generalization, let us note that the Lemma 4.17 in [2] could be easily extended. Indeed, if \( P \) is a b.t.s. logic program, then for every finite set of variables \( V \), goal \( G \) and \( k \geq 1 \), the relation \( \sim_{V,G,k} \) has only finitely many equivalence classes. As a consequence, the Theorem 1 can be easily extended to b.t.s. programs.

**References**


---

**Solving Systems of Equations over Hypersets**

**Abstract**

A universe composed by rational ground terms is characterized (both constructively and axiomatically), where the interpreted set constructs \( \emptyset \) and with (the latter designating the element insertion operation) coexist with free Herbrand functors. Ordinary syntactic equivalence must be superseded by an equivalence relation \( \sim \), between trees labelled over a signature, that suitably reflects the semantics of with. Membership (definable as \( d \in t = \text{pci}(t \text{ with } d) \approx t \)) meets the non-well-foundedness property characteristic of hyperset theory. An algorithm for solving the NP-complete unification problem pertaining to hollow hyperset terms is provided, and shown to be totally correct. An application to the matching of finite state automata is hinted at.

**1 Introduction**

With this paper we wish to contribute to the field of constraint logic programming mainly in two ways. First, by characterizing hypersets both constructively and axiomatically (cf. Sec. 2, 3); then, by specifying a procedure for normalizing constraints of the most basic kind over hypersets. Our procedure will be shown to always terminate and to give the proper results (cf. Sec. 4). A by-product of our constraint normalization procedure is an algorithm for solving a novel NP-complete unification problem pertaining to hypersets, which has direct applications on its own: it can be used, for instance, to determine whether two finite state automata characterize the same language (cf. Sec. 5).

Hypersets of the kind we will propose are very intimately related to Aczel's non-well-founded sets (cf. [1]), but they all have finite cardinality and height; moreover they are hybrid, in that their construction involves free Herbrand functors. The finiteness restriction comes from our willingness to regard hypersets as instances of an algorithmic data structure; the proposed hybridization comes from our goal to
soon have hypersets and first-order terms treated together and homogeneously inside a declarative programming system.

Hypersets share all formal properties with sets, save one. To wit, membership is traditionally assumed to be a well-founded relation over the universe of sets. Over the broader universe of hypersets, on the contrary, it must infringe well-foundedness in all possible ways. To state this more clearly, let us focus on pure (i.e., non-hybrid) hypersets for a short while. One can associate with every set/hyperset $\xi$ the rooted graph $\text{trans}(\xi)$ whose nodes are those $\zeta$ for which a chain $\zeta \in \cdots \in \xi$ of length 0,1 or more exists leading from $\zeta$ to $\xi$, and whose edges are all pairs $(\zeta, \xi_1)$, with $\xi_1 \in \xi$, of such nodes. Then $\xi$ is classified as being a set if $\text{trans}(\xi)$ has no paths of infinite length, as being a proper hyperset otherwise.

The (von Neumann) universe of sets is rich enough that, given a single-source graph $G$ devoid of infinite paths, a specific set $\xi$ can be found with $\text{trans}(\xi)$ isomorphic to $G$, provided no two nodes have the same immediate successors in $G$. The latter proviso reflects the extensionality postulate, according to which no two sets have the same members.

In the (Aczel) universe of hypersets a similar 'richness' principle — no longer restrained by the path finiteness requirement — holds, but it is so conceived as to reflect a cautious variant of extensionality insuring, e.g., that when $\xi_b$ is the sole member of $\xi_b$ for $b = 0, 1, \xi_0$ and $\xi_1$ cannot be different. Being clearer on this point will be possible only after we have introduced the notion of bisimulation (cf. Def 2), adapted from Milner's study of the semantics of concurrent processes. Then we can state the extensionality condition a rooted graph $(G, \varrho)$ must fulfill in order a graph $\text{trans}(\xi)$ isomorphic to $G$ can be found (the root $\varrho$ being the image of $\xi$ in the isomorphism): no bisimulation exists between distinct rooted full subgraphs $(G_0, e_0), (G_1, e_1)$ of $G$.

Notice that the structure of a pure hyperset $\xi$ is made fully explicit by the rooted unordered graph $\text{trans}(\xi)$. Our finiteness restriction, mentioned above, translates into the fact that finitely many arcs issue from each node $\zeta$ of $\text{trans}(\xi)$ and that the latter has finitely many nodes. Thus, $\xi$ is a set if and only if $\text{trans}(\xi)$ is acyclic.

