, s_1), \ldots, d(Q_i, s_k)) \wedge \\ & & \bigwedge_{Q_i \in F} Q_i \doteq \delta'(d(Q_i, s_1), \ldots, d(Q_i, s_k)) \end{array}$$

Given two DFAs A and B, it is easy to determine whether or not they accept the same language, as shown by the following simple example. Consider the two DFAs



DFA A



DFA B

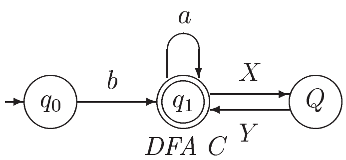
where the sets of accepting states are $F_A = \{q_1, q_2\}$ and $F_B = \{q'_1\}$, respectively. These automata can be modeled by the following two constraints

$$\begin{aligned}
\mathcal{E}^A &= A \doteq \{Q_0, Q_1, Q_2\} \wedge \\
&\quad\; Q_0 \doteq \delta(\bot, Q_1) \wedge Q_1 \doteq \delta'(Q_1, Q_2) \wedge Q_2 \doteq \delta'(Q_1, Q_2)\ , \\
\mathcal{E}^B &= B \doteq \{Q'_0, Q'_1\} \wedge \\
&\quad\; Q'_0 \doteq \delta(\bot, Q'_1) \wedge Q'_1 \doteq \delta'(Q'_1, Q'_1)\ .
\end{aligned}$$

The conjunction of these constraint is satisfiable, and one of its solutions is given by the solved form constraint:

$$Q_1 \doteq Q_2 \wedge Q'_1 \doteq Q_2 \wedge \ldots$$

Automata matching. The problem considered in the above example could be solved with more efficient algorithms (classical $n \log n$ Hopcroft's result). However, notice that the actual question we can answer using the encoding based on SM-constraints is that of determining whether two *partially defined* DFAs can be completed in order to represent the same automaton. Hence, this problem stands to the latter as ground comparison stands to matching. As an example, one may be interested in knowing whether the following partially defined automaton



DFA C

where X, Y, and Q are unknowns, can be completed so as to accept the same language accepted by *DFA B* defined above. This problem is in fact NP-complete (cf. [13]).

Temporal logic problems. The $\Box$-as-$\wp$ translation [12, 9] establishes a connection between modal validity (satisfiability) and validity (satisfiability) over a particular set theory, called Ω (to be precisely defined in the sequel). A complete presentation of the results about the $\Box$-as-$\wp$ translation is outside the scope of this example: our intent here is simply to provide an intuitive idea of another possible application of set-based constraints in a completely different context.

Let $P = \{p_1, p_2, p_3, \ldots\}$ be an infinite set of propositional literals. A *modal formula* γ is defined as follows:

$$\gamma \quad := \quad p_i \mid \gamma_1 \vee \gamma_2 \mid \neg\gamma_1 \mid \Box\gamma_1$$

The semantics of Modal Logic is defined on *Kripke structures* [20].

The key idea of the $\Box$-as-$\wp$ set-theoretic translation is simply to replace the accessibility relation R of Kripke semantics by the membership relation $\in$. This consideration leads to the following translation from a modal formula γ to a set-theoretic formula γ^s. Let

$p_1, \ldots, p_n$ be propositional literals and $\gamma(p_1, \ldots, p_n)$ be a modal formula; the set term $\gamma^*(x, z_1, \ldots, z_n)$, where $x, z_1, \ldots, z_n$ are variables, is inductively defined as follows:

$$\begin{array}{rcl} p_i^* & = & z_i \\ (\gamma_1 \vee \gamma_2)^* & = & \gamma_1^* \cup \gamma_2^* \\ (\neg\gamma)^* & = & x \setminus \gamma^* \\ (\Box\gamma)^* & = & \wp(\gamma^*). \end{array}$$

Observe that the set language uses only the symbols $\cup$ (union), $\setminus$ (set difference), and $\wp$ (powerset). We can also use set inclusion that however is replaced by a $\cup$-formula ($s \subseteq t \leftrightarrow t = s \cup t$). The *set-theoretic translation* $\gamma^s(x)$ of the modal formula $\gamma(p_1, \ldots, p_n)$ is the set-theoretic formula

$$\forall z_1, \ldots, z_n (x \subseteq \gamma^*(x, z_1, \ldots, z_n)).$$

In order to establish a link between the derivability of a formula γ in modal logic and the first-order derivability of its translation γ^s it is necessary to single out the behaviour of membership and of the other considered set operators. They are precisely characterized by the axioms of the set theory Ω presented here below.

