A Language with Finite Sets Embedded in the CLP-scheme

Agostino Dovier

Università di Pisa, Dip. di Informatica
C.so Italia, 40. 56100–Pisa (Italy)
e-mail: `dovier@di.unipi.it`

Abstract. Problems and solutions related to the introduction of finite set formers and basic operations on sets in a Logic Programming language are discussed. In particular it is shown that a good solution is to begin with a CLP-scheme whose signature Σ is endowed with two functional symbols: $\emptyset$ for the *empty set* and `with` for the *set construction symbol*, using the symbols $\in$, $\notin$, $=$, $\neq$ as *constraint predicate symbols*. The axioms of the selected *set theory* are described, along with the corresponding algebraic interpretation and the constraint satisfiability algorithm. Other usual set operators (such as $\subseteq$, $\cup$, etc.) are shown to be definable in the extended language. Also, such an approach turns out to be well suited to accommodate for intensional set formers, providing the language is endowed with some form of negation.

1 Introduction

Aiming at extending a logic programming language with the addition of set manipulation capabilities, it is necessary to decide first what kind of objects and which operations on them the language should provide. Possible choices are, for instance:

- (finite) extensional sets, such as $\{t_0, \ldots, t_n\}$;
- (finite) intensional sets, such as $\{x \in S : \varphi\}$;
- predicate symbols $=$, $\in$, $\subseteq$, $\subset$;
- operators (by the way of functional/predicative symbols) $\cup$, $\cap$, $\setminus$.

Other similar objects such as *hyper-sets* (i.e. non well founded sets) or *multi-sets* [10] are not explicitly considered in this paper. Furthermore, it is important to decide *which* logic language has to be extended. In particular, we can choose between:

- Horn Clauses Language with SLD-resolution [9];
- Constraint Logic Programming [6].

Throughout the introduction we will not be too formal. Our aim here is to give an informal overview of the problem to an interested reader with some knowledge of logic Programming and of sets.

As a starting point we analyze the problem of representing extensional sets. At least two alternative solutions are viable:

i. $\{t_0, \ldots, t_n\}$ is represented as ***union of singletons***, i.e. $\{t_0\} \cup \ldots \cup \{t_n\}$;
ii. $\{t_0, \ldots, t_n\}$ is represented as a *list*, i.e. $(\ldots(\emptyset \, \texttt{with} \, t_n)\ldots) \, \texttt{with} \, t_0$.

In *i* three functional symbols are needed: $\emptyset$, of arity 0, $\{.\}$, of arity 1, and $\cup$, of arity 2. In a non trivial set theory (such as *ZF*), $\cup$ must be *Associative* (i.e. $A \cup (B \cup C) = (A \cup B) \cup C$), *Commutative* (i.e. $A \cup B = B \cup A$) and *Idempotent* (i.e. $A \cup A = A$). Moreover $\emptyset$ is the *identity* w.r.t. union (i.e. $A \cup \emptyset = \emptyset \cup A = A$).

In *ii* only two functional symbols are needed: $\emptyset$, of arity 0, and **with**, of arity 2. Again in a significant set theory, **with** must exhibit a *Right permutativity property* (i.e. $(X \, \texttt{with} \, Y) \, \texttt{with} \, Z = (X \, \texttt{with} \, Z) \, \texttt{with} \, Y$) and a *Right absorption Property* (i.e. $(X \, \texttt{with} \, Y) \, \texttt{with} \, Y = X \, \texttt{with} \, Y$).

However, notice that the unification problem for *set terms* would be NP-complete [5] (no matter which representation has been choosen).

In this paper, as well as in [3, 5], we have choosen approach *ii*. This is mainly motivated by the simpler (in the sense of number of independent solutions) unification problems characterizable with the latter approach.

Assuming the set representation approach *ii*, we could define a logic programming language, based on HCL with resolution but with the additional capability of performing the unification between set terms taking into account the properties of **with** described above.

The following basic predicates for set manipulation are then definable in the language[1]:

$$\begin{array}{ll}
=(X,X) \leftarrow & \cup(A,B,C) \leftarrow \\
\tilde{\in}(X, Y \, \texttt{with} \, X) \leftarrow & \quad (\forall X \in A)\tilde{\in}(X,C), \\
\subseteq(X,Y) \leftarrow & \quad (\forall Y \in B)\tilde{\in}(Y,C), \\
\quad (\forall Z \in X)\tilde{\in}(Z,Y) & \quad (\forall Z \in C)\, or(Z,B,C) \\
or(X, B \, \texttt{with} \, X, C) \leftarrow & or(X,B,C \, \texttt{with} \, X) \leftarrow
\end{array}$$

The resulting language (i.e. HCL + extensional set terms + set unification) is powerful though simple. Nevertheless, the following two issues (at least) cannot be resolved adequately:

- *effectiveness*: if, for instance, the resolution algorithm is applied to the goal $\leftarrow \subseteq (A, \emptyset \, \texttt{with} \, a)$ then an infinite SLD-tree is generated trying to compute the (sound) answers $A \mapsto \emptyset$, $A \mapsto \emptyset \, \texttt{with} \, a$, $A \mapsto \emptyset \, \texttt{with} \, a \, \texttt{with} \, a$, ... To solve the problem one could add the literal $X \notin Z$ to the body of the second clause defining $\subseteq$ (see *footnote 1*);

[1] The construct $(\forall x \in y)\varphi$ is intended to denote the formula $\forall x(x \in y \rightarrow \varphi)$. In [5] it is shown that an extended Horn clause containing such restricted universal quantifiers can be translated via simple pre-processing to a set of pure Horn clauses. For instance the definition of the predicate $\subseteq$, if $\Sigma = \{\emptyset, \texttt{with}\}$, can be re-written as follows: $\subseteq (\emptyset, Y) \leftarrow . \subseteq (Z \, \texttt{with} \, X, Y) \leftarrow \tilde{\in}(X,Y), \subseteq (Z,Y)$.

- *expressive power*: other basic set-operations, such as $\notin$, $\neq$, $\cap$, $\subset$, $\setminus$ cannot be defined unless some form of negation is introduced in the language (thus introducing also a number of well-known new problems).

Actually, having the definition of $\notin$ and (or) $\neq$, would suffice to solve these problems. In particular it is now possible to define the remaining set operations (set predicates are used infixed):

$$\begin{array}{ll}
\cap(A,B,C) \leftarrow & \setminus(A,B,C) \leftarrow \\
\quad (\forall X \in C)(X \tilde{\in} A \wedge X \tilde{\in} B), & \quad (\forall X \in A)\, or(X,A,B), \\
\quad (\forall Y \in A)\, \mathit{iff}(Y,B,C), & \quad (\forall Y \in C)\,(Y \tilde{\in} A \wedge Y \notin B) \\
\quad (\forall Z \in B)\, \mathit{iff}(Z,A,C) &
\end{array}$$

where a negative information is needed to define *iff*:

$$\begin{array}{ll}
\mathit{iff}(X,A,B) \leftarrow & \mathit{iff}(X,A,B) \leftarrow \\
\quad X \tilde{\in} A, X \tilde{\in} B & \quad X \notin A, X \notin B.
\end{array}$$

Furthermore, $\subset(X,Y) \leftarrow X \subseteq Y, X \neq Y$.

Hence the problem is that of introducing $\notin$ and $\neq$ into the HC language with set unification; if we want to avoid the drawbacks of full negation, the most elegant way to do it is introducing them as *constraints*.

Notice that $\notin$ and $\neq$ can be easily defined one in terms of the other:

$$\begin{array}{ll}
\neq(A,B) \leftarrow & \notin(A,B) \leftarrow \\
\quad A \notin \emptyset \;\mathtt{with}\; B & \quad B \;\mathtt{with}\; A \neq B.
\end{array}$$

Therefore, the simplest language able to deal with extensional sets in the desired way is a *CLP-like* language equipped with the proper handling of the constraint $=$ (for set unification) and $\neq$ (or alternatively $\notin$).

