Dealing with Infinite Intensional Sets in CLP

Roberta Carmona[♣] Agostino Dovier[♡] Gianfranco Rossi[♣]

Abstract

Very often a set S is denoted *intensionally*, that is by providing a condition φ that is necessary and sufficient for an element X to belong to S. This paper addresses the problem of embedding intensional sets into a CLP language which offers extensional sets and a few basic operations on them (namely, $=$, $\in$, and their negative counterparts). We consider the translation-based technique used in $\{log\}$ and we show that it can be extended so as to allow also *infinite sets* to be dealt with in a number of common situations. This is obtained by providing program rewriting rules that eliminate intensional sets when occurring as arguments of the basic set-theoretic operations. The rewriting process is shown to be correct, complete, and always terminating. A number of possible future developments are also pointed out.

1 Introduction

A number of different proposals have been put forward in the area of the integration between logic-based programming paradigms and set theory (e.g., [1, 5, 8, 11, 12]). Most of them, however, limit their expressivity to *extensional sets*. In the practice of mathematics, only seldom a set S is denoted *extensionally*, that is by enumeration of its elements. Much more often, one provides a condition φ that is necessary and sufficient for an element X to belong to S. This *intensional* denotation of a set is achieved by use of an *intensional set term*, whose typical syntactic form is $\{ X : \varphi[X] \}$. Intuitively, $\{ X : \varphi \}$ means "the set of all instances of X corresponding to the instances of φ which are true".

Conventional Prolog systems provide some facilities for intensional set definition in the form of the `setof` built-in predicate, which, however, completely lacks of a precise

[♣]Univ. di Parma, Dip. di Matematica. gianfr@prmat.math.unipr.it

[♡]Univ. di Udine, Dip. di Matematica e Informatica. dovier@dimi.uniud.it

semantical characterization (an attempt to provide such a characterization can be found in [4]). Intensional sets are supplied also by Gödel, still, however, with little attention to semantics. Cleaner characterizations of intensional sets are provided by $\mathcal{LDL}$ (cf., e.g., [1]), *Subset-assertion Programming* (cf., e.g., [10]), and $\{log\}$ [5], all of them using different approaches for embedding intensional sets in the language.

However, all these proposals do not consider the case that intensional set formers represent *infinite* sets, though they do not exclude the possibility that this can happen. Some hints on how to deal with infinite intensional sets can be found in [14].

In this paper we show how the translation-based technique used in $\{log\}$—a CLP language which offers extensional sets and a few basic operations on them—can be extended so as to allow intensionally denoted infinite sets to be dealt with in a number of common situations. This is obtained by providing a set of program rewriting rules that eliminate intensional sets when occurring as arguments of the basic set-theoretic operations by generating equivalent new programs without intensional sets. The proposed rewriting rules are general enough to be exploitable also in a wider context than the one considered in this paper. They also make evident the strong connection between intensional set handling and the ability to deal with negation in goals and clause bodies.

The paper is organized as follows. Sec. 2 presents the basic approaches to set grouping in the context of logic programming languages. Sec. 3 provides a brief review of the (CLP) language $\{log\}$ (read 'setlog'). In Sec. 4 we define our program rewriting rules for removing intensional set terms from $\{log\}$ programs and prove correctness and completeness of the rewriting process. The connection between intensional sets and negation is briefly discussed in Sec. 5. Some lines for future work are sketched in Sec. 6.

2 Set-grouping

The introduction of intensional sets of the form $\{X : \varphi[X]\}$ is based on the availability of a *set-grouping* mechanism [15] capable of collecting in a set S all the instances of X satisfying the formula φ. There are, at least, two alternative approaches for providing set-grouping.

Set-grouping can be a *built-in* feature of the language as in [1]. This implies, among others, that one must be able to provide both a declarative and a procedural semantics for the language with set-grouping. And this may be a difficult task, as shown in [15, 1], where a bottom-up procedural semantics for the language is presented. As a consequence, a number of syntactic restrictions (akin to those usually required for negation as failure) are imposed to programs with intensional sets, to ensure the existence of a minimum model.

