Intensional Sets in *CLP*

A. Dovier[1], E. Pontelli[2], and G. Rossi[3]

[1] Univ. di Udine, Dip. di Matematica e Informatica. dovier@dimi.uniud.it
[2] New Mexico State University, Dept. Computer Science. epontell@cs.nmsu.edu
[3] Univ. di Parma, Dip. di Matematica. gianfr@prmat.math.unipr.it

Abstract. We propose a parametric introduction of intensionally defined sets into any $CLP(\mathcal{D})$ language. The result is a language $CLP(\{\mathcal{D}\})$, where constraints over sets of elements of $\mathcal{D}$ and over sets of sets of elements, and so on, can be expressed. The semantics of $CLP(\{\mathcal{D}\})$ is based on the semantics of logic programs with aggregates and the semantics of constraint logic programming over sets. We investigate the problem of constraint resolution in $CLP(\{\mathcal{D}\})$ and propose algorithms for constraints simplification.

1 Introduction

The literature is rich of proposals aimed at developing declarative programming frameworks that incorporate different types of *set-based primitives* (e.g., [1, 10, 17, 4]). These frameworks provide a high level of data abstraction, where complex algorithms can be encoded in a natural fashion, by directly using the popular language of set theory. These features make this type of languages particularly effective for modeling and rapid prototyping of algorithms.

A recognized downside of most of the existing languages embedding sets is the focus on extensional set constructions [4, 10, 12, 8] and/or the severe restrictions imposed on the use of *intensional set constructions* [17, 16]. Intensionally defined sets (or simply, intensional sets) are collections of elements where the membership is decided by properties (instead of enumeration). There is significant evidence that the ability to handle general intensional sets can drastically simplify the development of solutions to complex problems, leading to more convenient languages and more compact programs.

In this work we propose a parametric introduction of intensionally defined sets into any $CLP(\mathcal{D})$ language—for instance $\mathcal{D}$ can be FD for Finite Domains, $\mathbb{R}$ for Real numbers, and so on. Given a language $CLP(\mathcal{D})$ and its interpretation domain D, we define the domain $\mathcal{U}_D$, which is used to construct an intuitive interpretation for intensional sets and set-based constraints (Section 3). We define a new language, $CLP(\{\mathcal{D}\})$, where $\mathcal{D}$-constraints can be expressed, as well as arbitrarily nested extensional and intensional sets of elements over $\mathcal{D}$ and constraints over these entities. The development of a semantics for $CLP(\{\mathcal{D}\})$ program involving intensional sets introduces problems similar to those arising for the semantics of logic programs with aggregates. In Section 2 we explain the relationships between aggregates and intensional sets.

The semantic characterization of $CLP(\{\mathcal{D}\})$ (Section 4) is provided as a generalization of Gelfond and Lifschitz stable model semantics; this allows us to provide a semantics to a larger class of programs than various previously proposed schemes (e.g., [17, 6])—in particular those relying on the use of stratification.

In Section 5 we build on our previous research on constraint solving in presence of sets [4] to provide an incomplete solver for constraints of $CLP(\{\mathcal{D}\})$. The proposed solver is aimed at simplifying constraints to a canonical form. In particular, the main goal is to eliminate occurrences of intensional sets from the constraints *without* explicitly enumerating their elements. Some constraints between intensional sets are easily seen to be undecidable (e.g., $\{X : p(X)\} = \{X : q(X)\}$); this prevents us from developing a parametric and *complete* constraint solver. To address this issue we subdivide the constraint solving process in two parts. The first part (*propagation*) relies on the use of rewriting rules that take advantage of the semantics of set operation to avoid the explicit computation of intensional sets. The second part (*labeling*) forces the removal of intensional sets—via translation to formulae requiring the use of negation and/or explicit grouping of solutions. The intuition is that, while propagation is transparently performed whenever possible, labeling—a potentially expensive step—should be explicitly requested.

2 Related work

A number of proposals have been made to support the introduction of aggregate functions in deductive databases and logic programming. Among them we discuss [11, 18, 15, 14, 3], where a *constraint aggregate* is an equation of the form

$$E = \mathbb{F}(\{\, e[\bar{X}, \bar{Y}, \bar{Z}] \,:\, (\exists \bar{Z})\, p[\bar{X}, \bar{Y}, \bar{Z}]\}) \tag{1}$$

whose intuitive semantics is: given values for the variables $\bar{Y}$ (grouping variables), collect all the expressions involving $\bar{X}$ such that there are values for the variables $\bar{Z}$ (local variables) such that $p[\bar{X}, \bar{Y}, \bar{Z}]$ holds. Then apply the function $\mathbb{F}$ on this (multi) set.

In this paper we consider a particular function $\mathbb{F}$: if S is a multiset of elements (defined by a property), then $\mathbb{F}(S)$ returns the set of all those elements (namely, it removes repeated occurrences of elements). On one hand, this may appear as a simplification—there is no need to compute possibly complex functions. On the other hand, this leads to a number of complications. In particular, sets have to be introduced as first-class citizens of the language and they must be properly interpreted and dealt with.

[14] provides a minimal model semantics for *monotonic* program (i.e., programs for which the T_P operator is monotonic). However, monotonicity is in general undecidable, and the syntactic restrictions they impose to ensure it are rather strong. Moreover, $\mathbb{F}$ can be only a simple function of the elements of the aggregate (such as SUM, MIN, MAX).

In [11], the authors introduce aggregate subgoals and they investigate both the three-valued Well-Founded Semantics and the two-valued Stable Model Semantics for programs containing aggregations. As in the case above, the aggregate function $\mathbb{F}$, is a simple function of multisets. In [15], the authors investigate the problem of checking satisfiability for programs with aggregate subgoals when the function $\mathbb{F}$ is a simple multiset SQL aggregate function. The well-founded and stable model semantics for logic programs with aggregates is extended through the use of approximation theory in [3].