$(\cup)$	$\forall x, y, z\, (x \in y \cup z \leftrightarrow x \in y \vee x \in z)$
$(\setminus)$	$\forall x, y, z\, (x \in y \setminus z \leftrightarrow x \in y \wedge x \notin z)$
$(\subseteq)$	$\forall x, y\, (x \subseteq y \leftrightarrow \forall z\, (z \in x \rightarrow z \in y))$
$(\wp)$	$\forall x, y\, (x \in \wp(y) \leftrightarrow x \subseteq y)$

In [12] the following result has been proved: let Λ be a modal logic with a finite axiomatization α in the propositional literals $q_1, \ldots, q_m$ and let γ be a modal formula in the propositional literals $p_1, \ldots, p_n$, with all the q_i's different from the p_j's. It holds that

$$\vdash_\Lambda \gamma \quad \Rightarrow \quad \Omega \vdash \forall x(x \subseteq \wp(x) \wedge \alpha^s \rightarrow \gamma^s)).$$

Moreover

$$\Lambda \Vdash \gamma \quad \Leftarrow \quad \Omega \vdash \forall x(x \subseteq \wp(x) \wedge \alpha^s \rightarrow \gamma^s)).$$

Hence, for instance, in order to prove that the modal formula $\Box p_1 \rightarrow \Box(p_1 \vee p_2)$ is valid in the minimal modal logic K we could equivalently prove that

$$\Omega \vdash \forall x, z_1, z_2 (x \subseteq \wp(x) \rightarrow x \subseteq (x \setminus \wp(z_1)) \cup \wp(z_1 \cup z_2)).$$

8 CLP with Sets

The Constraint Logic Programming (CLP) scheme [27] defines a class of languages, $CLP(\mathcal{X})$, which are parametric in the constraint domain $\mathcal{X}$. $\mathcal{SC}$ and $\mathcal{SET}$ are two such domains, both concerned with (different forms of) sets. Therefore, specific CLP languages dealing with sets can be obtained by simply considering the instances of the general CLP scheme based on these constraint domains: $CLP(\mathcal{SC})$ is the instance based on $\mathcal{SC}$, CLP($\mathcal{SET}$) that based on $\mathcal{SET}$. CLPS [32] and Conjunto [23] are other two examples of CLP languages with sets based on quite different notions of set constraints.

Being instances of the general CLP scheme, $CLP(\mathcal{SC})$ and CLP($\mathcal{SET}$) inherit from this scheme all its general features. In particular, the syntactic form of a program, as well as the semantics of the user defined predicates are those described in the general scheme [27, 28]. Constraints can be used in these languages as primitive operations for building programs in order to enhance expressiveness and/or to control the pruning of the search tree during the execution.

In $CLP(\mathcal{SC})$ [29, 21] the admissible constraints are the SI-constraints: constraint formulae are based on the inclusion symbol, and the interpretation structure is that of the rational solutions of SI-constraints, i.e., $\mathcal{A}_{\mathcal{SC}}$. The constraint solving algorithms used within $CLP(\mathcal{SC})$ are drawn from the decidability results for SI-constraints mentioned in the previous sections.

In $CLP(\mathcal{SET})$ the admissible constraints, the interpretation structure, and the constraint solver are those briefly described for SM-constraints in the previous sections. An interpreter of the CLP($\mathcal{SET}$) language—implemented in SICStus Prolog—is available at http://www.math.unipr.it/~gianfr/setlog.Home.html. CLP($\mathcal{SET}$) has been designed mainly as a general-purpose programming language for declarative programming. By exploiting its very flexible and general usage of sets, integrated with the computational capabilities inherited from the general CLP scheme, it is possible to define in CLP($\mathcal{SET}$) other powerful control and data abstractions such as *Restricted Universal Quantifiers* (RUQ) and *intensionally defined sets*:

- RUQs 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 $\forall X((X \in s) \to G)$.