Alternatively, set-grouping can be programmed in the language itself, provided the language has sufficient (extensional) set manipulation facilities and negation is allowed to occur in goals and clause bodies. Basically, the definition of set-grouping can exploit the following intended semantics of intensional sets:

$$\begin{aligned} \{X : \varphi[X, \bar{Y}]\} = S \quad &\leftrightarrow \quad \forall X (X \in S \rightarrow \varphi[X, \bar{Y}]) \wedge \forall X (\varphi[X, \bar{Y}] \rightarrow X \in S) \\ &\leftrightarrow \quad \forall X (X \in S \rightarrow \varphi[X, \bar{Y}]) \wedge \neg \exists X (X \notin S \wedge \varphi[X, \bar{Y}]). \end{aligned} \tag{1}$$

where $\bar{Y}$ denotes a possible empty list of variables, $Y_1, \ldots, Y_n$, which are free in φ. Therefore, according to this approach, intensional set terms are just a syntactic extension of the language.

This is the approach adopted in $\{log\}$. A $\{log\}$ program with intensional sets is translated (via simple preprocessing) into a new $\{log\}$ program with negation, without intensional sets, which is able to implement the required set-grouping facilities. The $\{log\}$ language without intensional sets—called $CLP(\mathcal{SET})$ in previous works and hereafter—has been proved to be an instance of the general CLP scheme ([7]) and, hence, it inherits all its semantics properties. The semantics (both declarative and procedural) of a $\{log\}$ program with intensional sets, therefore, is that of the corresponding $CLP(\mathcal{SET})$ program with negation.

3 An Overview of $CLP(\mathcal{SET})$ and $\{log\}$

$CLP(\mathcal{SET})$ [7] is an instance of the general CLP scheme ([9]) which supplies extensional set terms and a few basic set-theoretical operations. In this section, we briefly introduce the basic notions and notations used in $CLP(\mathcal{SET})$ and $\{log\}$.

Sets are represented using the functional symbol $\{\cdot \mid \cdot\}$ as the set constructor, and the constant $\emptyset$ to denote the empty set. Intuitively, $\{t \mid s\}$ denotes $\{t\} \cup s$.[1]

[1] Observe the difference between this functional symbol and the intensional-set constructor symbol $\{\cdot : \cdot\}$, whose intuitive semantics is that of formula (1).

A *set term* is defined recursively as follows: $\emptyset$ is a set term; a variable X is a set term; $\{t \mid s\}$, where t and s are terms, is a set term. Hereafter, we assume to represent $\{t_0 \mid \{t_1 \mid \cdots \{t_n \mid X\} \cdots\}\}$ as $\{t_0, \ldots, t_n \mid s\}$, and, when s is $\emptyset$, simply as $\{t_0, \ldots, t_n\}$.[2]

The *primitive constraints* in $CLP(\mathcal{SET})$ are the (positive and negative) literals based on the predicate symbols $=$ and $\in$; a *constraint* is any conjunction of primitive constraints. The constraint domain $\mathcal{SET}$ is the domain of *hereditarily finite hybrid sets*, that is sets whose elements are uninterpreted Herbrand terms as well as other (finite) hybrid sets. Such a domain has been also characterized via a simple axiomatic first-order set theory, named *Set* [5]. Basically, *Set* includes, besides the standard equality axioms, the axioms for the existence of the empty set, the axioms for the element insertion operation, the Clark Equality Theory axioms for the non-set terms, and a weak form of the foundation axiom. Moreover, a suitable version of the extensionality axiom is adopted: $(E)\quad \forall xy\,((\mathtt{ker}(x) = \mathtt{ker}(y) \wedge \forall z\,(z \in x \leftrightarrow z \in y)) \rightarrow x = y)$. Intuitively, `ker` returns what remains (i.e., the kernel) of an object when all its elements have been removed. For instance, $\mathtt{ker}(\{a, b\}) = \mathtt{ker}(\{a, b \mid \emptyset\}) = \emptyset$; $\mathtt{ker}(\{a, b \mid b\}) = b$; $\mathtt{ker}(f(a)) = f(a)$.

In order to check satisfiability of a given constraint c, c is transformed to an equivalent disjunction of constraints in solved form, which is proved to be always $\mathcal{SET}$-satisfiable.