A hybrid solution based on this approach is described in [3, 5]. In that paper, $\notin$ and $\neq$ are treated as constraints; on the other hand $\in$ e $=$ are built in the language and their treatment is introduced directly into the resolution algorithm (for the sake of a more precise completeness result). The resolution step does not differ too much from the *logic* one of the CLP-scheme [6]. The main drawback there is the non-uniformity of treatment between $\in$ and $=$ and their negative counterparts.

In this paper we try to obviate these difficulties starting from a real CLP-Schema containing $\notin$, $\neq$, $\in$, and $=$ in the set of constraint predicative symbols.

Section 2 presents the language, its interpretation and its logical axiomatization. The satisfiability algorithm is described in section 3. A solution for the problem of introducing intensional sets is presented in section 4. In section 5 some future directions of the work are shown.

2 The language

Standard CLP notations and results [6] are assumed. As noticed in the previous section, we would like to have a CLP language able to deal with extensional set terms together with standard Herbrand terms. Therefore, we introduce into the signature Σ two particular functional symbols:

- **with**, binary, and
- $\emptyset$, nullary.

with will be used infixed, left associative: for example the term $\emptyset\,\mathbf{with}\,c\,\mathbf{with}\,(\emptyset\,\mathbf{with}\,b\,\mathbf{with}\,a)\,\mathbf{with}\,a$ will denote the set $\{a,\{a,b\},c\}$ (in this particular case we assume that a, b, and c belong to Σ).

If other functional symbols (i.e. a and b) are in Σ, we would like to write terms of the form $(a\,\mathbf{with}\,\emptyset)\,\mathbf{with}\,b$. Such a term will be interpreted as a 'coloured' set, i.e. a set based on an object different from $\emptyset$ (in this case a). Two sets will be considered equal if (and only if) they have the same elements and they are based on the same 'seed'. Furthermore we fix $\Pi_C = \{\in, \notin, =, \neq\}$.

2.1 Interpretation

First of all we must define the interpretation domain $\mathcal{A}$ (a single sort is sufficient for our purpose). Let us consider U_H, the Herbrand universe on $\Sigma = \{\emptyset, \mathbf{with}, \ldots\}$. Fixed an ordering on U_H, let $\equiv$ be the finest equivalence relation on U_H built with the right absorption and right permutativity property. Suppose we are able to choose a canonical representative for each equivalence class; then $\mathcal{A} = \{t : t \in U_H \wedge t \text{ is canonical }\}$.

Constructively, let $<_\Sigma$ be an order relation on Σ, we extend it to terms in reverse lexicographic order (in particular $r\,\mathbf{with}\,t < s\,\mathbf{with}\,u$ holds if $t < u$ or $t = s$ and $r < s$).

A ground term t is said to be *canonical* if:

- it is a constant, or
- each its subterm is canonical and, furthermore, for each subterm of t of the form $s\,\mathbf{with}\,u\,\mathbf{with}\,v$, $u < v$ holds.

We define the function τ mapping each term in its canonical representation, in the following way:

- $\tau(f(t_1,\ldots,t_n)) = f(\tau(t_1),\ldots,\tau(t_n))$ if f is different from **with**;
- $\tau(k\,\mathbf{with}\,t_1\,\mathbf{with}\ldots\mathbf{with}\,t_n) = \tau(k)\,\mathbf{with}\,s_1\,\mathbf{with}\ldots\mathbf{with}\,s_m$ where $s_1,\ldots,s_m$ ($m \leq n$) are the *distinct* canonical representatives of $t_1,\ldots,t_n$ such that $s_1 < \cdots < s_m$ (i.e. $\{s_1,\ldots,s_m\}$ and $\{\tau(t_1),\ldots,\tau(t_n)\}$ *coincides*).

Now we are ready to define the interpretation functions. $I_f^{\mathcal{A}}$ is defined as $I_f^{\mathcal{A}} = \lambda(x_1,\ldots,x_n).\tau(f(x_1,\ldots,x_n))$ for each f n-ary occurring in Σ. $I_=^{\mathcal{A}}$ is the identity function on $\mathcal{A}$; $I_{\neq}^{\mathcal{A}}(s,t) = \mathit{True}$ if and only if $I_=^{\mathcal{A}}(s,t) = \mathit{False}$. $I_\in^{\mathcal{A}}(s,t) = \mathit{False}$ if t is of the form $f(t_1,\ldots,t_n)$, f different from **with**; $I_\in^{\mathcal{A}}(r, s\,\mathbf{with}\,t) = \mathit{True}$ if and only if $I^{\mathcal{A}}_{=}(r,t)$ or $I^{\mathcal{A}}_{\in}(r,s) = \textit{True}$. $I^{\mathcal{A}}_{\notin}(r,s) = \textit{True}$ if and only if $I^{\mathcal{A}}_{\in}(r,s) = \textit{False}$. $I^{\mathcal{A}}(t\ \pi\ s) = I^{\mathcal{A}}_{\pi}(\tau(t), \tau(s))$, for π in $\{=, \in, \neq, \notin\}$. $I^{\mathcal{A}}$ will be then inductively extended to first order formulas in the usual way.

Such an interpretation is clearly ***solution compact*** (in fact each element $a \in \mathcal{A}$ is uniquely definable by the finite constraint $C = \{X = a\}$ - no limit elements occur in $\mathcal{A}$).

2.2 The theory

We are looking for a set theory $\mathcal{T}$ such that $\mathcal{A}$ and $\mathcal{T}$ *correspond* [6]. In what follows, free variables are intended to be universally quantified.

(U) (Scheme) $x \notin f(x_1, \ldots, x_n)$, for each f different from **with**;
(W) $x \in z \,\mathbf{with}\, y \leftrightarrow (x \in z \vee x = y)$
(L) $y \in x \rightarrow \exists z\, (y \notin z \wedge x = z \,\mathbf{with}\, y)$
(E) $v \,\mathbf{with}\, x = w \,\mathbf{with}\, y \leftrightarrow$

$$(x = y \wedge v = w) \vee (x = y \wedge v \,\mathbf{with}\, x = w) \vee$$
$$(x = y \wedge v = w \,\mathbf{with}\, y) \vee \exists z\, (v = z \,\mathbf{with}\, y \wedge w = z \,\mathbf{with}\, x)$$

(R) $\exists z \forall y (y \in x \rightarrow (z \in x \wedge y \notin z))$

Remarks: since $\emptyset \in \Sigma$, (U) states, in particular, the existence of an object which does not contain any element: the *emptyset*; (W) describes the *behaviour* of the functional symbol **with**, the set constructor; the *less* axiom (L) states the existence of the set $x \setminus \{y\}$; the *extensionality* axiom (E) shows how to decide if two sets can be considered equal; finally *regularity* axiom (R) assures that membership cannot form cycles.

Such a theory departs from the 'standard one' in two aspects:

- Presence of *ur–elements*. By (U) each term with a main functional symbol different from **with** is a set lacking in elements. In particular it is possible to introduce by definition the predicate $ur(x) \leftrightarrow \forall y\, (y \notin x)$ in the theory.
- Each term t has a *kernel* associated with it; if $ur(t)$ then t is also its kernel, otherwise the kernel is the seed on which the set term is based. We may then define, by induction, for each term x:

$$\begin{aligned} ker(x) = y \leftrightarrow {} & (ur(x) \wedge y = x) \vee \\ & (\exists v\, w\, (x = w \,\mathbf{with}\, v \wedge v \notin w \wedge y = ker(w))). \end{aligned}$$

It is easy to derive the following properties in the theory:

- $(x \,\mathbf{with}\, y) \,\mathbf{with}\, z = (x \,\mathbf{with}\, z) \,\mathbf{with}\, y$ (*permutativity*),
- $(x \,\mathbf{with}\, y) \,\mathbf{with}\, y = x \,\mathbf{with}\, y$ (*absorption*): (by using (W) and (E));
- Let (E*) be the formula $v \,\mathbf{with}\, x = w \,\mathbf{with}\, y \leftrightarrow (ker(v) = ker(w) \wedge \forall z\, (z \in v \,\mathbf{with}\, x \leftrightarrow z \in w \,\mathbf{with}\, y))$. (E*) holds in $\mathcal{T}$, if v, w, x, y are Σ-terms.