The work that comes closer in spirit to what we propose here is [18], where Van Gelder provides a treatment of aggregates based on the capability of expressing a collection (findall) of answers to a predicate. The idea is that the function $\mathcal{F}$ can be easily programmed on top of this aggregate capability. Analogously to the proposal in [4] (and implicitly provided also in [17]), Van Gelder shows how to program findall using negation. As discussed at length in [5], the definition of set-grouping can exploit the following intended semantics of intensional sets:

$$\begin{aligned} E = \{X \,:\, p(X)\} &\leftrightarrow \forall X(X \in E \rightarrow p(X)) \wedge \forall X(p(X) \rightarrow X \in E) \\ &\leftrightarrow \forall X(X \in E \rightarrow p(X)) \wedge \neg\exists X(X \notin E \wedge p(X)). \end{aligned} \tag{2}$$

The first subformula can be implemented using a recursively defined predicate (for finite sets)—this corresponds to what authors have called *restricted universal quantification* [4, 12]—while the second subformula can be expressed as a negated predicate, whose single clause definition has $X \notin E \wedge p(X)$ as its body. Semantics is therefore reduced to the semantics of Prolog programs with negation. In [18] the Well-founded semantics has been employed to handle negation in this framework, while in [5] a general form of constructive negation was used. This approach is very general. Nevertheless, the direct transformation approach used to handle aggregations via negation has drawbacks. If the desire is to allow a general use of sets as first-class citizens of the language, then one needs to explore the interactions between set constructions and negation—which are not straightforward [5]. Moreover, investigating the semantics of aggregation through translation to other constructs hampers the development of implementation techniques that could be directly targeted to aggregation—especially implementations based on constraint solving and delaying techniques.

Our investigation builds on the work conduct in the development of a full-blown CLP language over sets. The language $CLP(\mathcal{SET})$ [4] is an instance of the CLP framework, whose constraint domain is that of *hereditarily finite sets*. $CLP(\mathcal{SET})$ allows sets to be *nested* and *partially specified*, e.g., set elements can contain unbound variables and sets can be only partially enumerated. $CLP(\mathcal{SET})$ provides a collection of primitive constraint predicates sufficient to cover all the basic set-theoretic operations (along with their negative counterparts, e.g., $\in$ and $\notin$). In [4] we presented a complete constraint solver capable of deciding the satisfiability of arbitrary conjunctions of these primitive set constraints. Intensional sets are allowed, but they are rewritten [5] according to the technique sketched in formula (2). Moreover, $CLP(\mathcal{SET})$ does not interoperate with other constraints solvers.

3 Syntax

Following the notation of [9], let $\mathcal{D}$ be an arbitrary constraint domain and $\mathcal{C}_{\mathcal{D}}$ be a class of *admissible constraints* for $\mathcal{D}$. The language we propose has a signature $\Sigma = \langle \Pi_{\mathcal{S}}, \Pi_{\mathcal{D}}, \Pi_{\mathcal{P}}, \mathcal{F}, \mathcal{V}\rangle$ ($\mathcal{S}$ stands for *Set*, $\mathcal{D}$ for *Domain*, and $\mathcal{P}$ for *Program*). Intuitively, $\Pi_{\mathcal{S}}$ provides constraint predicates to handle sets, $\Pi_{\mathcal{D}}$ provides the constraint predicates inherited from the underlying constraint domain $\mathcal{D}$, while $\Pi_{\mathcal{P}}$ contains the user-defined predicates. In particular, we assume that $\Pi_{\mathcal{S}}$ contains $\in, \cup_3, \cap_3, \subseteq, ||$ (these are the basic set predicates used in [4]). Furthermore, $\mathcal{F} = \mathcal{F}_{\mathcal{S}} \cup \mathcal{F}_{\mathcal{P}} \cup \mathcal{F}_{\mathcal{D}}$, where $\mathcal{F}_{\mathcal{D}}$ contains the function symbols provided by the language of $\mathcal{D}$, $\mathcal{F}_{\mathcal{P}}$ contains the free function symbols, while $\mathcal{F}_{\mathcal{S}}$ contains the traditional function symbols used to create terms denoting sets. In particular $\emptyset$ and the binary set-constructor symbol $\{\cdot \mid \cdot\}$ [4] are expected to be present in $\mathcal{F}_{\mathcal{S}}$ (Intuitively, the term $\{s \mid t\}$ denotes the set $\{s\} \cup t$). $\mathcal{F}_{\mathcal{S}}, \mathcal{F}_{\mathcal{P}}$, and $\Pi_{\mathcal{S}}$ allow to write terms and constraints regarding extensionally defined finite sets, such as $\emptyset, \{a, b\}, \{\emptyset, \{a, b\}\}$. The set $\mathcal{V}$ contains a countable number of variables. Variables are separated in the three sorts D, P, and S.

Definition 1. *We allow three sorts* D, P, *and* S *for* terms.

– *For* $\mathcal{X} \in \{\mathcal{D}, \mathcal{P}, \mathcal{S}\}$*, constant symbols from* $\mathcal{F}_{\mathcal{X}}$ *are terms of sort* X.
– *We assume that function symbols from* $\mathcal{F}_{\mathcal{D}}$ *of arity* n *have the sort* $\mathsf{D}^n \longrightarrow \mathsf{D}$.
– *The sort of* $\{\cdot \mid \cdot\}$ *is* $(\mathsf{D} \cup \mathsf{P} \cup \mathsf{S}) \times \mathsf{S} \longrightarrow \mathsf{S}$.
– *The sort of a function symbol from* $\mathcal{F}_{\mathcal{P}}$ *of arity* n *is:* $(\mathsf{D} \cup \mathsf{P} \cup \mathsf{S})^n \longrightarrow \mathsf{P}$.

Definition 2. Atoms *are defined as follows:*
– *If* $p \in \Pi_{\mathcal{P}}$ *with arity* n *and* $t_1, \ldots, t_n$ *are terms (of any sort), then* $p(t_1, \ldots, t_n)$ *is a* $\Pi_{\mathcal{P}}$*-atom.*
– *If* $p \in \Pi_{\mathcal{D}}$ *with arity* n *and* $t_1, \ldots, t_n$ *are terms of sort* D*, then* $p(t_1, \ldots, t_n)$ *is a domain constraint atom (or simply a* $\mathcal{D}$*-atom).*
– *If* $p \in \Pi_{\mathcal{S}}$ *with arity* n *and* $t_1, \ldots, t_n$ *are terms of sort* S*, then* $p(t_1, \ldots, t_n)$ *is an Intensional constraint atom. The only exception is represented by* $\in$*: if* t *is a term of any sort and* s *is a term of sort* S*, then* $t \in s$ *is also an Intensional constraint atom.*
– *if* $E \in \mathcal{V}$*,* $t_1, \ldots, t_n$ *are terms,* $p \in \Pi_{\mathcal{P}} \cup \Pi_{\mathcal{D}}$ *with arity* n*, then* $E = \{X : p(t_1, \ldots, t_n)\}$ *is an aggregate constraint atom. The sort of* E *is* S.