A constraint c is in *solved form* if it is either `false` or a conjunction of primitive constraints of one of the following forms:[3]

- $X = t$, X does not occur neither in t nor in the rest of c;
- $X \neq t$ or $t \notin X$, and X does not occur in t.

The constraint satisfiability test is performed by a procedure (named SAT) which non-deterministically computes, for a given constraint c, a finite collection of constraints in solved form $\{c'_1, \ldots, c'_n\}$ such that

$$Set \vdash c \leftrightarrow \tilde{\exists} \bigvee_{c'_i \text{ is returned by } SAT(c)} c'_i$$

Therefore, a constraint c is $\mathcal{SET}$-satisfiable if and only $SAT(c)$ returns at least one constraint in solved form other than `false`.

[2]When s is *not* a set term, $\{t \mid s\}$ is intended to designate a so called *colored set*, that is a set based on a *kernel* other than $\emptyset$ ([7, 5]).

[3]Observe that constraints of the form $s \in t$ are always eliminated by replacing them by equality constraints. In particular, the equivalence $t \in X \leftrightarrow \exists N(X = \{t \mid N\})$ can be proved to hold in *Set*.

From $CLP(\mathcal{SET})$ to $\{log\}$

$\{log\}$ extends $CLP(\mathcal{SET})$ by providing special syntax for Restricted Universal Quantifiers and intensional sets.

Restricted Universal Quantifiers (RUQs) are formulas of the form $(\forall X \in s)\, G$, with G an arbitrary $\{log\}$ goal. This form stands for the quantified implication $\forall X((X \in s) \rightarrow G)$. For example, the following is a $\{log\}$ clause defining a predicate `subset` that tests whether `S1` is a subset of `S2`

$\mathtt{subset(S1,S2)} \mathrel{:-} (\forall \mathtt{X} \in \mathtt{S1})(\mathtt{X} \in \mathtt{S2}).$

RUQs are programmable directly in $CLP(\mathcal{SET})$. A procedure that transforms $\{log\}$ clauses with RUQs to equivalent $CLP(\mathcal{SET})$-clauses without RUQs is described in detail in [5]. As an example, the $\{log\}$ clause for `subset` is transformed by this procedure to the equivalent $CLP(\mathcal{SET})$ clauses:

$\mathtt{subset}(\emptyset, \mathtt{S2}).$

$\mathtt{subset}(\{\mathtt{A} \mid \mathtt{R}\}, \mathtt{S2}) \mathrel{:-} \mathtt{A} \notin \mathtt{R}, \mathtt{A} \in \mathtt{S2}, \mathtt{subset}(\mathtt{R}, \mathtt{S2}).$

Intensional set terms are terms of the form $\{X : G[X]\}$, where X is a variable and G is an arbitrary $\{log\}$ goal containing X. Note that the scope of X, in the clause containing the intensional set term, is the term itself.

Intensional set terms are always translated into the corresponding $CLP(\mathcal{SET})$ clauses with negation that are necessary to implement set-grouping, according to the intended semantics expressed by formula (1). As a matter of fact, the right-hand part of this formula can be rendered in $CLP(\mathcal{SET})$ by the following two clauses:

$\mathtt{setof\_}\varphi(\mathtt{S}, \bar{\mathtt{Y}}) \mathrel{:-} (\forall \mathtt{X} \in \mathtt{S})\varphi, \mathtt{not}\ \mathtt{partial\_}\varphi(\mathtt{S}, \bar{\mathtt{Y}}).$

$\mathtt{partial\_}\varphi(\mathtt{S}, \bar{\mathtt{Y}}) \mathrel{:-} \mathtt{Z} \notin \mathtt{S}, \varphi[\mathtt{X}/\mathtt{Z}].$

where $\mathtt{setof\_}\varphi$ and $\mathtt{partial\_}\varphi$ are new predicate symbols, and the RUQ can be eliminated as shown above. The $\mathtt{partial\_}\varphi$ predicate is intended to reject any partial collection of values satisfying the property φ. As an example, the following $\{log\}$ clause

$\mathtt{powerset(S,P)} \mathrel{:-} \mathtt{P} = \{\mathtt{X} : \mathtt{subset(X,S)}\}.$

can be used to construct the power set of a set `S`.