Standard equality axioms are assumed, together with the following *freeness* schemes of axioms:

1. $f(x_1,\ldots,x_n) \neq g(y_1,\ldots,y_m)$, if f is different from g;
2. $f(x_1,\ldots,x_n) = f(y_1,\ldots,y_m) \rightarrow (x_1 = y_1 \wedge \ldots \wedge x_n = y_n)$, if f is different from **with**;
3. $t[x] \neq x$ ($t[x]$ stands for a term different from x, with main functor different from **with**, in which x appears);
4. $t[x] \notin x$, and $ker(x) \neq t[x]$: they are needed to enforce (R) for 'sets' containing terms built not only with $\emptyset$, **with**, and variables.

In section 3.5 we will show that $\mathcal{T}$ is satisfaction complete. Furthermore the following lemma holds:

Lemma 1. *$\mathcal{T}(\Pi_C,\Sigma)$ and $\mathcal{A}(\Pi_C,\Sigma)$ correspond.*
Proof: (1): $\mathcal{A} \models \mathcal{T}$; it is sufficient to show that $I^{\mathcal{A}}(\varphi) = True$ for each axiom φ of $\mathcal{T}$.

(U) Immediate from the definition of $I^{\mathcal{A}}_{\in}(s,t)$;
(W) $I^{\mathcal{A}}(x \in z \,\mathbf{with}\, y) = True$ iff $I^{\mathcal{A}}_{\in}(\tau(x), \tau(z \,\mathbf{with}\, y)) = True$, iff $\tau(z \mathbf{with} y) \overset{\mathcal{A}}{=} h \mathbf{with}\{t_0,\ldots,t_n\}$[2] and $\tau(x)(\overset{\mathcal{A}}{=} x) \overset{\mathcal{A}}{=} t_i$ for some $i \in \{0,\ldots,n\}$. If $t_i \overset{\mathcal{A}}{=} \tau(y)(\overset{\mathcal{A}}{=} y)$ then $I^{\mathcal{A}}(x = y) = True$; else $I^{\mathcal{A}}(x \in z) = True$. The converse is straightforward;
(L) Let $x, y \in \mathcal{A}$ such that $I^{\mathcal{A}}(y \in x) = True$; then $x = h \,\mathbf{with}\, \{t_0,\ldots,t_n\}$ and $y \overset{\mathcal{A}}{=} t_i$ for some $i \in \{0,\ldots,n\}$. $z = h \,\mathbf{with}\, \{t_0,\ldots,t_{i-1},t_{i+1},\ldots,t_n\}$ is the z such that $\exists z \in \mathcal{A}\, I^{\mathcal{A}}(y \notin z \wedge x = z \,\mathbf{with}\, y) = True$;
(E) $I^{\mathcal{A}}(v \mathbf{with} x = w \mathbf{with} y) = True$ iff $\tau(v \mathbf{with} x) \overset{\mathcal{A}}{=} \tau(w \mathbf{with} y) \overset{\mathcal{A}}{=} h \mathbf{with}\{t_0,\ldots, t_n\}$. Now, $\tau(v \mathbf{with} x) \overset{\mathcal{A}}{=} h \mathbf{with}\{t_0,\ldots,t_n\}$ iff there exists i such that $t_i \overset{\mathcal{A}}{=} \tau(x)$ and $\tau(v) = h \,\mathbf{with}\, \{t_0,\ldots,t_n\}$ or $\tau(v) = h \,\mathbf{with}\, \{t_0,\ldots,t_{i-1},t_{i+1},\ldots,t_n\}$. Similarly, $\tau(w \,\mathbf{with}\, y) \overset{\mathcal{A}}{=} h \,\mathbf{with}\, \{t_0,\ldots,t_n\}$ iff there exists j such that $t_j \overset{\mathcal{A}}{=} \tau(y)$ and, $\tau(v) \overset{\mathcal{A}}{=} h \mathbf{with}\{t_0,\ldots,t_n\}$ or $\tau(v) \overset{\mathcal{A}}{=} h \mathbf{with}\{t_0,\ldots,t_{j-1},t_{j+1},\ldots,t_n\}$.
  - if $i = j$ then $\tau(x) \overset{\mathcal{A}}{=} \tau(y)$ and i) $\tau(v) \overset{\mathcal{A}}{=} \tau(w)$ or ii) $\tau(v) \overset{\mathcal{A}}{=} \tau(w \,\mathbf{with}\, y)$ or iii) $\tau(v \,\mathbf{with}\, x) \overset{\mathcal{A}}{=} \tau(v)$ or iv) let $z = h \mathbf{with}\{t_0,\ldots,t_{i-1},t_{i+1},\ldots,t_n\}$. Then $\tau(v) \overset{\mathcal{A}}{=} \tau(z \,\mathbf{with}\, y)$ and $\tau(w) \overset{\mathcal{A}}{=} \tau(z \,\mathbf{with}\, x)$.
  - if $i \neq j$, let $z = h \,\mathbf{with}\, \{t_0,\ldots,t_n\}$. Since τ eliminates duplications we have: $\tau(v) \overset{\mathcal{A}}{=} \tau(z \,\mathbf{with}\, y)$ and $\tau(w) \overset{\mathcal{A}}{=} \tau(z \,\mathbf{with}\, x)$.

  The converse is straightforward.
(R), (equality) and (freeness) are straightforward.

(2) $\mathcal{A} \models \exists C$ implies $\mathcal{T} \vdash \exists C$ for each constraint C: it is sufficient to prove the claim for atomical constraints: (a) $\mathcal{A} \models \exists \bar{x}(s = t)$, (b) $\mathcal{A} \models \exists \bar{x}(s \neq t)$, (c) $\mathcal{A} \models \exists \bar{x}(s \in t)$, (d) $\mathcal{A} \models \exists \bar{x}(s \notin t)$.

In order to show (a) we will prove the claim: *if* $\tau(s) \overset{\mathcal{A}}{=} \tau(t)$ *then* $\mathcal{T} \vdash (s = t)$, by structural induction on the ground term s.

[2] with the object $h \,\mathbf{with}\, \{t_0,\ldots,t_n\}$ we will denote the term $h \,\mathbf{with}\, t_0 \,\mathbf{with} \ldots \mathbf{with}\, t_n$

- s is a constant: $\tau(s) = s$. This implies that $t = \tau(t) = s$ and (by equality) we have $\mathcal{T} \vdash (s = t)$;
- s is $f(s_1, \ldots, s_n)$, f different from **with**: $\tau(f(s_1, \ldots, s_n)) = f(\tau(s_1), \ldots, \tau(s_n)) \overset{\mathcal{A}}{=} \tau(t)$. This means that t has the form $f(t_1, \ldots, t_n)$ and for each i $\tau(t_i) \overset{\mathcal{A}}{=} \tau(s_i)$. By induction hypothesis, for each $i \in \{1, \ldots, n\}$ we have $\mathcal{T} \vdash (s_i = t_i)$. Then $\mathcal{T} \vdash (s_1 = t_1 \wedge \cdots \wedge s_n = t_n)$, and, by equality, $\mathcal{T} \vdash (f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n))$;
- s is $h\,\mathtt{with}\{s_0, \ldots, s_m\}$: $\tau(s) = \tau(h)\,\mathtt{with}\{\tau(s_{i_0}), \ldots, \tau(s_{i_p})\}$ $(p \leq m)$; then t has the form $k\,\mathtt{with}\{t_0, \ldots, t_n\}$, where $\tau(t) = \tau(k)\,\mathtt{with}\{\tau(t_{j_0}), \ldots, \tau(t_{j_p})\}$, $(p \leq n)$ and, moreover, $\tau(h) \overset{\mathcal{A}}{=} \tau(k)$ and for each $r \in \{0, \ldots, p\}$, $\tau(s_{i_r}) \overset{\mathcal{A}}{=} \tau(t_{j_r})$. By induction hypothesis and equality we have: $\mathcal{T} \vdash (h\,\mathtt{with}\{s_{i_0}, \ldots, s_{i_p}\} = k\,\mathtt{with}\{t_{j_0}, \ldots, t_{j_p}\})$. By means of ***absorption*** and ***permutativity*** properties proved in the theory, the desired result follows.