We must represent each hyperset $\xi$ of ours by a structure slightly more complex than $\text{trans}(\xi)$, because $\xi$ can be hybrid: accordingly, we will employ graphs with labelled nodes and an order imposed on the arcs issuing from each node. In essence, we are to generalize the notion of ground term so that it encompasses the notion of hyperset. A similar hybridization was carried out in [6], where we took into account sets only, and combined them with ordinary terms. In the current wider framework, it turns out natural to call into play infinite (rational) terms (cf. [8, 10]), because these are to ordinary terms as hypersets are to sets.

The reasons for moving from sets to hypersets are indeed very similar to those for moving from finite to infinite terms, from trees and dags to cyclic graphs, etc. It is not just the intellectual challenge that forces research outside the reassuring realm of hierarchic structures. Circularities, self-references, and the like, occur in concrete situations as well as in natural language. Formal methods cannot effectively cope with circular real-life phenomena unless (governable) circularities progress into the methods themselves, into programming habits, into data structures, etc. — cf. [2, 3].

2 The Hybrid Hyperset Universe

The entities that form a Herbrand universe are sometimes characterized as being finite trees coherently labelled over a signature $\Sigma$. This abstract view of ground terms becomes almost mandatory when one comes to consider the generalized kind of terms that form the completion of a Herbrand universe: typically this is done by withdrawing the requirement that labelled trees must have finitely many nodes — cf. [9], Ch. 6.2 From this graph-theoretical perspective, syntactic equivalence between terms turns out to coincide with the notion of isomorphism between labelled, ordered trees.

Given a term $T$, one can ‘fold’ it by fusing two nodes $\nu, \mu$ of $T$ into a single node whenever the subterms rooted at $\nu, \mu$ are equivalent to each other. This will lead to a rooted graph $G$ retaining information of all essential features of $T$: the picture of $T$, as we name it. If there are no infinite paths in $G$, this indicates that the original $T$ was already finite: this is the case of an ordinary term. When, less demandingly, $G$ is finite, $T$ might be infinite nevertheless, but distinct nodes $\nu, \mu$ lie on every infinite path $\varpi$ of $T$, such that the subterms rooted at $\nu, \mu$ are equivalent to one another. In the latter case, $T$ is said to be a rational term.

Generally speaking, the complete Herbrand universe is not constituted by algorithmic data structures. However, if one restricts one's own attention to rational terms and represents them suitably (e.g., by their graph pictures), then, assuming the signature $\Sigma$ is finite, even infinite terms can be algorithmically construed and manipulated.

In the following, we adjust the whole circle of ideas discussed so far to a case when the construction of the universe is not entirely free. We assume, in fact, that $\Sigma$ comprises a symbol to which a special, fixed meaning is attributed: this is $\text{with}$, used as a left-associative infix operator, by which we intend to model the operation of inserting an element into a set — or, more generally, into a hyperset (cf. [2]). The intuitive semantics of this construct must reflect into the criteria we adopt for equivalencing rooted graphs. Such criteria cease accordingly, in our specialized context, to be purely syntactic. At an even more fundamental level, we will have to discard certain trees labelled over $\Sigma$, that cannot be regarded as ground terms due to the semantics of $\text{with}$.

The terms whose root bears a label distinct from $\text{with}$ will be regarded as memberless entities, named colors. Quite unconventionally, we will allow insertions like $C \text{with } X$ for any color $C$ distinct from $\emptyset$ (our name for the 'official' empty set), regarding any hyperset that results from an insertion of this kind as something distinct from $\emptyset \text{with } X$.

To proceed more formally, let us denote by $w$ the set of all non-negative integers and by $\Sigma$ a finite collection of symbols comprising $\emptyset$ and $\text{with}$, with an arity mapping $\text{ar} : \Sigma \rightarrow w$ such that $\text{ar}(\emptyset) = 0$ and $\text{ar}(\text{with}) = 2$. We start by recalling a classical definition (cf. [9]), which still awaits a minor adaptation to our aims:

**Definition 1** A ground term (over $\Sigma$) is a mapping $T : \text{dom}(T) \rightarrow \Sigma$ such that

---

1 A rooted graph is one with a designated node $\varrho$ whence every other node is reachable. Here we clearly have $\varrho = \xi$.

2 Alternatively, as explained at the end of this section, one could characterise generalised ground terms by means of systems, possibly infinite, of equations involving ordinary hollow terms (cf. [8, 11]).


Figure 1: The terms depicted by these graphs cannot be regarded as ground.

- the domain \( \text{dom}(T) \) of \( T \) is a non-empty ordered tree whose root is \( [] \);
- for all \( \nu \) in \( \text{dom}(T) \), \( \text{ar}(T(\nu)) = \{ [i: [\nu, i] \text{ in } \text{dom}(T)] \} \), where \([\nu, i]\) stands for the \( i \)-th son of \( \nu \).