Therefore, intensional sets in $\{log\}$ exist at the syntactic level only, without affecting the semantic structure of the language. Unification between intensional set terms is not required at all. The extensional representations corresponding to intensional sets are always built first, and then set unification can be applied to the extensional set terms.

4 Dealing with infinite intensional sets

The translation technique used in $\{log\}$ is rather "crude". Intensional set terms are always replaced by the corresponding set-grouping clauses disregarding the context in which they occur. However, not always set-grouping is necessary, and sometimes it is not desirable, at all. In particular, this technique clearly fails to work properly with intensional sets that denote infinite sets. In this case, in fact, set-grouping necessarily leads to an infinite computation. For example, given the goal

$$:- \mathtt{s(0)} \in \{\mathtt{X} : \mathtt{nat(X)}\}$$

where `nat` defines a natural number, as usual, as

$$\mathtt{nat(0).} \qquad\qquad \mathtt{nat(s(X))} :- \mathtt{nat(X).}$$

$\{log\}$ tries to collect the (infinite) set of all natural numbers, and, so, it is unable to prove that the given goal is true (actually, the set-grouping mechanism repeatedly tries to collect larger and larger sets of natural numbers always finding that they are all partial sets).

Alternatively, one could try to exploit the intended semantics of intensional sets and rewriting the above operation simply as:

$$:- \mathtt{nat(s(0))}$$

which is easily proved to hold.

The very same idea could be applied also to the other primitive constraints of $\{log\}$.

In this section, we define four general procedures that allow to eliminate intensional set terms when occurring as arguments of simple set-theoretic operations, namely $\in$, $\notin$, $=$, and $\neq$.

These procedures can be proved to be correct and complete with respect to a theory that extends the set theory *Set* by the addition of two new axioms for characterizing intensional sets.

The new simplification rules for intensional set terms can be used, in addition to the rule for set-grouping already used in $\{log\}$, for obtaining a "finer" translation of a $\{log\}$ program P with intensional sets into a $CLP(\mathcal{SET})$ program P' with negation. Again, the $CLP(\mathcal{SET})$ program P' can be used as the semantics of the given $\{log\}$ program P with intensional sets.

It turns out that, although operations involving infinite sets can be undecidable in the general case, there are many special cases for which execution of the simplified program P' can be carried out successfully.

4.1 The theory Set^+

In order to precisely characterize intensional set terms we need to extend the theory *Set* with the addition of the well-known *comprehension scheme* of the ZF set theory

$$\forall s \, \forall y \, (y \in \{X \in s : \varphi[X]\} \leftrightarrow (y \in s \wedge \varphi[y])) \quad \text{for any f.o.f. } \varphi \, .$$

Condition '$X \in s$'—introduced by Zermelo in 1908—is used to overcome Russell's famous paradox (pick φ as $X \notin X$).

Actually, in $\{log\}$ unrestricted set formers $\{X : G\}$, where G is any goal, are admitted. Hence, the task of choosing G so as to constraint (either explicitly or implicitly) the scope of X is left to the programmer.

Moreover, we also add an axiom scheme stating that we assume that intensional sets are always based on the empty set kernel: $\mathtt{ker}(\{X : \varphi\}) = \emptyset$ for any f.o.f. φ.

The theory *Set*, extended with the two above axiom schemata, will be denoted Set^+.

4.2 Simplification rules for intensional sets

The following program transformation rules are intended to apply to any primitive constraint involving intensional set terms, occurring in goals (including those of intensional set terms) and in clause bodies of the given programs, as well as of the new programs generated by the repeated applications of the rules themselves.

Observe that intensional set terms can occur everywhere ordinary terms can occur. In particular, as argument of any functional symbol f, as well as in the goal part of another intensional set terms. Hereafter, we will refer to a term containing an intensional set term as an *intensional term*.