(a) follows immediately. (c) follows from (a) and (W).

Likewise, to prove (b) and (d), we need the to prove following *if* $\tau(s) \overset{\mathcal{A}}{\neq} \tau(t)$ *then* $\mathcal{T} \vdash (s \neq t)$ always by structural induction on the ground term s:

- s is a constant: $\tau(s) = s$: by the first freeness axiom;
- s is $f(s_1, \ldots, s_n)$, f different from **with**: $\tau(s) = f(\tau(s_1), \ldots, \tau(s_n))$. $\tau(t) \overset{\mathcal{A}}{\neq} \tau(s)$ if $\tau(t) = g(r_1, \ldots, r_m)$, f different from g, or $\tau(t) = f(\tau(r_1), \ldots, \tau(r_n))$ and there exists i such that $\tau(s_i) \overset{\mathcal{A}}{\neq} \tau(r_i)$. By the first freeness axiom in the former and by the second freeness axiom and induction hypothesis in the second, we can get the proof.
- s is $h\,\mathtt{with}\,\{s_0, \ldots, s_m\}$: $\tau(s) = \tau(h)\,\mathtt{with}\,\{\tau(s_{i_0}), \ldots, \tau(s_{i_n})\}$. Therefore $\tau(t) = f(\tau(t_1), \ldots, \tau(t_p))$, f different from **with** (by first freeness axiom) or $\tau(t) = \tau(k)\mathtt{with}\{\tau(t_{j_0}), \ldots, \tau(t_{j_p})\}$ and *(i)* $\tau(h) \overset{\mathcal{A}}{\neq} \tau(k)$ or *(ii)* there exists l_1 such that $\tau(s_{i_{l_1}})$ does not appear in $\{\tau(t_{j_0}), \ldots, \tau(t_{j_p})\}$, or *(iii)* there exists l_2 such that $\tau(t_{j_{l_2}})$ does not appear in $\{\tau(t_{j_0}), \ldots, \tau(t_{j_n})\}$. By induction hypothesis and (E) the result holds.

This is sufficient for (b); (b) and (W) implies (d). □

3 Satisfiability algorithm

An *atomic constraint* is a (Π_C, Σ)-atom or the predicative constant *False*; a *constraint* is a finite set of atomic constraint. Given a constraint C, let $C = C_= \cup C'_{\neq} \cup C''_{\neq} \cup C_\in \cup C_\notin \cup C^F$ be a constraint, where

- each C_π is a finite set of atomic constraints over predicate symbol π;
- $C'_{\neq}, C''_{\neq}$ are respectively composed by atomic constraints not involving the symbol *ker* at all, and involving it at least once;
- C^F is empty or $\{False\}$.

We will refer to a non-variable term with main functor different from **with** as a *nucleo*.

The constraint solver is an algorithm which verifies the solvability in the structure (which implies satisfiability of the theory because of lemma 1) of a generic conjunction of (Π_C, Σ)–atoms. The initial constraint is successively transformed into an equisatisfiable disjunctive normal form; each disjunct is in a simplified form for which the satisfiability is guaranteed.

Here below we describe the actions taken by the algorithm on the different components of the constraint C.

3.1 Constraint $\in$

We eliminate all membership atomic constraints by replacing them with adequate equality atomic constraints:

```
function member($C$)
  if $C_\in = \emptyset$ then return $C$
  else choose $c$ in $C_\in$; let $C' = C \setminus \{c\}$
      case $c$ of
 1. $t \in s$ and $s$ is a *nucleo*: return $\{False\}$;
 2. $t \in X$ and $X$ is a variable: return $\{X = N \texttt{ with } t\} \cup$ member($C'$), N new
    variable;
 3. $t \in v \texttt{ with } w$: select non-deterministically from:
    (a) return member($\{t \in v\} \cup C'$)
    (b) return $\{t = w\} \cup$ member($C'$).
```

3.2 Constraint $=$

A constraint $C_=$ is said in *canonical form* if $C_= = \{X_1 = t_1, \dots, X_n = t_n\}$ and each variable X_i does not occur in $C \setminus \{X_i = t_i\}$. An equality atomic constraint $X = t$ of C is said to be in *canonical form* if X does not appear either in t or in $C \setminus \{X = t\}$:

```
function unify($C$)
  if $C_=$ is in canonical form then return $C$
  else choose $c$ (not in canonical form) in $C_=$; let $C' = C \setminus \{c\}$
      case $c$ of
 1. $X = X$: return unify($C'$);
 2. $t = X$, $t \notin \mathcal{V}$: return unify($\{X = t\} \cup C'$);
 3. $X = t$, $t$ is a *nucleo* and $X$ occurs in $t$: return $\{False\}$;
 4. $X = t \texttt{ with } t_n \texttt{ with} \dots \texttt{ with } t_0$ and $X$ occurs in $t_0$ or ... or in $t_n$, or $t$ is a
    *nucleo* and $X$ occurs in $t$: return $\{False\}$;
 5. $X = X \texttt{ with } t_n \texttt{ with} \dots \texttt{ with } t_0$ and $X$ does not occur in $t_0 \dots t_n$:
    return unify($\{X = N \texttt{ with } t_n \texttt{ with} \dots \texttt{ with } t_0\} \cup C'$), $N$ new variable;
 6. $X = t$, $X$ does not occur in $t$: return unify($C'\sigma$)$\cup\{X = t\}$, where $\sigma = \{X \mapsto t\}$;
 7. $f(t_1, \dots, t_n) = g(s_1, \dots, s_m)$, $f$ different from $g$: return $\{False\}$;
 8. $f(t_1, \dots, t_n) = f(s_1, \dots, s_n)$, $f$ is not with:
    return unify( $\{t_1 = s_1, \dots, t_n = s_n\} \cup C'$);
```