A $\mathcal{S}$*-atom is either an intensional constraint or an aggregate constraint atom.*

Observe that the aggregate constraint atom is exactly the constraint aggregate (1) where $\mathbb{F}$ is the identity function and the expression e is simply the variable X. We admit intensional sets only in atoms of the form above. Observe that limiting our attention to this form of aggregation does not lead to any loss of generality. The variables indicated with $\bar{Y}$ in (1) can appear as arguments of $p(t_1 \ldots, t_n)$, while the local (existentially quantified) variables can be directly placed within the program rules defining p.

Definition 3. *A* $\mathcal{D}$*-admissible constraint is any formula belonging to the class of constraints* $\mathcal{C}_{\mathcal{D}}$*. A* $\mathcal{S}$*-admissible constraint is a propositional combination of* $\mathcal{S}$*-atoms. For the sake of simplicity we do not allow the use of negation applied to aggregate constraint atoms. Let us denote with* $\mathcal{C}_{\mathcal{S}}$ *the class of* $\mathcal{S}$*-admissible constraints.*

A $\{\mathcal{D}\}$*-admissible constraint is a propositional composition of* D*-admissible and* $\mathcal{S}$*-admissible constraints. We will denote with* $\mathcal{C}_{\{\mathcal{D}\}}$ *the class of* $\{\mathcal{D}\}$*-admissible constraints.*

Definition 4. *A* $CLP(\{\mathcal{D}\})$ rule *is a formula* $H \leftarrow c_{\mathcal{S}}, c_{\mathcal{D}} | B_1, \ldots, B_n$ *where*
– $c_{\mathcal{D}}$ *is a* $\mathcal{D}$*-admissible constraint,*
– $c_{\mathcal{S}}$ *is an* $\mathcal{S}$*-admissible constraint, and*
– H *is a* $\Pi_{\mathcal{P}}$*-atom and* $B_1, \ldots, B_n$ *are* $\Pi_{\mathcal{P}}$*-literals (i.e.,* $\Pi_{\mathcal{P}}$*-atoms or their negations).*

A $CLP(\{\mathcal{D}\})$ *program* P *is a finite collection of* $CLP(\{\mathcal{D}\})$ *rules.*

4 Semantics

In this section we provide the semantics for the language $CLP(\{\mathcal{D}\})$. In particular, we propose an interpretation of the aggregate constraint atoms based on a variation of stable model semantics [7].

Let $\mathcal{D}$ be the initial domain constraint, D its interpretation domain, and $I_{\mathcal{D}}$ its interpretation function. We define the domain $\mathcal{U}_D = \bigcup_{i \geq 0} \mathcal{U}_i$ where:

$$\begin{cases} \mathcal{U}_0 = D \\ \mathcal{U}_{i+1} = \mathcal{U}_i \cup \wp(\mathcal{U}_i) \end{cases}$$

Observe that, if D is finite, then $\mathcal{U}_D$ is a set of hereditarily finite sets. Otherwise, $\mathcal{U}_D$ contains infinite sets. We will use the following partial order on $\mathcal{U}_D$: given two elements $a, b \in \mathcal{U}_D$: $a \leq b$ if and only if $a \subseteq b$ or $a, b \in \mathcal{D}$ and $a = b$.

Stable model semantics for logic programs are based on the notion of *grounding* of a clause. This semantic notion becomes rather syntactical when the interpretation domain is based on the Herbrand Universe. However, in a *CLP* context the domain is in general more complex than the Herbrand Universe. Thus, we redefine the notion of grounding based on our interpretation domain $\mathcal{U}_D$.

Definition 5. *(Pre-interpretation) Let t be a term and $\sigma : vars(t) \mapsto \mathcal{U}_D$ be a valuation function that maps variables of sort* D *to elements in D, variables of sort* S *to elements in $\mathcal{U}_D \setminus \mathcal{U}_0$, and variables of sort* P *to elements in $\mathcal{U}_D$. Furthermore, let R be a map that associates to each element $f \in \mathcal{F}_P$ of arity k a k-ary function from ${\mathcal{U}_D}^k$ to $\mathcal{U}_D$. R is called the* base *of the pre-interpretation. The pre-interpretation $t^{R,\sigma}$ of a term t w.r.t. R, σ is defined as follows:*
- *if t is a variable, then $t^{R,\sigma}$ is $\sigma(t)$.*
- *if t is a constant of sort* D*, then $t^{R,\sigma} = t^{\mathcal{D}}$ (i.e., the standard interpretation of the constant in the constraint domain $\mathcal{D}$).*
- *if t is $f(t_1, \dots, t_k)$ and $f \in \mathcal{F}_D$, then $t^{R,\sigma} = f^{\mathcal{D}}(t_1^{R,\sigma}, \dots, t_k^{R,\sigma})$, where $f^{\mathcal{D}}$ is the standard interpretation of f in $\mathcal{D}$.*
- *if t is $\emptyset$, then $t^{R,\sigma}$ is the empty set.*
- *if t is $\{a \,|\, b\}$, then $t^{R,\sigma}$ is the set $b^{R,\sigma} \cup \{a^{R,\sigma}\}$.*
- *if t is $f(t_1, \dots, t_k)$ and $f \in \mathcal{F}_{\mathcal{S}}$, then $t^{R,\sigma} = f^{\mathcal{S}}(t_1^{R,\sigma}, \dots, t_k^{R,\sigma})$. where $f^{\mathcal{S}}$ is the standard set-theoretic interpretation of f.*
- *if t is a constant in $\mathcal{F}_{\mathcal{P}}$ then $t^{R,\sigma}$ is simply t^R.*
- *if t is a variable of sort* P *then $t^{R,\sigma} = \sigma(t)$.*
- *if t is $f(t_1, \dots, t_k)$ and $f \in \mathcal{F}_{\mathcal{P}}$ then $t^{R,\sigma} = f^R(t_1^{R,\sigma}, \dots, t_k^{R,\sigma})$.*

If t does not contain variables, then we will simply use t^R instead of $t^{R,\sigma}$. In the rest of this discussion we will assume that the base R of the pre-interpretations is fixed.