- *intensionally defined sets* are sets defined by stating a property that must hold for any element X to belong to the set itself:

$$\{\, X \,:\, \varphi[X] \,\}\,.$$

With a slight abuse of terminology, we indicate SM-constraints augmented with RUQ's and intensionally defined sets as *CLP($\mathcal{SET}$)$^+$-constraints*. The CLP($\mathcal{SET}$) interpreter is capable to deal with CLP($\mathcal{SET}$)$^+$-constraints. In [18] we have shown that using CLP($\mathcal{SET}$) we can further narrow the gap between the two set-based constraint languages considered in this paper. In fact, it is possible to express *any* SI-constraint in terms of CLP($\mathcal{SET}$)$^+$-constraints. The problem with this translation, however, is that checking the satisfiability of CLP($\mathcal{SET}$)$^+$-constraints is not decidable in the general case, due to the presence of RUQ's atoms (either explicit or "hidden" in an intensional set definition). Indeed, whenever the input constraint admits solutions that involve infinite sets, a satisfaction procedure based on enumeration—like that of CLP($\mathcal{SET}$)—does not terminate.

Example 8.1 *The CLP($\mathcal{SET}$)$^+$-constraint*

$$\emptyset \in X \wedge (\forall Y \in X)(\{Y \,|\, Y\} \in X)$$

admits only solutions on the domain of infinite sets. In particular, its minimal solution is the set ω of all the numerals à la *Von Neumann, i.e., $\{\emptyset, \{\emptyset\}, \{\{\emptyset\}\}, \ldots\}$. Giving this constraint as input to CLP($\mathcal{SET}$) the CLP($\mathcal{SET}$) interpreter will loop forever while trying to collect all elements of the infinite set.*

In [18], however, we show also that the CLP($\mathcal{SET}$) interpreter can be adapted to decide satisfiability in a domain including infinite sets of some sub-class of SI-constraints which is a superset of the class of SI-constraints which are directly representable as SM-constraints. Precisely, we show that it is possible to find a bound on the number of the computation steps of the CLP($\mathcal{SET}$) interpreter that, if reached, ensures infinite satisfiability of the SI-constraint.

As a final remark, the availability of CLP languages embedding sets allows, in general, a remarkable enlargement of the possible applications of set-based constraints. Though CLP languages with sets can be used as *general-purpose* languages, there are certain application areas in which the use of sets fits more naturally. These areas include database applications (see for instance [31]), combinatorial problems, graph-related applications and operational research in general (e.g., resource allocation problems), as pointed out for instance in [23, 32, 10]. A number of simple examples in some of these areas using CLP($\mathcal{SET}$) can be found in [16].

9 Conclusions

We have presented two different kinds of set-based constraints: SI-constraints (based on set inclusion and using the powerset of T_Σ as its domain), and SM-constraints (based on membership and union and on the domain of hereditarily finite sets of Herbrand terms). We have pointed out differences and analogies between them and identified a class of set-based constraints (class $\mathcal{S}_0$) for which both proposals can be used to decide satisfiability in the relevant structures. We have shown that both proposals (focusing, in particular, on SM-constraints) can be used for applications in program analysis. Moreover, we have briefly presented some other possible examples of applications. We have also addressed the embedding of these constraint domains into a CLP framework, and, in particular, we have briefly described the CLP language CLP($\mathcal{SET}$). Though the discussion is specifically devoted to SI- and SM-constraints, most of the considerations we have done are general enough to be easily adaptable to other CLP languages endowed with sets (e.g., CLPS [32]).

As future work one could try to characterize other classes of decidable set-based constraints and possibly use them to define new more general notions of set-based constraints. With this respect, possible—either alternative or complementary—future directions of work are:

- enlarging the class of SM-constraints by moving to the domain of rational trees and hypersets [14];
- considering also other forms of aggregates, such as multisets and lists, and the relevant basic constraints, by exploiting the results in [17];
- using the results in [7] concerning *intensional set constraints* to single out (other) cases for which we can decide satisfiability using an extended CLP($\mathcal{SET}$) interpreter, even in presence of infinite sets;
- exploiting as much as possible the results obtained in the Computable Set Theory area to devise other, possible wider, classes of decidable set-based formulae.

The availability of these kinds of set-based constraints would make it feasible, among others, to explore new solutions for set-based program analysis, which allow to exploit the new facilities (e.g., multisets) to define more precise abstractions for some specific properties, for both logic and imperative programs.