9. $h\,\mathtt{with}\,\{t_n,\ldots,t_0\} = k\,\mathtt{with}\,\{s_m,\ldots,s_0\}$, h and k *nucleos* or variables:
   (a) if h and k are not the same variable, then select non-deterministically one of the following actions:
      i. return unify($\{t_0 = s_0, h\mathtt{with}\{t_n,\ldots,t_1\} = k\mathtt{with}\{s_m,\ldots,s_1\}\} \cup C'$)
      ii. return unify($\{t_0 = s_0, h\mathtt{with}\{t_n,\ldots,t_0\} = k\mathtt{with}\{s_m,\ldots,s_1\}\}\} \cup C'$)
      iii. return unify($\{t_0 = s_0, h\mathtt{with}\{t_n,\ldots,t_1\} = k\mathtt{with}\{s_m,\ldots,s_0\}\}\} \cup C'$)
      iv. return unify($\{h\,\mathtt{with}\,\{t_n,\ldots,t_1\} = N\,\mathtt{with}\,s_0,$
      $N\,\mathtt{with}\,t_0 = k\,\mathtt{with}\,\{s_m,\ldots,s_1\}\} \cup C'$), N new variable;
   (b) otherwise select non-deterministically a number i in $\{0,\ldots,m\}$ and one of the following actions:
      i. return unify($\{t_0 = s_i, h\,\mathtt{with}\,\{t_n,\ldots,t_1\} =$
      $k\,\mathtt{with}\,\{s_m,\ldots,s_{i-1},s_{i+1},\ldots,s_0\}\} \cup C'$)
      ii. return unify($\{t_0 = s_i, h\,\mathtt{with}\,\{t_n,\ldots,t_0\} =$
      $k\,\mathtt{with}\,\{s_m,\ldots,s_{i-1},s_{i+1},\ldots,s_0\}\} \cup C'$)
      iii. return unify($\{t_0 = s_i, h\mathtt{with}\{t_n,\ldots,t_1\} = k\,\mathtt{with}\,\{s_m,\ldots,s_0\}\} \cup C'$)
      iv. return unify($\{h = N\,\mathtt{with}\,t_0, N\,\mathtt{with}\,\{t_n,\ldots,t_1\} =$
      $N\,\mathtt{with}\,\{s_m,\ldots,s_0\}\} \cup C'$), N new variable.

3.3 Constraint $\notin$

An atomic constraint $t \notin X$ is said to be in *canonical form* if X is a variable and X does not occur in t. A constraint $C_{\notin}$ is said to be in *canonical form* if every element is:

```
function notmember($C$)
  if $C_{\notin}$ is in canonical form then return $C$
  else choose $c$ (not in canonical form) in $C_{\notin}$; let $C' = C \setminus \{c\}$
      case $c$ of
 1. $t \notin r\,\mathtt{with}\,s$: return $\{t \neq s\} \cup$ notmember($\{t \notin r\}$)
 2. $t \notin f(t_1,\ldots,t_n)$, $f$ different from with: return notmember($C'$)
 3. $t \notin X$, $X$ variable occurring in $t$: return notmember($C'$).
```

3.4 Constraint $\neq$

We first deal with the constraints in $C'_{\neq}$, those in which *ker* does not appear. An atomic constraint in $X \neq t$ in $C'_{\neq}$ is said to be in *canonical form* if X is a variable not occurring in t. A constraint $C'_{\neq}$ is said to be in *canonical form* if all its elements are:

```
function notequal($C$)
  if $C'_{\neq}$ is in canonical form then return $C$
  else choose $c$ (not in canonical form) in $C'_{\neq}$; let $C' = C \setminus \{c\}$
      case $c$ of
 1. $f(t_1,\ldots,t_n) \neq g(s_1,\ldots,s_m)$, $f$ different from $g$: return notequal($C'$);
 2. $f(t_0,\ldots,t_n) \neq f(s_0,\ldots,s_n)$, $f$ different from with: select non-deterministical-
    ly $i$ in $\{0,\ldots,n\}$: return notequal($\{t_i \neq s_i\} \cup C'$);
```

3. $f \neq f$, f constant: return $\{False\}$;
4. $X \neq X$, X variable: return $\{False\}$;
5. $t \neq X$ and t is not a variable: return notequal($\{X \neq t\} \cup C'$);
6. $X \neq t$, t is a *nucleo* and X occur in t, or t is h **with** $t_n \ldots$ **with** t_1, h *nucleo* or variable, and X occurs in t_1 or ...or in t_n: return notequal(C');
7. $X \neq X$ **with** $t_n \ldots$ **with** t_0 select non-deterministically i in $\{0, \ldots, n\}$: return $\{t_i \notin X\} \cup$ notequal(C');
8. r **with** $s \neq t$ **with** u: select non-deterministically one of the three following actions (X denotes a new variable):
   (a) return $\{X \in r \textbf{ with } s, X \notin t \textbf{ with } u\} \cup$ notequal(C')
   (b) return $\{X \in t \textbf{ with } u, X \notin r \textbf{ with } s\} \cup$ notequal(C')
   (c) if $\Sigma \supset \{\emptyset, \textbf{with}, ker\}$ then return notequal($C' \cup \{ker(r) \neq ker(t)\}$) [3].

A few words about constraints involving the functional symbol *ker* are in order. We require explicitly that they can not be introduced by the user but only by the step 8-(c) of the function notequal. Moreover,we fix their canonical form either as:

- $ker(X) \neq f(t_1, \ldots, t_n)$, where X is a variable, f is different from **with** and from *ker* or
- $ker(X) \neq ker(Y)$, where X, Y are distinct variables.

The constraint $C''_{\neq}$ is in *pre-normalized form* if all its atomic constraints are in canonical form; it is in *canonical form* if it is in pre-normalized form and kernel_sat($C''_{\neq}$) $= true$, where kernel_sat is defined below:

```
function kernel_analyzer($C$)
  if $C''_{\neq}$ is in pre-normalized form
  then if kernel_sat($C''_{\neq}$)= $true$
       then return $C$
       else return $False$
  else choose $c$ (not in canonical form) in $C''_{\neq}$; let $C' = C \setminus \{c\}$
       case $c$ of
1. $ker(s) \neq \textbf{with}(t_1, t_2)$: return kernel_analyzer($C'$);
2. $ker(r \textbf{ with } s) \neq t$, $t$ has not the form $\textbf{with}(t_1, t_2)$:
   return kernel_analyzer($\{ker(r) \neq t\} \cup C'$);
3. $ker(f(t_1, \ldots, t_n)) \neq t$, $f$ different from with and $t$ has not the form
   $\textbf{with}(s_1, s_2)$: return kernel_analyzer($\{f(t_1, \ldots, t_n) \neq t\} \cup C'$);
4. $f(t_1, \ldots, t_n) \neq ker(t)$, $f$ different from $ker$:
   return kernel_analyzer($\{ker(t) \neq f(t_1, \ldots, t_n)\} \cup C'$);
5. $ker(X) \neq ker(X)$, $X$ is a variable: return $\{False\}$.
```

The function kernel_sat tests the satisfiability of the constraint $C''_{\neq}$ in pre-normalized form. It is ensured whenever the signature contains an infinite number of constant symbols, or at least a functional symbol of arity greater than 0,

[3] by $\supset$, we mean strict inclusion.

distinct from **with** and *ker* that $C''_{\neq}$ has solutions (if these are the cases, we are able to construct an infinity of different *kernels*). If the signature contains only a *finite* number of constant symbols, more than $\emptyset$, then satisfiability has to be checked.

```
function kernel_sat($C$)
  if $\Sigma$ is infinite or $\exists g \in \Sigma$ s.t. $arity(g) > 0$ or $C$ is the empty constraint
  then return $true$
  else ($\Sigma = \{\mathtt{with}, ker, \emptyset, a_1, a_2, \ldots, a_n\}, n > 1$)
      Let $\{X_1, \ldots, X_m\} = Vars(C)$;
      Consider the non-oriented graph of vertixes $V$ and edges $E$ s.t.
            $V = \{v_1, \ldots, v_m, a_1, \ldots, a_n\}$
            $< v_i, v_j > \in E$ iff $(ker(X_i) \neq ker(X_j)) \in C$
            $< v_i, a_j > \in E$ iff $(ker(X_i) \neq a_j) \notin C$
      if there exists an assignment $f$ from $\{v_1, \ldots, v_m\}$ to $\{a_1, \ldots, a_n\}$ s.t.
            1. $f(v_i) = a_j$ only if $< v_i, a_j > \in E$, and
            2. $\forall i, j \in \{1, \ldots, m\}, i \neq j$ no cycles of the form
               $< v_i, f(v_i) >, < f(v_i), v_j >, < v_j, v_i >$ occur
      then return $true$
      else return $False$.
```

3.5 The satisfiability function

A constraint C is in *canonical form* either if it is '*False*' or its components $C_=, C_{\notin}, C'_{\neq}, C''_{\neq}$ are in canonical form, and $C_{\in}$ is empty.