Equality constraints (=)

1.	$t = \{X : \varphi\}$ t is not a variable, and t is not an intensional set term	$\mapsto$	$\{X : \varphi\} = t$
2.	$\{X : \varphi\} = t$ $t = f(\cdots)$, $f \not\equiv \emptyset$, $f \not\equiv \{\cdot \mid \cdot\}$, and t is not an intensional set term	$\mapsto$	`false`
3.	$\{X : \varphi\} = \emptyset$	$\mapsto$	$\neg(\exists X\, \varphi)$
4.	$\{X : \varphi\} = \{t \mid s\}$	$\mapsto$	$(i)\quad t \in \{X : \varphi\} \wedge \{X : (\varphi \wedge X \neq t)\} = s$ $\vee$ $(ii)\quad t \in \{X : \varphi\} \wedge \{X : \varphi\} = s$
5.	$\{X : \varphi\} = \{X : \psi\}$	$\mapsto$	$\neg(\exists X\, ((\varphi \wedge \neg\psi) \vee (\neg\varphi \wedge \psi)))$

Observe that rules (3) and (5) would require the ability to deal with negated formulae involving existentially quantified variables (or, equivalently, formulae of the form $\exists\bar{Y}\forall X \neg\varphi[X, \bar{Y}]$) that are not directly expressible in $\{log\}$. In practice, however, the problem can be avoided by adding new clauses introducing the required existentially quantified variables. Precisely, if $\{\bar{Y}\} = vars(\varphi) \setminus \{X\}$,[4] then rule (3) is equivalent to replace the constraint $\{X : \varphi\} = \emptyset$ with $\neg\delta_\varphi(\bar{Y})$, where δ_φ is defined by the clause $\delta_\varphi(\bar{Y}) :\!-\ \varphi$. Moreover, if $\{\bar{Y}\} = (vars(\varphi) \cup vars(\psi)) \setminus \{X\}$, then the constraint $\{X : \varphi\} = \{X : \psi\}$ of rule (5) can be replaced by $\neg\delta_{\varphi,\psi}(\bar{Y})$, where $\delta_{\varphi,\psi}$ is defined by the clauses

$$\delta_{\varphi,\psi}(\bar{Y}) \quad :\!- \quad \varphi \wedge \neg\psi. \quad \delta_{\varphi,\psi}(\bar{Y}) \quad :\!- \quad \neg\varphi \wedge \psi.$$

which are immediately rendered as $\{log\}$ clauses with negation (see next section for a brief discussion about negation and intensional sets in $\{log\}$).

Also, observe that the disjunctions involved in rules (4) and (5) of $=$ (as well as in rules (4) and (5) of $\neq$ in the next table) can be conveniently rendered, in practice, through the non-determinism of the clause selection rule of the $\{log\}$ computation procedure.

Finally, note that part (ii) of rule (4) is devoted to cope with the presence of duplicates in the given (extensional) set term $\{t \mid s\}$. For instance, with predicate p defined simply as $p(a)$, the goal $\{X \mid p(X)\} = \{a, a\}$ is correctly rewritten into $a \in \{X \mid p(X)\} \wedge \{X \mid p(X)\} = \{a\}$ using rule $(4.ii)$ (whereas using rule $(4.i)$ would lead to a failure).

[4] $vars(\varphi)$ denotes a function that returns the set of free variables occurring in a formula φ.

Inequality constraints ($\neq$)

1.	$t \neq \{X : \varphi\}$ t is not a variable, and t is not an intensional set term	$\mapsto$	$\{X : \varphi\} \neq t$
2.	$\{X : \varphi\} \neq t$ $t = f(\cdots)$, $f \not\equiv \emptyset$, $f \not\equiv \{\cdot \mid \cdot\}$, and t is not an intensional set term	$\mapsto$	`true`
3.	$\{X : \varphi\} \neq \emptyset$	$\mapsto$	$\varphi[X/X']$, X' a new variable
4.	$\{X : \varphi\} \neq \{t \mid s\}$	$\mapsto$	(i) $t \notin \{X : \varphi\}$ $\vee$ (ii) $t \in \{X : \varphi\} \wedge t \in s \wedge \{X : \varphi\} \neq s$ $\vee$ (iii) $t \in \{X : \varphi\} \wedge t \notin s \wedge \{X : (\varphi \wedge X \neq t)\} \neq s$
5.	$\{X : \varphi\} \neq \{X : \psi\}$	$\mapsto$	(i) $Z \in \{X : \varphi\} \wedge Z \notin \{X : \psi\}$ $\vee$ (ii) $Z \in \{X : \psi\} \wedge Z \notin \{X : \varphi\}$ where Z is a new variable