Definition 6. *(Grounding) Given an atom A and a pre-interpretation R, σ for the terms in A (where σ is defined for all variables in A), we define the notion of* grounding of A *w.r.t. R, σ as follows:*
- *If $A = p(t_1, \dots, t_n)$ is a D-constraint atom, the grounding of $p(t_1, \dots, t_n)$ w.r.t. R, σ is* `true` *if $\mathcal{D} \models p(t_1^{R,\sigma}, \dots, t_n^{R,\sigma})$,* `false` *otherwise.*
- *If $A = p(t_1, \dots, t_n)$ is an Intensional constraint atom, then the grounding of the atom w.r.t. R, σ is the atom* `true` *if $p^{\mathcal{U}_D}(t_1^{R,\sigma}, \dots, t_n^{R,\sigma})$ is true (where $p^{\mathcal{U}_D}$ is the traditional set-theoretic interpretation of the predicate p on the domain $\mathcal{U}_D$), the atom* `false` *otherwise. In particular, given R and σ:*
  - *$\cup_3(s_1, s_2, s_3)$ is pre-interpreted as $s_3^{R,\sigma} = s_1^{R,\sigma} \cup s_2^{R,\sigma}$*
  - *$s \in t$ is pre-interpreted as $s^{R,\sigma} \in t^{R,\sigma}$*
  - *$s \subseteq t$ is pre-interpreted as $s^{R,\sigma} \subseteq t^{R,\sigma}$*
  - *$s || t$ is pre-interpreted as $s^{R,\sigma} \cap t^{R,\sigma} = \emptyset$*
  - *$\cap_3(s_1, s_2, s_3)$ is pre-interpreted as $s_3^{R,\sigma} = s_1^{R,\sigma} \cap s_2^{R,\sigma}$*
- *If $A = p(t_1, \dots, t_n)$ is a Π_P-atom, then its grounding w.r.t. R, σ is the object $p(t_1^{R,\sigma}, \dots, t_n^{R,\sigma})$.*
- *If A is an atom of the form $E = \{X : B\}$ and $B' = B[X/X']$ is B with X renamed to X' (X' a new variable), then its grounding w.r.t. R, σ is the equality $E^{R,\sigma} = \{X' : B'^{R,\sigma}\}$.*

Given an atom A and a pre-interpretation base R, a grounding of A w.r.t. R is a grounding of A w.r.t. R, σ for an arbitrary $\sigma : vars(A) \mapsto \mathcal{U}_D$ (such that R, σ is a pre-interpretation for the terms in A). The notion can be easily extended to rules.

Let us assume the base of the pre-interpretation R to be fixed. Let $I_D = \langle D, (\cdot)^D \rangle$ be the standard interpretation adopted for the constraint domain $\mathcal{D}$.

Definition 7. *An interpretation I is a pair $\langle \mathcal{U}_D, (\cdot)^I \rangle$, where the interpretation function $(\cdot)^I$ is defined as follows:*
- *I coincides with $I_\mathcal{D}$ on the interpretation of atoms built using $\mathcal{F}_\mathcal{D}$ and $\Pi_\mathcal{D}$.*
- *$\in, \subseteq, \cup_3$, and the other symbols in $\Pi_\mathcal{S}$ are interpreted in $\mathcal{U}_D$ according to their standard set-theoretical meaning.*
- *$=$ is interpreted as the identity over $\mathcal{U}_D$.*
- *$(\cdot)^I$ interprets each predicate symbol in Π_P as a predicate over $\mathcal{U}_D$.*
- *for each grounding R, σ of $E = \{X : A\}$, $(\cdot)^I$ interprets $(E = \{X : A\})^{R,\sigma}$ to* `true` *if $E^{R,\sigma}$ is equal to the set $\{X' : (B'^{R,\sigma})^I\}$ (where X' is a new variable and B' is B with X renamed to X').*

Observe that, under the assumption that the variable X does not occur in $p[\bar{Y}]$, then semantics of $\{X : p[\bar{Y}]\}$ then its semantics is the empty set $\emptyset$ if there is no value for $\bar{Y}$ s.t. $p[\bar{Y}]$; it is the entire universe $\mathcal{U}_D$ otherwise.

Let I and J be two interpretations on $\mathcal{U}_D$. We say that $I \preceq J$ if for each atom $P \equiv p(t_1, \ldots, t_n)$ and for each grounding R, σ of P there exists a an atom $Q \equiv p(s_1, \ldots, s_n)$ and a grounding R, θ of Q such that $(P^{R,\sigma})^I \to (Q^{R,\theta})^J$ and $(t_i^{R,\sigma})^I \leq (s_i^{R,\theta})^J$ for $i = 1, \ldots, n$. Given an atom P and a grounding R, σ of P, we define an interpretation I to be a *model* of $P^{R,\sigma}$ if $(P^{R,\sigma})^I$ is true. We denote this fact with $I \models P^{R,\sigma}$. Similarly, we can extend the definition of $\models$ to conjunctions of atoms.

We define an interpretation I to be a *model* of a grounded rule $head \leftarrow c_\mathcal{S}, c_\mathcal{D} \mid body$ if $I \models c_\mathcal{S} \wedge c_\mathcal{D} \wedge body$ implies $I \models head$. I is a model of a rule if it is a model of each grounding of the rule.

Stable Model Semantics. Let P be a $CLP(\{\mathcal{D}\})$ program and let I be an interpretation (built on the pre-interpretation base R). Let P' contain all possible R, σ groundings of the rules in P. Following the principles used in [7], we define the intensional stable model transformation $G(P, I)$. The transformation is achieved in two steps, i.e., $G(P, I) = G_{set}(G_{neg}(P, I), I)$.

The first transformation, G_{neg} is defined as follows: for each

$$head \leftarrow c_\mathcal{S}, c_\mathcal{D} \mid A_1, \ldots, A_n, \neg B_1, \ldots, \neg B_m$$

rule in P', if for all B_i, $1 \leq i \leq m$ it holds that $I \models \neg B_i$, then the rule instance

$$head \leftarrow c_\mathcal{S}, c_\mathcal{D} \mid A_1, \ldots, A_n$$

is added to $G_{neg}(P, I)$ (otherwise the rule is erased).