The function *rank*, defined as:

$$rank(x) = \begin{cases} 0 & \text{if } ur(x) \\ \max\{rank(s), 1 + rank(t)\} & \text{if } x \text{ has the form } s \,\mathtt{with}\, t \end{cases}$$

returns the 'depth' of a ground set. The function *find*, defined as:

$$find(x,t) = \begin{cases} \{0\} & \text{if } t \text{ coincides with } x \\ \emptyset & \text{if } t \text{ is } \emptyset \\ \{1 + n : n \in find(x,y)\} \cup find(x,s) & \text{if } t \text{ has the form } s \,\mathtt{with}\, y \end{cases}$$

returns the set of 'depth' in which a given element x occurs in the set t.

These two functions will be used in the following proposition to find a suitable $\mathcal{A}$-solution of atomical constraints. For instance, consider the atomical constraint $x \neq \{y\}$. By definition, $find(y, \{y\}) = \{1\}$. Considering the integer equation $v_x \neq v_y + 1$, obtained by the conditions above, and picking up one solution (i.e. $v_y = 0, v_x = 2$) we may define the substitution: $\sigma = \{x \mapsto \{\{\emptyset\}\}, y \mapsto \emptyset\}$ ($\{\{\emptyset\}\}$ has 'depth' 2, while $\emptyset$ has 'depth' 0). $(x \neq \{y\})\sigma$ is obviously true in $\mathcal{A}$.

Lemma 2. *Let C be a constraint in canonical form different from* $\{\text{False}\}$. *Then C is $\mathcal{A}$-solvable and $\mathcal{T}$-satisfiable.*

Proof: To start we assume that $C''_{\neq}$ is empty; successively we will show how to extend the proof to the general case. Let $C = C_= \cup C_{\notin} \cup C_{\neq}$, and let C_π in canonical form for each π in $\{=, \notin, \neq\}$.

$C_=$ takes the form $\{X_1 = t_1, \ldots, X_m = t_m\}$ and $\forall i = 1, \ldots, m$ X_i appears uniquely in $X_i = t_i$ and $X_i \notin Vars(t_i)$. Let us define $\theta_1 = \{X_1 \mapsto t_1, \ldots, X_m \mapsto t_m\}$. It is clear that $\mathcal{A} \models \forall C_= \theta_1$.

$C_{\notin}$ takes the form $\{r_1 \notin Y_1, \ldots, r_n \notin Y_n\}$ (Y_i does not appear in r_i) and $C_{\neq}$ takes the form $\{Z_1 \neq s_1, \ldots, Z_o \neq s_o\}$ (Z_i does not appear in s_i). Let $W_1, \ldots, W_h$ be the variables occurring in $r_1, \ldots, r_n, s_1, \ldots, s_o$ other than $Y_1, \ldots, Y_n, Z_1, \ldots, Z_o$. Furthermore, let $\theta_2 = \{W_1 \mapsto \emptyset, \ldots, W_h \mapsto \emptyset\}$, and let
$C^I_{\notin} = \{(t \notin X) \in C_{\notin}\theta_2 :$ **with** and $\emptyset$ are the only functional symbols in $t\}$,
$C^I_{\neq} = \{(X \neq t) \in C_{\neq}\theta_2 :$ **with** and $\emptyset$ are the only functional symbols in t, and t is not a variable$\}$.

Let $\bar{s} = max(\{size(t) : (t \notin X) \in C_{\notin}\theta_2\} \cup \{size(t) : (X \neq t) \in C_{\neq}\theta_2\})$ and let $V_1, \ldots, V_k$ the variables occurring in $C_{\neq}\theta_2 \cup C_{\notin}\theta_2$ but not in $C^I_{\neq} \cup C^I_{\notin}$. Let $\theta_3 = \{V_1 \mapsto \emptyset(\mathtt{with}\emptyset)^{\bar{s}+1}, \ldots, V_k \mapsto \emptyset(\mathtt{with}\emptyset)^{\bar{s}+k}\}$ and let $\bar{r} = \bar{s} + k + 1$.

It is straightforward to prove that $\mathcal{A} \models \forall(C_{\notin} \setminus C^I_{\notin})\theta_2\theta_3$ and $\mathcal{A} \models \forall(C_{\neq} \setminus C^I_{\neq})\theta_2\theta_3$.

Let $R_1, \ldots, R_j$ be the variables occurring in $C^I_{\notin} \cup C^I_{\neq}$. Let $n_1, \ldots, n_j$ be auxiliary variables. Build an integer disequation system E in the following way:

1. $E = \{n_i > \bar{r} : \forall i \in \{1, \ldots, j\}\} \cup \{n_{i_1} \neq n_{i_2} : \forall i_1, i_2 \in \{1, \ldots, j\}, i_1 \neq i_2\}$.
2. For each atomical constraint $(R_{i_1} \neq t) \in C^I_{\neq}$:
   $E = E \cup \{n_{i_1} \neq n_{i_2} + c : \forall i_2 \neq i_1, \forall c \in \mathit{find}(R_{i_2}, t)\}$
3. For each atomical constraint $(t \neq R_{i_1}) \in C^I_{\notin}$:
   $E = E \cup \{n_{i_1} \neq n_{i_2} + c + 1 : \forall i_2 \neq i_1, \forall c \in \mathit{find}(R_{i_2}, t)\}$

To solve the problem of finding a solution for E is trivial (it is sufficient to choose arbitrarily big solutions satisfying the constraints). Let $\{n_1 = \bar{n}_1, \ldots, n_j = \bar{n}_j\}$ be a solution, define $\theta_4 = \{R_i \mapsto \emptyset(\mathtt{with}\emptyset)^{\bar{n}_i} : \forall i \in \{1, \ldots, k\}\}$. Furthermore, let $\theta_5 = \{X \mapsto \emptyset : X$ appears in $C\theta_1\theta_2\theta_3\theta_4\}$, and let $\theta = \theta_1\theta_2\theta_3\theta_4\theta_5$; $C\theta$ is a ground constraint. Let us show that $\mathcal{A} \models C\theta$.

1. Pick $(X = t) \in C$; since $X\theta_1$ coincides with $t\theta_1 = t$, $\mathcal{A} \models (X = t)\theta$;
2. Pick $(t \notin X) \in C$: three cases are possible:
   (a) if f different from **with** occurs in t then $\mathcal{A}$ cannot model the membership of $t\theta$ (in which f occurs) in $X\theta$ (term of the form $\emptyset(\mathtt{with}\emptyset)^i$);
   (b) if t is one of the variables $W_1, \ldots, W_k$, then $t\theta = t\theta_2 = \emptyset$ cannot belong to $X\theta = \emptyset(\mathtt{with}\emptyset)^i$ since $i > \bar{s} + k + 1 > 1$;
   (c) Otherwise, from the solution of the integer system E, we obtain $rank(t\theta) \neq rank(X\theta) - 1$.
3. Analogous considerations can be applied to the constraints of the form $X \neq t$.

By (R) we get $rank(s) \neq rank(t) \rightarrow s \neq t$; if $C''_{\neq}$ is not empty, the function kernel_sat automatically supplies the elements to be used as kernels in the sets used to define the θ_is substitutions. Then C is $\mathcal{A}$-solvable. By lemma 1, C is $\mathcal{T}$–satisfiable. □

The canonical form algorithm is performed by the following (non-deterministic) function:

$$\text{step}(C) = \text{kernel\_analyzer}(\text{notequal}(\text{notmember}(\text{unify}(\text{member}(C))))).$$

Lemma 3. *Whatever are the non-deterministic choices performed during the execution of* step *there exists* n *such that* $\mathsf{step}^{n+1}(C) = \mathsf{step}^n(C)$ *is a constraint in canonical form (by* $\mathsf{step}^n(C)$ *we mean the iteration n times of* step *on an input* C *- conjunction of* (Π_C, Σ)*-atoms).*

Proof: In the case of termination, the procedure returns a constraint in canonical form (otherwise one of the steps of the algorithm would be applicable). Termination of the algorithm is based on the termination of each single function at any call. By introducing a measure K of structural complexity of the constraints relative to a specific predicate symbol, it is immediate that it decreases every time a new call of the function is performed. These partial results are combined into a global termination proof.