To avoid under-specified situations, we then add:

Groundness restriction. The requirement henceforth becomes integral part of the definition of (ground) term that there be no infinite sequence \( \nu_0, \nu_1, \nu_2, \ldots \) of nodes with \( T(\nu_i) = \text{with} \) and \( \nu_{i+1} = [\nu_i, 1] \) for all \( i \).

To see why the presence of a path \( \nu_0, \nu_1, \nu_2, \ldots \) as above, in a term, would conflict with the very notion of groundness, let us examine the two graphs of Fig. 1. Either of them is the picture of a labelled tree that violates the groundness restriction. The left arc in either graph indicates — if anything — self-inclusion; hence it conveys no information about the entity \( X_a \) and \( X_b \) respectively represented by the root. The second arc of Fig. 1(a) indicates that \( \emptyset \) must belong to \( X_a \) a property which is clearly insufficient to characterize \( X_a \). The right arc of Fig. 1(b) indicates that \( X_b \) must belong to itself. If \( X_a \) were to be an ordinary set, this would be an absurdity, but we are dealing with hyperset here. Since membership cannot form cycles among such entities, we are again facing an under-specified situation.

Our next step will be to get rid of irrational terms (an interesting example of set-theoretic irrational term is the one whose picture is the graph of Fig. 2). Preliminary to that, we need the notion of bisimulation, which in turn presupposes the following couple of auxiliary notions.

For every term \( T \) and any \( \nu \) in \( \text{dom}(T) \), let \( \tau_0, \ldots, \tau_g \) and \( \mu_0, \ldots, \mu_{g-1} \) be the sequences of nodes such that: \( \tau_0 = \nu; T(\tau_i) = \text{with}; \tau_{i+1} = [\tau_i, 1] \) and \( \mu_i = [\tau_i, 2] \) for \( i = 0, \ldots, g-1 \); \( T(\tau_g) \neq \text{with} \). We denote by \text{Color}(\nu) \) the node \( \tau_g \) and call \( \epsilon \)-predecessors of \( \nu \) the \( \mu_i \).

Definition 2 Let \( T_0, T_1 \) be terms. A relation \( B \subseteq \text{dom}(T_0) \times \text{dom}(T_1) \) is said to be a bisimulation between \( T_0 \) and \( T_1 \) iff: 

- \( [\text{Color}(\nu_0); B] [\text{Color}(\nu_1); T_0(\nu_0) = T_1(\nu_1)], \) and moreover
- for every \( \epsilon \)-predecessor \( g_0 \) of \( \nu_0 \) in \( T_0 \) \( (b = 0 \text{ or } b = 1) \), there corresponds at least one \( \epsilon \)-predecessor \( g_{1-1} \) of \( \nu_1 \) in \( T_1 \) such that \( g_0 \equiv g_{1-1} \);
- if \( T_0(\nu_0) \neq \text{with} \), then \([\nu_0, i] B [\nu_1, i] \) for \( i = 1, \ldots, \text{ar}(T_0(\nu_0)) \).

We write \( T_0 \approx T_1 \) iff there is a bisimulation \( B \) between \( T_0 \) and \( T_1 \).

Bisimulations are, in a sense, isomorphisms complying with the intended (hyperset) semantics of with. Accordingly, \( T \) will be regarded as a rational term iff it has only finitely many subterms that cannot bisimulate one another. To make this idea precise, let us denote by \( T(\nu) \) the subterm of \( T \) issuing from a given node \( \nu \).

Definition 3 A ground term \( T \) is said to be rational iff there are \( \nu_0, \ldots, \nu_m \) in \( \text{dom}(T) \), with \( m \) in \( \omega \), such that for every \( \mu \) in \( \text{dom}(T) \) there is an \( i \), \( 0 \leq i \leq m \), fulfilling \( T(\nu_i) \approx T(\mu) \).

In conclusion, indicating by \( \mathcal{G}_\Sigma, \mathcal{G}_\Sigma \) the family of all rational terms over \( \Sigma \) and its subfamily consisting of the terms that have finitely many nodes, our hyperset universe and set universe will be

\[
\mathcal{H}_\Sigma = \text{det} \mathcal{G}_\Sigma / \approx, \quad \mathcal{H}_\Sigma = \text{det} \mathcal{G}_\Sigma / \approx
\]

respectively. Representing by \( T^\omega \) the \( \omega \)-class of \( T \), the element insertion operation and membership relation over these universes can be straightforwardly defined as

\[
T_0^\omega \text{ with } T_1^\omega = \text{det } W^\omega, \quad T_0^\omega \in T_0^\omega \text{ if det } W \approx T_0, \quad \text{where } W \text{ is a tree whose root, labelled with, has left and right subtree isomorphic to } T_0, T_1 \text{ respectively. Notice that we are leaving a layer of formal details implicit, regarding the criteria to be followed for choosing a canonical representative of each } \omega \text{-class: had such criteria been stated, representatives could be taken as our rational ground terms proper.}