Observe that $G_{neg}(P, I)$ is the grounding of a program without negation.

The transformation G_{set} is defined as follows; for each rule $head \leftarrow c_\mathcal{S}, c_\mathcal{D} \mid A_1, \ldots, A_n$ in $G_{neg}(P, I)$ (i.e., a grounding of a program rule without negative literals), if for all atoms I in $c_\mathcal{S}$ it holds that $I \models A$ then the rule

$$head \leftarrow c_\mathcal{D} \mid A_1, \ldots, A_n$$

is added to $G_{set}(G_{neg}(P, I), I)$ (otherwise the rule is erased). The program $G_{set}(G_{neg}(P, I), I)$ is the grounding of a program without negation and without set atoms.

Definition 8. *I is a* stable model *of P if I is the least $\mathcal{U}_D$-model of the program $G_{set}(G_{neg}(P,I),I)$.*

Example 1. Consider the program: $r(1).p(X) \leftarrow X = \{Y : r(Y)\}$. and let us consider $G_{set}(P,I)$ for $I = \{r(1), p(\{1\})\}$. With $X = \emptyset$ the constraint aggregate becomes: $\emptyset = \{1\}$ which is an atom false in its interpretation in $\mathcal{U}_D$. This grounding is removed. The only true grounded clause is: $p(\{1\}) \leftarrow \{1\} = \{Y : r(Y)\}$. Thus, $G_{set}(G_{neg}(P,I),I) = \{r(1), p(\{1\})\}$. I is a stable model of P.

Proposition 1. *Let P be a $CLP(\{\mathcal{D}\})$ program.*
1. if I be a stable model of P, then I is a model of P.
2. if I is the unique stable model of P, then I is the $\preceq$-minimal model of P.

We cannot claim minimality in general. Consider:

$$\begin{array}{lll} r(1). & q(1). & \\ r(2). & q(2) \leftarrow Z = \{X : r(X)\}, p(Z). & p(Y) \leftarrow Y = \{X : q(X)\}. \end{array}$$

If we study the stable models we find two of them: $I_1 = \{q(1), q(2), p(\{1,2\}), r(1), r(2)\}$ and $I_2 = \{q(1), p(1), r(1), r(2)\}$. Observe that $I_1 \preceq I_2$. The fact that non-minimal models can be stable models is a common problem in the use of stable model semantics for handling of aggregates (without restrictions on the aggregate operation) [11]. Similar problems are present for the limited set aggregations described in [13].

5 Constraint solver

A $CLP(\{\mathcal{D}\})$ constraint is a conjunction of $CLP(\{\mathcal{D}\})$ primitive constraints. In the language $CLP(\{\mathcal{D}\})$ we recognize different classes of primitive constraints:

- $\mathcal{D}$-constraints: these are constraints that are built using exclusively symbols from $\mathcal{D}$; we assume that all the $\mathcal{D}$-constraints that can be generated belong to the class $\mathcal{C}_\mathcal{D}$. We also assume that this class of constraints is decidable, and we denote with $\mathcal{SAT}_\mathcal{D}$ the procedure used to solve this class of constraints.
  For the resolution of certain classes of constrains we will also require the class $\mathcal{C}_\mathcal{D}$ to be closed with respect to negation.
- Extensional $\mathcal{S}$-constraints: these are set constraints that do not involve any occurrence of intensional sets. This class of constraints is decidable and it corresponds to the class of constraints supported by the language $CLP(\mathcal{SET})$. In [4] an effective procedure to solve constraints in this class is proposed—we denote with $SAT_{\mathcal{SET}}$ such procedure. Traditional equality and disequality constraints between terms of the sort P are also treated in this procedure.
- Intensional $\mathcal{S}$-constraints: these are $\mathcal{S}$-constraints that contain occurrences of intensional sets. A procedure to handle this type of constraints is described in this section; we refer to this procedure as $SAT_\mathcal{S}$.

In order to accomplish the goal of resolving constraints in $CLP(\{\mathcal{D}\})$, we develop a constraint solver, called $\mathsf{Solve}_{\{\mathcal{D}\}}$. This solver repeatedly simplifies the constraint until no further simplifications are possible. The overall structure is shown in Fig. 1. Since constraint solvers can be non-deterministic, with $c' = SAT_\mathcal{X}(c)$ we mean one of the possible non-deterministic solutions returned. Thus, $\mathsf{Solve}_{\{\mathcal{D}\}}(C)$ is a non-deterministic procedure. In addition, for a program P we also introduce another function, called $\mathsf{Solvegoal}_P$.

$\mathsf{Solve}_{\{\mathcal{D}\}}(C)$:
repeat
  select c **in** C;
  if c *is a $\mathcal{D}$-constraint* **then** $c' = SAT_{\mathcal{D}}(c)$;
  if c *is an Extensional $\mathcal{S}$-constraint* **then** $c' = SAT_{\mathcal{SET}}(c)$;
  if c *is an Intensional $\mathcal{S}$-constraint* **then** $c' = SAT_{\mathcal{S}}(c)$;
  replace c **by** c' **in** C;
until *no rewriting is possible;*

Fig. 1. Overall structure of $\mathsf{Solve}_{\{\mathcal{D}\}}$

The predicate $\mathsf{Solvegoal}_P$ applied to a $\Pi_{\mathcal{P}}$ atom or formula means that the solving of its argument is delayed at the end of the constraint solving and it will be done by the general (constraint) resolution procedure. The only requirement we impose is that: $\mathsf{Solvegoal}_P(G) \Leftrightarrow G$. The intuition is that the $\mathsf{Solvegoal}_P$ will encode the language mechanisms required to support the explicit removal of intensional sets—e.g., using negation (as in Sect. 2). In this section we develop constraint solvers for intensional constraints, showing the generality of some rewriting rules (propagation) and the difficulties introduced by other rewriting rules (labeling). We also discuss the cooperation between the three solvers. We allow intensional terms to occur freely as terms in programs. It is immediate to automatically transform a program into the flat form of Def. 4.