(1) Each function terminates at any call:

member. Assume $K = \sum_{(X \in t) \in C_\in} size(t)$; then, it decreases at every call;
unify. See [4];
notmember. Assume $K = \sum_{(X \notin t) \in C_\notin} size(t)$; then, it decreases at every call;
notequal. Let $K_1 = \sum_{(s \neq t) \in C'_{\neq}} size(s), K_2 = \sum_{(s \neq t) \in C'_{\neq}} (size(s) + size(t))$; then the pair $\langle K_1, K_2 \rangle$, ordered by the lexicographic order, is the selected complexity, which decreases at each call.
kernel_analyzer Let $K_1 = |\{(s \neq ker(t)) \in C''_{\neq}\}|$, $K_2 = \sum_{(s \neq t) \in C''_{\neq}} size(s)$. Due the peculiar form of the inequations containing ker, the selected complexity $< K_1, K_2 >$, with the lexicographic order decreases at each call.

(2) Suppose C_1 is returned by notmember (unify(member(C))).
Then, notequal(C_1) may introduce atomic constraints over predicates other than '$\neq$' only in the following cases:

7. In the successive call, if X does not occur in t_i, the constraint is in canonical form and then no actions may be performed on it; otherwise the constraint is eliminated by step 3 of notmember;
8. An atomic constraint of the form
   $h \,\mathbf{with}\, s_0 \ldots \mathbf{with}\, s_m \neq k \,\mathbf{with}\, t_0 \ldots \mathbf{with}\, t_n$, where h, k are variables or *kernels*, is replaced, according to step (a), by the following atomic constraints:
   $$Z \in h \,\mathbf{with}\, s_0 \ldots \mathbf{with}\, s_m \qquad (i)$$
   $$Z \notin k \,\mathbf{with}\, t_0 \ldots \mathbf{with}\, t_n \qquad (ii)$$
   (case (b) is analogous; case (c) does not raise By applying member, the constraint *(i)* is replaced by one of the following ones:
   - $Z = s_i$, for some $i \in \{0, \ldots, m\}$.
     Therefore, unify applies the substitution $\{Z \mapsto s_i\}$ (only in *(ii)*), and notmember deals with the atomical constraint $s_i \notin k \,\mathbf{with}\, t_0 \ldots \mathbf{with}\, t_n$. Then, notmember replaces the last constraint with: $\{s_i \neq t_0, \ldots, s_i \neq t_n, s_i \notin k\}$, where k is a variable and $s_i \notin k$ is in canonical form. The new call of notmember will work on objects having a smaller *size* and this implies the full termination.
   - $h = N \,\mathbf{with}\, Z$, where h is a *variable.*
     Then, unify applies the substitution $\{h \mapsto N \,\mathbf{with}\, Z\}$, making the con-

straint *size* bigger. This may be done a number of times less or equal to the number of occurrences of the variable h in C'.

(3) The total termination follows from the termination of the function obtained as follows:

1. $C_1 = \mathsf{notmember}(\mathsf{unify}(\mathsf{member}(C)))$;
2. $C_2 = C_1\theta$, where $\theta = \{X \mapsto \emptyset\,\mathtt{with}\,Z_1^{(X)}\,\mathtt{with}\ldots\mathtt{with}\,Z_{|X|}^{(X)}$, in which $\forall X$ occurring in $C_{1_{\neq}}$, we denote by $|X|$ the number of occurrences of X in $C_{1_{\neq}}$, $Z_j^{(v)}$ are new distinct variables;
3. $C_3 = \mathsf{notequal}(C_2)$.

The computation restarts from $\mathsf{step}(C_3)$; termination follows by the fact that the critical case is never generated in this way. □

Let sat be the function which computes the minimum fixpoint of step on the input C, defined as follows:

```
sat(C) = while step(C) ≠ C do
             C := step(C);
         return C.
```

The termination of step, proved in lemma 3, and the finiteness of the number of non deterministic choices generated by sat in correspondence of each call of step, ensure the finiteness of the number of constraints non-deterministically returned by sat. We may then state the following:

Lemma 4. $\mathcal{T} \vdash C \leftrightarrow \exists \bigvee_{\tilde{C}=\mathsf{sat}(C)} \tilde{C}$.
Proof: It is sufficient to show that, for each atomical action of each constraint function we have $\mathcal{T} \vdash c \leftrightarrow \exists(c_1 \vee \cdots \vee c_n)$ where c is the analyzed atomical constraint and $c_1, \ldots, c_n$ are the $n \geq 0$ constraints produced non-deterministically by the single action (see algorithm description for the notation):
member:

1. if $f \not\equiv \mathtt{with}$, $\mathcal{T} \vdash t \in f(t_1, \ldots, t_n) \leftrightarrow \mathit{False}$: by (U);
2. $\mathcal{T} \vdash t \in X \leftrightarrow \exists N\, X = N\,\mathtt{with}\,t$: $\leftarrow$ by (L), $\rightarrow$ by (W);
3. $\mathcal{T} \vdash t \in v\,\mathtt{with}\,w \leftrightarrow (t \in v \vee t = w)$: by (W).

unify:

1. For each variable X, $\mathcal{T} \vdash X = X \leftrightarrow \mathit{True}$ (the empty constraint represents True): by (equality);
2. $\mathcal{T} \vdash t = X \leftrightarrow X = t$: by (equality);
3. $\mathcal{T} \vdash X = t[X] \leftrightarrow \mathit{False}$: by (freeness 3);
4. $\mathcal{T} \vdash X = t[X]\,\mathtt{with}\,\{t_0, \ldots, t_n\} \leftrightarrow \mathit{False}$, and $\mathcal{T} \vdash X = h\,\mathtt{with}\,\{\ldots t[X] \ldots\} \leftrightarrow \mathit{False}$: by (freeness 4);
5. $\mathcal{T} \vdash X = X\,\mathtt{with}\,\{t_0, \ldots, t_n\} \leftrightarrow \exists N\, X = N\,\mathtt{with}\,\{t_0, \ldots, t_n\}$: $\leftarrow$ by (E), $\rightarrow$ by (W) e (L);

6. $\mathcal{T} \vdash C \cup \{X = t\} \leftrightarrow \{X = t\} \cup C^{\{X \mapsto t\}}$: by (equality);
7. $\mathcal{T} \vdash f(t_1, \ldots, t_n) = g(s_1, \ldots, s_m) \leftrightarrow \textit{False}$: by (freeness 1);
8. $\mathcal{T} \vdash f(t_1, \ldots, t_n) = f(s_1, \ldots, s_n) \leftrightarrow (t_1 = s_1 \vee \ldots \vee t_n = s_n)$: $\leftarrow$ by (equality), $\rightarrow$ by (freeness 2);
9. *(a)* is just (E); *(b)* follows from (L), (W) and (E).

notmember:

1. $\mathcal{T} \vdash t \notin r\, \mathtt{with}\, s \leftrightarrow t \neq r \wedge t \notin r$: by (W);
2. $\mathcal{T} \vdash t \notin f(t_1, \ldots, t_n) \leftrightarrow \textit{False}$: by (U);
3. $\mathcal{T} \vdash X \notin X \leftrightarrow \textit{False}$: by (R), $\mathcal{T} \vdash t[X] \notin X \leftrightarrow \textit{False}$: by (freeness 4).