Membership constraints ($\in$)

1.	$t \in \{X : \varphi\}$ t is not an intensional term	$\mapsto$	$\varphi[X/t]$

Negated membership constraints ($\notin$)

1.	$t \notin \{X : \varphi\}$ t is not an intensional term	$\mapsto$	$\neg(\varphi[X/t])$

The condition on t in rule (1) of $\in$ and $\notin$ is motivated by the following observation. Consider the constraint: $\{X : X \in X\} \in \{X : X \in X\}$. If we could use an extended version of rule (1) of $\in$ in which the condition for t is removed then the given constraint would be rewritten into $X \in X[X/\{X : X \in X\}]$, namely, $\{X : X \in X\} \in \{X : X \in X\}$: exactly the starting constraint! (this is, in a sense, the counterpart of Russell's paradox in our transformation rules). The condition on t in rules (1) of $\in$ and $\notin$, on the contrary, is sufficient to guarantee the termination of the global rewriting process, as shown in the next subsection.

Theorem 4.1 (Correctness and completeness) *The set of simplification rules for intensional sets shown above is sound and complete with respect to the set theory Set^+.*

Proof. By case analysis, we show that each rule fulfills the correctness and completeness property. Rules (1) of $=$ and (1) of $\neq$ are justified by equality theory. Correctness and completeness of rule (1) of $\in$ and (1) of $\notin$ follow directly from comprehension scheme.

Correctness and completeness of rules (2), (3), and (5) of $=$ and of $\neq$ follow directly from kernel axioms, comprehension scheme, and extensionality. Rule (4) of $=$ is justified by kernel axioms, comprehension scheme, extensionality, and the axioms of $\{\cdot \mid \cdot\}$. The fact that t can belong to s or not distinguishes the two cases. Rule (4) of $\neq$ is justified as follows. Given t, t either belongs to $\{X : \varphi\}$ or not. If $t \notin \{X : \varphi\}$ then the two sets are different by extensionality (case (i)). Assume $t \in \{X : \varphi\}$. Two cases are possible: $t \in s$ or $t \notin s$. In the former (case (ii)) it should be (by extensionality and comprehension) that $\{X : \varphi\} \neq s$. In the latter (case (iii)), that $\{X : \varphi\} \setminus \{t\} \neq s$. 4.1 □

4.3 Program transformation

A $\{log\}$ program P and goal G is transformed to an equivalent (w.r.t. the theory Set^+) $\{log\}$ program without intensional sets via a two steps process:

(i) repeatedly apply the simplification rules of Sec. 4.2 to P and G until no rule applies;

(ii) remove the remaining occurrences of intensional set terms by replacing them with the corresponding set-grouping clauses (cf. Sec. 2).

Termination of step (i) is assured by the following theorem. In the proof of the theorem, we will make use of the measure *intensional rank* (ir) of a term t. $ir(t)$ is the maximum nesting of intensional set definitions inside t. For instance, $ir(f(\{X : p(\{Y : r(Y,X)\})\}, \{X : p(X)\}) = 2$. For a non-intensional term t, $ir(t) = 0$; for a constraint or a formula φ, $ir(\varphi)$ is the maximum of $ir(t)$ for t term occurring in φ.

Theorem 4.2 (Termination) *Given a constraint C, the repeated application of the simplification rules for intensional sets shown above is always terminating.*

Proof. First observe that given an atomic constraint c in C, if some simplification rule replaces c with a new constraint, the rest of C is left untouched by this transformation. This allows us to reason about the termination of the simplification rules for each single atomic constraint.

Rules (1) of $=$ and of $\neq$ can be performed at most once for any constraint. Thus, they do not affect termination. Rules (2) of $=$ and of $\neq$ introduce simple constraints (resp., `false` and `true`) that will not start any other rule.

For the other cases, we need to introduce the notion of *intensional rank* of a constraint. We show that all other cases, when they do not carry to immediate termination, introduce constraints having fewer ir than the original constraint.