Propagation Procedures. In this subsection we present some rewriting rules for $\mathcal{S}$-constraints of $CLP(\{\mathcal{D}\})$ that can be easily implemented starting from any initial domain $\mathcal{D}$; these rules allow us to deal finitely with intensional sets (without any restriction on the finiteness of the intensional sets). The application of these rewriting rules to a $\mathcal{S}$ constraint C leads to a disjunction of constraints that is equisatisfiable to C. However, the rewriting rules are incomplete, since constraints obtained might be unsatisfiable, but unsatisfiability could be not detected right away. Thus, the behavior is not dissimilar from that of incomplete constraint solvers used in other CLP systems—e.g., arc and bound consistency employed in FD solvers (e.g., in SICStus Prolog) are insufficient to detect unsatisfiability of a constraint such as

$$X \ \mathsf{in}\ 1..2, Y \ \mathsf{in}\ 1..2, Z \ \mathsf{in}\ 1..2, X \neq Y, X \neq Z, Y \neq Z.$$

A complete constraint solver can be obtained using a procedure analogous to the $\mathsf{labeling}$ used in $CLP(\mathcal{FD})$. We discuss this enhanced capability in the following subsection.

Some propagation rewriting rules are presented in Figure 2. These propagation rules are meant to complement the rewriting rules that have been proposed in our previous works to handle constraints over *extensionally defined sets*—due to lack of space we omit reproducing these rewriting rules, and the interested reader is referred to [4, 2]. Due to space limits, we give only the rewriting rules for some of the predicates involved, in particular for those assumed primitive in [4] and $\subseteq$. No rules are needed for $\not\cup_3$ since the unique rule in [4] apply to intensional sets, as well.

All the propagation rules presented in Fig. 2 do not actually compute the intensional sets—i.e., they do not force the explicit enumeration of the elements of the intensional set; thus, they can work without any assumption on the finiteness of these sets. Moreover, they do not introduce negation, but negated constraint literals that are treated as constraints—this can be seen, for example, in case =-2. in Fig. 2. In addition, some of the propagation rules are non-deterministic—see, for example, the case $\neq$-3. The presence of non-determinism leads to a family of constraints that are returned at the end of the processing, whose disjunction is equisatisfiable to the initial constraint C. In the rewriting rules we also omit the explicit description of the steps used to verify violation of the sorts of the predicates (with the exception of the =-1 and $\neq$-1 cases).

We use some syntactic sugar in the rewriting rules. We make use of the notation $\{X : \varphi_1 \vee \varphi_2\}$ to represent $\{X : p(\bar{Y})\}$, where $\{\bar{Y}\} = FV(\varphi_1) \cup FV(\varphi_2)$ and p is defined as: $p(\bar{Y}) \leftarrow \varphi_1 \vee \varphi_2$. Another syntactic sugar is $\mathsf{less}(X, Y, Z)$, defined as

$$\forall X, Y, Z(\mathsf{less}(X, Y, Z) \leftrightarrow Y = \{X \mid Z\} \wedge X \notin Z)$$

namely, Z is the set Y without the element X. Other rewriting rules for propagation are also easy to be defined; for instance,

$$\cap_3(\{X : \varphi_1\}, \{X : \varphi_2\}, s) \mapsto s = \{X : \varphi_1 \wedge \varphi_2\}$$

we omit a complete list for space reasons.

Other rewriting rules can be considered if we accept the use of negation. Negation is allowed in $CLP(\{\mathcal{D}\})$ programs. However, it is clear that it might introduce new problems. In particular, it introduces requirements on the capabilities of the $\mathcal{D}$ constraint solver to handle more difficult constraints. The rewriting rules proposed in Figure 3 rely on the assumption that the classes of constraints $\mathcal{C}_\mathcal{D}$ and $\mathcal{C}_\mathcal{S}$ are closed w.r.t. negation. Observe that the rules in Fig. 3 produce a negated version of the property employed to construct the intensional set. It is important to observe that the interaction between the constraint solvers may actually facilitate the handling of negated constraints—e.g., by grounding its argument and thus making it easier to solve. Although the rules for $\notin$- and $||$-constraints make use of negation, the intensional sets (that could be infinite) do not need to be explicitly enumerated (as described in equation (2) of Sect. 2). Thus, these rules are applicable *without* having to rewrite the aggregates using negation.

Correctness and completeness of the rewriting rules are immediate consequences of the semantics of the set-based operators involved. These results can be formally proved in the theory *Set* [4], a minimal set theory that deals with $\emptyset, \{\cdot \mid \cdot\}, =, \in, \cup_3, ||$ (and also with $\cap_3$ and $\subseteq$, that can be easily defined in terms of the previous ones). The theory has to be extended by adding the well-known *comprehension scheme* of the ZF set theory

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

Condition '$X \in s$'—introduced by Zermelo in 1908—is used to overcome Russell's famous paradox (pick φ as $X \notin X$). In the syntax for intensional sets this condition is not required. We assume that $X \in \mathcal{U}_D$. With $c \in \mathsf{Solve}_{\{\mathcal{D}\}}(C)$ we mean one of the possible non-deterministic solutions returned by $\mathsf{Solve}_{\{\mathcal{D}\}}(C)$.

Proposition 2. *(Non-negative Simplification) Let $\mathcal{C}_\mathcal{D}$ be a decidable class of $\mathcal{D}$-constraints; let $\mathsf{Solve}_{\{\mathcal{D}\}}$ be defined as in Fig. 1, with $SAT_\mathcal{S}$ composed of the rules in Figure 2. Let C be a $CLP(\{\mathcal{D}\})$ constraint and P be a $CLP(\{\mathcal{D}\})$ program. Then $Set, \mathcal{D}, P \models \left(C \Leftrightarrow \bigvee_{c \in \mathsf{Solve}_{\{\mathcal{D}\}}(C)} c\right)$.*

Proposition 3. *(Negative Simplification) Let $\mathcal{C}_\mathcal{D}$ be a decidable class of $\mathcal{D}$-constraints which is closed w.r.t. negation. Let $\mathsf{Solve}_{\{\mathcal{D}\}}$ be defined as in Fig. 1, with $SAT_\mathcal{S}$ composed of the rules in Figure 2 and in Figure 3. Let C be a $CLP(\{\mathcal{D}\})$ constraint and P be a $CLP(\{\mathcal{D}\})$ program. Then $Set, \mathcal{D}, P \models \left(C \Leftrightarrow \bigvee_{c \in \mathsf{Solve}_{\{\mathcal{D}\}}(C)} c\right)$.*