Acknowledgments

The authors wish to thank Adriana Morini for useful discussions on the topics of this paper and for her essential contribution to the development of the set-based program analysis technique described in Section 6. This work is partially supported by MURST project *Ragionamento su aggregati e numeri a supporto della programmazione e relative verifiche.*

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. Introduction to Set Constraint-Based Program Analysis. *Science of Computer Programming* 35(2):79–111, 1999.

[3] A. Aiken, D. Kozen, M. Y. Vardi, and E. L. Wimmers. The Complexity of Set Constraints. In E. Börger, Y. Gurevich, and K. Meinke eds., *CSL '93*, LNCS, Vol. 832, Page 1–17, Springer-Verlag 1994.

[4] A. Aiken, D. Kozen, and E. Wimmers. Decidability of systems of set constraints with negative constraints. *Information and Computation* 122(1): 30-44 (1995).

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

[6] J. P. Banatre and D. Le Metayer. Programming by Multiset Transformation. *Communications of the ACM 36,* 1 (Jan.), 98–111, 1993.

[7] R. Carmona, A. Dovier, and G. Rossi. Dealing with Infinite Intensional Sets in CLP. In M. Falaschi, M. Navarro, and A. Policriti, editors, *APPIA-GULP-PRODE'97. Joint Conf. on Declarative Programming*, pages 467–477. Grado, Italy, June 1997 (available at www.math.unipr.it/~gianfr/PAPERS/AGP.CDR97.ps).

[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] D. Cantone, E. G. Omodeo, and A. Policriti. *Set Theory for Computing.* Monographs in Computer Science. Springer Verlag, 2001.

[10] Y. Caseau and F. Laburthe. Introduction to the CLAIRE Programming Language. Technical report, LIENS, 1996.

[11] 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.

[12] G. D'Agostino, A. Policriti, and A. Montanari. A set-theoretic translation method for polymodal logics. *Journal of Automated Reasoning*, 15(3):317–337,1995.

[13] A. Dovier. *Computable Set Theory and Logic Programming.* PhD thesis, Università degli Studi di Pisa, March 1996. TD–1/96.

[14] A. Dovier, E. Omodeo, and A. Policriti. Solvable set/hyperset contexts: II. A goal-driven unification algorithm for the blended case. *Applicable Algebra in Engineering, Communication and Computing*, 9(4):293–332, Springer-Verlag, 1999.

[15] 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.

[16] A. Dovier, C. Piazza, E. Pontelli, and G. Rossi. Sets and constraint logic programming. *ACM Transaction on Programming Language and Systems* (TOPLAS), 22(5) 2000, pp. 861–931.

[17] A. Dovier, C. Piazza, and G. Rossi. A uniform approach to constraint-solving for lists, multisets, compact lists, and sets. Technical Report, Dipartimento di Matematica, Università di Parma, Quaderno di Matematica n.235, Sept. 2000 (available at www.cs.unipr.it/Publications/Abstracts/Q235).

[18] A. Dovier, C. Piazza, and G. Rossi. Relating Set Constraints and Constraint Logic Programming with Finite Sets. Technical Report, Dipartimento di Matematica, Università di Parma, Quaderno di Matematica n.196, May 1999 (available at www.cs.unipr.it/Publications/Abstracts/Q196).

[19] 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.

[20] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, vol. B: Formal Models and Semantics. Elsevier, and MIT Press, 1990.

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

[22] F. Gecseg and M. Steinby. *Tree Automata.* Akadmiai Kiad, Budapest, 1984.

[23] C. Gervet. Interval Propagation to Reason about Sets : Definition and Implementation of a Practical Language. *International Journal of Constraints* 1, 191–246, 1997.

[24] 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.

[25] 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.

[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] J. Jaffar, M. J. Maher, K. Marriott, and P. J. Stuckey. The Semantics of Constraint Logic Programs. *Journal of Logic Programming 37,* 1–3, 1–46, 1998.

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

[30] K. Kunen. *Set Theory. An Introduction to Independence Proofs.* Studies in Logic. North Holland, Amsterdam, 1980.

[31] G. M. Kuper. Logic Programming with Sets. *Journal of Computer and System Science 41*, 1 (1990), 66–75.

[32] 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.

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