In rules (1) of $\in$ and (1) of $\notin$, since t is not an intensional term, then $ir(\varphi[X/t]) < ir(t \in \{X : \varphi\})$. In rules (3) and (5) of $=$ at least one intensional set term is eliminated from the given constraint, and, hence, ir strictly decreases. The constraint introduced by rule (4.i) of $\neq$ will start only rule (1) of $\notin$; thus, rule (4.i) of $\neq$ leads to reducing ir in two steps. The first constraint introduced by rules (4.i) and (4.ii) of $=$, $t \in \{X : \varphi\}$, leads to immediate termination if $ir(t) > 0$; otherwise (see (1) of $\in$ above) it will generate constraints of fewer ir. The second constraint introduced by these rules will start again rule (4.i) of $=$, as long as s contain elements. Thus, if s is of the form $\{s_1, \ldots, s_n \mid r\}$, with r not of the form $\{\cdot \mid \cdot\}$, it will start no more than n times such rule. The remaining equation $\{X : \varphi, \ldots\} = r$ will lead to immediate termination if r is a variable; otherwise, it will start one of the other terminating rules. The situation is very similar for rules (4.ii) of $\neq$ and (4.iii) of $\neq$. Finally, constraints introduced by rule (5) of $\neq$ will start rules (1) of $\notin$ and (1) of $\in$ that have been shown to reduce ir. 4.2 $\Box$

The new program P' and goal G' obtained at the end of step (i) can still contain intensional set terms either as arguments of those constraints to which no rewrite rule can be applied (e.g., $Z = \{X : \varphi\}$) or as arguments of user defined predicate and functional symbols. All these remaining occurrences of intensional set terms are removed by step (ii). Correctness and completeness of this final transformation come from formula (1), which in turn is justified by the comprehension scheme.

The final program is in a form such that it can be directly executed as a $CLP(\mathcal{SET})$ program with negation.

5 Intensional sets and Negation

As pointed out in [6], negation is strongly connected with set-grouping. Intuitively, negation is required when implementing $S = \{X : \varphi\}$ in the language itself to express the fact that φ is both a *necessary* and *sufficient* condition for X to belong to S.

The relation between intensional sets and negation is made even more evident by the rewriting rules shown in the previous section.

Basically, our approach allows to reduce the problem of handling intensional sets to the problem of handling negation: a subject, however, much more studied in the literature.

The problem of adding negation to $CLP(\mathcal{SET})$ has been analyzed in details in [2, 6].

When using the *Negation as Failure* approach to implement negation in $CLP(\mathcal{SET})$, the usual syntactic restrictions imposed by this approach to programs and goals to guarantee soundness and completeness of the language result in imposing too severe restrictions to the kind of intensional sets which can be collected (similar restrictions, however, apply also to $\mathcal{LDL}$ programs).

Most of these restrictions can be relaxed by taking a different approach to negation. The solution adopted in $CLP(\mathcal{SET})$ is that of extending the *Constructive Negation* (CN) technique proposed by Chan [3] and adapted to CLP by Stuckey [16], to the $CLP(\mathcal{SET})$ framework (where, in particular, unification is replaced by set unification, and the mgu uniqueness property does not hold anymore).

This solution, though it turns out that the CN technique does not cover all possible cases when used in the context of $CLP(\mathcal{SET})$—as well as of any CLP instance involving an undecidable structure—represents a true enhancement with respect to the use of the Negation as Failure rule. As a matter of fact, when using CN to implement intensional sets in $\{log\}$, the collected sets can contain both ground and non-ground elements, possibly involving constraints on the free variables occurring in the set former. For example, given the program

$\mathtt{p(a,a).}\qquad\qquad \mathtt{p(b,f(Z)).}$

and the goal $:-\ \{\mathtt{X} : \mathtt{p(X,Y)}\} = \emptyset$, the use of CN to implement $\neg\exists X p(X,Y)$ would allow us to get the answer $\mathtt{Y} \neq \mathtt{a}, \forall \mathtt{Z}(\mathtt{Y} \neq \mathtt{f(Z)})$.

6 Future work

There are a number of possible future developments of the ideas described in this paper, most of which are in progress at present.