Rule	Rewriting
	$=$-constraints
$=$-1.	$\left.\begin{array}{r}\{X : \varphi\} = t \\ \mathsf{sort}(t) \neq \mathsf{S}\end{array}\right\} \mapsto \texttt{false}$
$=$-2.	$\{X : \varphi\} = \{t \,\vert\, s\} \mapsto$ (i) $t \in \{X : \varphi\} \wedge \{X : (\varphi \wedge X \neq t)\} = s$ (ii) $t \in \{X : \varphi\} \wedge \{X : \varphi\} = s$
	$\neq$-constraints
$\neq$-1.	$\{X : \varphi\} \neq t \mapsto \texttt{true}$ $\mathtt{sort}(t) \neq \mathsf{S}$
$\neq$-2.	$\{X : \varphi\} \neq \emptyset \mapsto \mathsf{Solvegoal}_P(\exists X \varphi)$
$\neq$-3.	$\{X : \varphi_1\} \neq \{X : \varphi_2\} \mapsto$ (i) $Z \in \{X : \varphi_1\} \wedge Z \notin \{X : \varphi_2\}$ (ii) $Z \in \{X : \varphi_2\} \wedge Z \notin \{X : \varphi_1\}$
$\neq$-4.	$\{X : \varphi\} \neq \{s \mid t\} \mapsto$ (i) $Z \in \{X : \varphi\} \wedge Z \notin \{s \mid t\}$ (ii) $Z \in \{s \mid t\} \wedge Z \notin \{X : \varphi\}$
	$\in$-constraints
$\in$-1.	$t \in \{X : \varphi\} \mapsto \mathsf{Solvegoal}_P(\varphi[X/t])$
	$\cup_3$-constraints
$\cup_3$-1.	$\cup_3(\{X : \varphi\}, \emptyset, s) \mapsto s = \{X : \varphi\}$
$\cup_3$-2.	$\cup_3(\{X : \varphi\}, s, \emptyset) \mapsto \emptyset = \{X : \varphi\} \wedge \emptyset = s$
$\cup_3$-3.	$\cup_3(\{s \mid t\}, r, \{X : \varphi\}) \mapsto$ $s \in \{X : \varphi\} \wedge \mathsf{less}(s, \{s \mid t\}, N) \wedge$ $\cup_3(N, r, \{X : \varphi \wedge X \neq s\})$
$\cup_3$-4.	$\cup_3(\{X : \varphi_1\}, \{X : \varphi_2\}, s) \mapsto s = \{X : \varphi_1 \vee \varphi_2\}$
$\cup_3$-5.	$\cup_3(\{X : \varphi_1\}, r, \{X : \varphi_2\}) \mapsto$ $r \subseteq \{X : \varphi_2\} \wedge \{X : \varphi_1\} \subseteq \{X : \varphi_2\} \wedge$ $\{X : \varphi_2\} \subseteq \{X : \varphi_1 \vee X \in r\}$
$\cup_3$-6.	$\cup_3(\{X : \varphi\}, r, \{s \mid t\}) \mapsto$ $\mathsf{less}(s, \{s \mid t\}, N) \wedge$ (i) $s \in \{X : \varphi\} \wedge s \notin r \wedge \cup_3(\{X : \varphi \wedge X \neq s\}, r, N)$ (ii) $s \in \{X : \varphi\} \wedge \mathsf{less}(s, r, N_1) \wedge \cup_3(\{X : \varphi \wedge X \neq s\}, N_1, N)$ (iii) $s \notin \{X : \varphi\} \wedge \mathsf{less}(s, r, N_1) \wedge \cup_3(\{X : \varphi\}, N_1, N)$
	$\parallel$-constraints
$\parallel$-1.	$\emptyset \parallel \{X : \varphi\} \mapsto \texttt{true}$
	$\nparallel$-constraints
$\nparallel$-1.	$\emptyset \nparallel \{X : \varphi\} \mapsto \texttt{false}$
$\nparallel$-2.	$\{X : \varphi\} \nparallel \{s \mid t\} \mapsto$ (i) $\mathsf{Solvegoal}_P(\varphi[X/s])$ (ii) $\mathsf{less}(s, \{s \mid t\}, N) \wedge \{X : \varphi\} \nparallel N$
$\nparallel$-3.	$\{X : \varphi\} \nparallel \{X : \psi\} \mapsto \mathsf{Solvegoal}_P(\exists X(\varphi \wedge \psi))$
	$\subseteq$-constraints
$\subseteq$-1.	$\emptyset \subseteq \{X : \varphi\} \mapsto \texttt{true}$
$\subseteq$-2.	$\{X : \varphi\} \subseteq \emptyset \mapsto \emptyset = \{X : \varphi\}$
$\subseteq$-3.	$\{s \mid t\} \subseteq \{X : \varphi\} \mapsto s \in \{X : \varphi\} \wedge t \subseteq \{X : \varphi\}$
$\subseteq$-4.	$\{X : \varphi\} \subseteq \{s \mid t\} \mapsto$ (i) $s \in \{X : \varphi\} \wedge \mathsf{less}(s, \{s \mid t\}, N) \wedge$ $\{X : \varphi \wedge X \neq s\} \subseteq N$ (ii) $s \notin \{X : \varphi\} \wedge \mathsf{less}(s, \{s \mid t\}, N) \wedge$ $\{X : \varphi\} \subseteq N$

Fig. 2. Propagation Rewriting Rules

$\notin$-constraints	
$\notin$-1.	$t \notin \{X : \varphi\} \mapsto \mathsf{Solvegoal}_P(\neg\varphi[X/t])$

$\Vert$-constraints	
$\Vert$-2.	$\{X : \varphi\} \Vert \{X : \psi\} \mapsto \mathsf{Solvegoal}_P\left(\exists X(\varphi \wedge \neg\psi) \vee \exists X(\neg\varphi \wedge \psi))\right) \vee$ $\{X : \varphi\} = \emptyset \wedge \{X : \psi\} = \emptyset$
$\Vert$-3.	$\{X : \varphi\} \Vert \{s \mid t\} \mapsto \mathsf{Solvegoal}_P(\neg\varphi[X/s]) \wedge \mathsf{less}(s, \{s \mid t\}, N) \wedge \{X : \varphi\} \not\Vert N$

Fig. 3. Propagation with negation

Labeling. The propagation rules allow us, given an original constraint C, to determine a disjunction of constraints that is equi-satisfiable to C. Each constraint belonging to the disjunction is "simpler" than C—e.g., it may contain fewer occurrences of intensional constraints. On the other hand, these rewriting rules do not constitute a complete solver—in particular, there is no guarantee that unsatisfiable constraints are reduced to `false` and tautologies are reduced to `true`. This situation makes the rewriting procedure weaker than, for example, the procedures used in $CLP(\mathcal{SET})$, where each set constraint can be always reduced either to `false` or to a satisfiable constraint in canonical form.