notequal:

1. $\mathcal{T} \vdash f(t_1, \ldots, t_n) \neq g(s_1, \ldots, s_m) \leftrightarrow \textit{True}$: by (freeness 1);
2. $\mathcal{T} \vdash f(t_1, \ldots, t_n) \neq f(s_1, \ldots, s_n) \leftrightarrow (t_1 \neq s_1 \vee \ldots \vee t_n \neq s_n)$: $\leftarrow$ by (freeness 2), $\rightarrow$ by (equality);
3. $\mathcal{T} \vdash f \neq f \leftrightarrow \textit{False}$: as in (2) (the empty disjunction is equivalent to *False*);
4. $\mathcal{T} \vdash X \neq X \leftrightarrow \textit{False}$: as above;
5. $\mathcal{T} \vdash t \neq X \leftrightarrow X \neq t$: by (equality);
6. $\mathcal{T} \vdash X \neq t[X] \leftrightarrow \textit{False}$: by (freeness 3),
   $\mathcal{T} \vdash X \neq h\, \mathtt{with}\, \{\ldots t[X] \ldots\} \leftrightarrow \textit{False}$: by (freeness 4);
7. $\mathcal{T} \vdash X \neq X\, \mathtt{with}\, \{t_0, \ldots, t_n\} \leftrightarrow \exists i \in \{0, \ldots, n\}(X \neq t_i)$: $\leftarrow$ by (W), $\rightarrow$ by (L) and (E);
8. We have to show $\mathcal{T} \vdash r\, \mathtt{with}\, s \neq t\, \mathtt{with}\, u \leftrightarrow \exists x\, (x \in r\, \mathtt{with}\, s \wedge x \notin t\, \mathtt{with}\, u) \vee \exists x\, (x \notin r\, \mathtt{with}\, s \wedge x \in t\, \mathtt{with}\, u) \vee \textit{kernel}(r) \neq \textit{kernel}(t)$. This follows from (E), (L), (W) and the definition of *kernel.* □

As corollary we get:

Lemma 5. *C is $\mathcal{T}$-satisfiable if and only if there exists a non-deterministic choice such that* False $\notin$ sat(C). □

Theorem 6. *$\mathcal{T}$ is satisfaction complete.* □

The *algebraic derivation* is then algorithmically implementable. In order to implement the *logic* is therefore sufficient to choose one of the $\tilde{C}$ constraints different from $\{\textit{False}\}$ returned by sat(C) and as θ the substitution induced by $\tilde{C}_=$ (i.e. $\hat{\theta} = \tilde{C}_=$).

4 Intensional Sets

As pointed out in section 1, it would be interesting to introduce also intensional set formers in the language. If we allow set formers as general as $\{x : \varphi\}$, paradoxal situations would easily rise (for instance φ defined $x \notin x$ leads us to the Russell paradox). A more rigid syntax is then needed.

However, in this paper we want to focus on another problem: the correlation between set grouping and negation; suppose you wish to use an intensional construct in the definition of a predicate, for instance:

$$\begin{array}{ll} minimum_p(X) \leftarrow & minimum(Set, N) \leftarrow \\ \quad minimum(\{y : p(y)\}, X) & \quad N \in Set, \\ & \quad (\forall Z \in Set)\ lessorequal(N, Z). \end{array}$$

In order to compute *minimum*$_p$ we should be able to collect the set of computed answers of another predicate (set grouping facility [1]). Let us try to define such a facility using the language defined so far:

$$subsetof_p(X) \leftarrow (\forall Y \in X)\ p(Y);$$

such a predicate however does not compute $\{y : p(y)\}$ but all its subsets. Actually, we need a negative information, in order to be able to say: *'there does not exist any element Z not belonging to X such that $p(Z)$'*:

$$\begin{array}{ll} setof_p(X) \leftarrow & partial_p(X) \leftarrow \\ \quad (\forall Y \in X)\ p(Y), & \quad Z \notin X, \\ \quad \neg partial_p(X) & \quad p(Z). \end{array}$$

This motivates the need to introduce negation in the language. It is easy to write a transformation algorithm which transform a (Π_P, Π_C, Σ)-program with intensional set formers, into a $(\Pi_P \cup \Delta, \Pi_C, \Sigma)$-program with negation, where Δ is a new set of predicate names introduced by the translation.

5 Future work

Every work on the semantics and on implementation of the CLP-scheme may improve the property of the language described. In particular we wish to accomodate constructive negation techniques [2, 12] for the presented language, in order to be able to deal with intensional set terms.

Furthermore, wishing to extend the complexity of the manipulated terms (i.e. rational terms together with hyper-sets) it is sufficient to modify the satisfiability algorithm in a proper way. In particular, for each satisfiability result in set theories however defined, it is easy to find the corresponding programming language.

A purpose on extension of the CLP-scheme in the direction of providing it with set manipulation capability is presented in [8]; the representation choice is the one of *union of singletons* (see introduction); nevertheless semantics problems are not studied there and so it is not clear what kind of computed answer to expect from a given goal. Furthermore set operations are defined only on ground sets.

A general framework for the design of languages manipulating decidable theories based on modular extension of the resolution algorithm is presented in [11]. A comparison between it and the CLP-scheme seems to be promising.

Acknowledgements

My thanks are due to Davide Aliffi, Giorgio Levi, Alberto Policriti, Eugenio Omodeo, Alessandro Roncato, and, in particular, to Enrico Pontelli and Gianfranco Rossi.

References

1. C. Beeri, S. Naqvi et al. Set and Negation in a Logic Database Language. In *Proceedings* 6th *ACM SIGMOD Symposium*, 1987.
2. D. Chan. Constructive negation based on the completed databases. In *Proceedings 1988 Conference and Symposium on Logic Programming*, Seattle, Washington.
3. A. Dovier, E.G. Omodeo, E. Pontelli, and G. Rossi. {log}: A Logic Programming Language with Finite Sets. In *Logic Programming: Proceedings of the Eighth International Conference (K.Furukawa, ed.)*, The MIT Press, 1991, 111-124.
4. A. Dovier, E.G. Omodeo, E. Pontelli and G. Rossi. Embedding Finite Sets in a Logic Programming Language. *Research Report*, University of Rome, *"La Sapienza"*, 1993.
5. A. Dovier, E.G. Omodeo, E. Pontelli, and G. Rossi. Embedding Finite Sets in a Logic Programming Language. In E. Lamma, P. Mello eds, No. 660 of *Lecture Notes in Artificial Intelligence*, Springer Verlag.
6. J. Jaffar and J.L. Lassez. Constraint Logic Programming. *Research Report*, June 1986.
7. G.M. Kuper. Logic Programming with Sets. In *Proceedings* 6th *ACM SIGMOD Symposium*, 1987.
8. B. Legeard and E. Legros. CLPS: A Set Constraint Logic Programming Language. *Research Report*, Laboratoire d'automatique de Besançon. Institut de Productique, Besançon, France, 1991.
9. J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag series *Symbolic Computation - Artificial Intelligence*, 2nd edition, 1987.
10. E.G. Omodeo, A. Policriti and G. Rossi. Che genere di Insiemi–Multiinsiemi–Iperinsiemi incorporare nella Programmazione Logica? In 'GULP 93', *Proceedings of* 8th *Conference on Locgic Programming*. Gizzeria Lido, Italy, 1993.
11. A.Policriti and J.T.Schwartz. T-Theorem Proving. *Research Report*, University of Udine and Courant Institute of Mathematical Sciences, New York, 1992.
12. P.J. Stuckey. Constructive Negation for Constraint Logic Programming. In *Proc. Sixth IEEE Symp. on Logic In Computer Science*. IEEE Computer Society Press, 1991.