The translation-based approach presented in the previous section is by definition a static approach, which cannot account for the dynamic behavior of the program at hand. Considerable enhancements could be obtained by allowing the simplification rules of the previous section to be applied also at run-time, after substitutions, possibly involving

$\{X : \varphi\} \cap \{X : \psi\} \quad \mapsto \quad \{X : \varphi \wedge \psi\}$
$\{X : \varphi[X, \bar{Y}_1]\} \cup \{X : \psi[X, \bar{Y}_2]\} \quad \mapsto \quad \{X : \delta(X, \bar{Y})\}$, where $\{\bar{Y}\} = \{\bar{Y}_1\} \cup \{\bar{Y}_2\}$ and δ is defined as follows: $\delta(\bar{Y}) :- \varphi \qquad \delta(\bar{Y}) :- \psi$
$\{X : \varphi[X, \bar{Y}_1]\} \setminus \{X : \psi[X, \bar{Y}_2]\} \quad \mapsto \quad \{X : \varphi[X, \bar{Y}_1] \wedge \neg\delta'(X, \bar{Y}_2)\}$
$\overline{\{X : \psi[X, \bar{Y}_2]\}} \quad \mapsto \quad \{X : \neg\delta'(X, \bar{Y}_2)\}$ where δ' is defined as follows: $\delta'(X, \bar{Y}_2) :- \psi$

Figure 1

intensional set terms, have been performed. This kind of enhancement demands for the ability of the constraint solver (the procedure SAT) to deal with constraints involving intensional set terms. For this purpose, the constraint simplification algorithms used by SAT can be extended so as to include the simplification rules of Sec. 4.2. Proving termination of this extended version of SAT is the difficult task, still in progress at present.

Another possible enhancement can be achieved by extending the technique for intensional constraint simplification to other basic set-theoretic operations, such as union, intersection, difference, and complement. In spite of their apparent difficulty, intensional set elimination for these operations can be easily implemented, as shown in Figure 1.

Finally, a number of syntactic extensions for intensional set terms (e.g., various forms of intensional set formers, like for instance, those of SETL [13]) are easily feasible, and would allow naturalness of many expressions involving intensional sets to be greatly enhanced.

Acknowledgements

We are much obliged to Eugenio Omodeo and Enrico Pontelli, whose qualified contributions emerge from a large part of the paper. G. Rossi is partially supported by MURST40% and C.N.R. Grant 96.02032.CT12.

References

[1] C. Beeri, S. Naqvi, O. Shmueli, and S. Tsur. Set Constructors in a Logic Database Language. *Journal of Logic Programming*, 10(3):181–232, 1991.

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

[3] D. Chan. Constructive Negation Based on the Completed Database. In *Proc. Fifth International Conference and Symposium on Logic Programming* (1988), R. Kowalski and K. Bowen, Eds., The MIT Press, Cambridge, Mass., 111–125.

[4] G. Delzanno and M. Martelli. A Bottom-Up Characterization of Finite Success. In D. Miller, editor, *Proc. of Int'l Logic Programming Symposium, ILPS'93*, Poster n. 676. The MIT Press, Cambridge, Mass., 1993.

[5] 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.

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

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

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

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

[10] B. Jayaraman. Implementation of Subset-Equational Programs. *Journal of Logic Programming*, 12(4):299–324, 1992.

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

[12] E. Legros, F. Ambert, and B. Legeard. Constraint Logic Programming on Sets and Multisets. In *Proc. ILPS94 post-conference workshop on Constraint Languages and their use in Problem Modelling*, 1994.

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

[14] R. Sigal. Desiderata for Logic Programming with Sets. In P. Mello, editor, *Proc. GULP89: 4^{th} national Conference on Logic Programming*, 1989.

[15] O. Shmueli, and S. Naqvi. Set Grouping and Layering in Horn Clause Programs. In J-L. Lassez, editor, *Proc. Fourth Int'l Conf. on Logic Programming*, pages 152–177. The MIT Press, Cambridge, Mass., 1987.

[16] P. J. Stuckey. Negation and Constraint Logic Programming. *Information and Computation 1* (1995), 12–33.