In order to approximate a similar behavior in the context of $CLP(\{\mathcal{D}\})$ it is necessary to force the removal of intensional sets from the constraint. This process can be seen as a sort of *labeling* of the variables in the constraint. More in detail, intensional sets must be expanded (and, possibly, $\mathcal{D}$-constraints may have to be subjected to a similar transformation). We show in Figure 4 how the labeling can be accomplished in the various cases left. Observe that the right-hand side of the transformation can be encoded either through the use of negation or by introducing an explicit construct to collect solutions to a goal (e.g., `findall`). The additional rewriting rules cannot guarantee completeness in general, since the problem is inherently undecidable.

$=$-constraints
3. $\{X : \varphi\} = \emptyset \mapsto \mathsf{Solvegoal}_P(\neg(\exists X \varphi))$
4. $\{X : \varphi_1\} = \{X : \varphi_2\} \mapsto \mathsf{Solvegoal}_P(\forall X(\varphi_1 \leftrightarrow \varphi_2))$

$\subseteq$-constraints
5. $\{X : \varphi_1\} \subseteq \{X : \varphi_2\} \mapsto \mathsf{Solvegoal}_P(\forall X(\varphi_1 \rightarrow \varphi_2))$

Fig. 4. Labeling using negation

Further Considerations. Negation as failure in the context of a language with sets has been studied only for programs that are stratified and meet restrictive allowedness requirements to avoid floundering [17]. Constructive negation in the context of Constraint Logic Programming with Sets has been studied in [5]. However, the class of programs that can be dealt with successfully does not enlarge significantly the class of those that can be dealt with negation as failure and stratification. Moreover, the $\mathcal{D}$ constraint solver should be able to deal with a class of constraints $\mathcal{C}_\mathcal{D}$ that is closed w.r.t. negation (a requirement which could be problematic in certain domains).

6 Conclusions

In this paper we presented preliminary ideas on how to extend any $CLP(\mathcal{D})$ language with set-based primitives and constraints. The novelty of the framework is not only the presence of intensional sets but the ability to develop (extensional and intensional) sets on top of arbitrary constraint domains $\mathcal{D}$. We developed a syntactic and semantics specification of the new language (called $CLP(\{\mathcal{D}\})$). We also developed rewriting algorithms to simplify constraints containing intensional sets—relying on the use of negation.

In the immediate future we plan to effectively implement the technique at least for some largely used constraint domains, such as finite domain constraint. A preliminary result in this direction is [2] where the $CLP(\mathcal{SET})$ constraint solver is integrated with the $CLP(\mathcal{FD})$ constraint solver of SICStus Prolog. In this preliminary work intensional set constraints are allowed but currently solved via explicit enumeration.

Acknowledgments. The work is supported by MIUR project: *Automatic Aggregate and number Reasoning for Computing* and NSF grants 0220590, CCR9875279, and 0130887.

References

1. P. Arenas-Sánchez and M. Rodríguez-Artalejo. A General Framework for Lazy Functional Logic Programming with Algebraic Polymorphic Types. *TPLP*, 2(1), 2001.
2. A. Dal Palù, A. Dovier, E. Pontelli, and G. Rossi. Integrating Finite Domain Constraints and CLP with Sets. In *PPDP*. ACM Press, 2003. To appear.
3. M. Denecker et al. Ultimate well-founded and stable semantics for logic programs with aggregates. In *ICLP*, pp. 212–226. Springer, 2001.
4. A. Dovier et al. Sets and Constraint Logic Programming. *TOPLAS*, 22(5):861–931, 2000.
5. A. Dovier, E. Pontelli, and G. Rossi. Constructive Negation and Constraint Logic Programming with Sets. *New Generation Computing*, 19(3):209–255, 2001.
6. M. Gelfond. Representing Knowledge in A-Prolog. In *Computational Logic: Logic Programming and Beyond*, pages 413–451. Springer Verlag, 2002.
7. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *JICSLP*, pages 1070–1080. MIT Press, 1988.
8. C. Gervet. Interval Propagation to Reason about Sets: Definition and Implementation of a Practical Language. *Constraints*, 1:191–246, 1997.
9. J. Jaffar and M. J. Maher. Constraint Logic Programming: A Survey. *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. D. B. Kemp and P. J. Stuckey. Semantics of Logic Programs with Aggregates. In *ILPS*, pages 387–401. MIT Press, 1991.
12. G. M. Kuper. Logic Programming with Sets. *JCSS*, 41(1):66–75, 1990.
13. I. Niemela and P. Simons. Extending Smodels with Cardinality and Weight Constraints. In *Logic-Based Artificial Intelligence*, pp. 491–521. Kluwer, 2000.
14. K. A. Ross and Y. Sagiv. Monotonic Aggregation in Deductive Databases. *Journal of Computer ans System Science*, 54:79–97, 1997.
15. K. A. Ross et al. Foundations of aggregation constraints. *TCS*, 193(1–2):149–179, 1998.
16. J. T. Schwartz et al. *Programming with Sets: an Introduction to SETL*. Springer, 1986.
17. O. Shmueli, S. Tsur, and C. Zaniolo. Compilation of Set Terms in the Logic Data Language (LDL). *Journal of Logic Programming*, 12(1/2):89–119, 1992.
18. A. Van Gelder. The Well-Founded Semantics of Aggregation. In *11th Principles of Database Systems*, pages 127–138. ACM Press, 1992.