Let us now broaden the discourse by adjoining to our former signature \( \Sigma \) a denumerably infinite collection \( V \) of new symbols of arity \( 0 \), named variables. Labelled graphs, and in particular terms, whose labelling may involve variables, or that may violate the above-stated groundness restriction, will be said to be hollow. As will emerge from Sec. 4, every hollow, rooted and finite graph depicts a collection of ground terms, obtainable from it via substitutions.

As illustrated by Fig. 2 and 3, any ground labelled graph \( \mathcal{G} \) (possibly with cycles) can be variously rendered, up to isomorphism, as a conjunction (singleton when \( \mathcal{G} \)
The symbols $x, y, z, u, v, x_1, z_1^j, y_1$ will stand for distinct variables implicitly universally quantified in front of each axiom.

We begin with the extensionality axioms, according to which any two entities that have the same color and the same elements are equal. Formally

$$\forall x (x \in z \iff x \in y) \Rightarrow \text{color.of}(z) = \text{color.of}(y) \Rightarrow z = y.$$  

Then we have axioms concerning the empty entities named colors (in particular the null set $\emptyset$) and the element adjunction with and element removal less operations.

$$\begin{align*}
(Z_{0,1}) & \quad \forall x \in \text{color.of}(w) \\
(W_{0,1}) & \quad \forall x \in w \iff (x \in u \lor x = y) \\
(L_{0,1}) & \quad \forall x \notin u \iff (x \in v \land x \neq y) \\
\end{align*}$$

The anti-diagonal and self-loop axioms below ensure, for example, that for any color $y$ and any tuple $v_1, \ldots, v_m$ of hypersets, the system

$$x \notin v_1 \land \cdots \land x \notin v_m \land x \notin z \land \text{color.of}(z) = y$$

of constraints, as well as the equation

$$z = (\cdots(y \text{with } v_1) \cdots \text{with } v_m) \text{with } z,$$

can be satisfied (one independently of the other).

$$\begin{align*}
(D_{a}) & \quad \forall x (x = \text{anti-diagonal}(u, y) \land (x \in u \lor x = y)) \Rightarrow \text{color.of}(z) = \text{color.of}(y) \land z \notin v \\
(D_{b}) & \quad \forall x (x = \text{self-loop}(u, y) \Rightarrow \text{color.of}(z) = \text{color.of}(y) \land (x \in z \iff (x \in u \lor x = y)))
\end{align*}$$

We are arriving at the axioms essentially expressing our own weak version of Aczel's anti-foundation axiom AFA (see [1]). We will denote such axioms as anti-regularity (H) and hyper-extensionality (H). To informally introduce these two schemes, let us consider a system

$$\forall j \geq 0 \exists x_j \equiv \{x_{j1}, \ldots, x_{jm_j}\},$$

where $n \geq 0$, $m_j \geq 0$ for all $j$, $x_0, \ldots, x_n$ are distinct variables, each $x_{ij}$ is one of $x_0, \ldots, x_n$, and the $j$-th 'congruence' of the system is a short for $\forall z (z \in x_j \iff \bigwedge_{k=1}^{m_j} z = x_{jk})$. Anti-regularity states that each such system admits a solution with pre-assigned colors for the $x_{ij}$s. Hyper-extensionality states that the solution is uniquely determined by the colors, even if we consider a more general system of congruences

$$\forall j \geq 0 \exists x_j \equiv \{x_{j1}, \ldots, x_{jm_j}\} \; (\text{mod } z_j),$$

where $z$ is a fixed ' residue' (in the previous system, $z_j \equiv \emptyset$ for all $j$).

In the formal specification below, we employ distinct auxiliary variables $x_{00}, \ldots, x_{no},$ $x_{10}, \ldots, x_{1n}$, and convene to indicate by $x_{1j1}^{(k)}$, $x_{1jm_j}$ the variables $x_{jk}$ such that $x_{jk}$ occurs inside the right-hand side of the $j$-th congruence.
Let us incidentally observe that (R) is a sort of weak form of Aczel's AFA1, while
(H) corresponds to AFA2.

The following five axioms are Clark's freeness assumptions (cf. [4]), an adaptation of the weak domain closure assumption (cf. [14]), and a statement, antithetic to the occur-check scheme (cf. [4, 14]), which in a sense generalizes (R).

Here we assume that our signature $\Sigma$ contains solely $\emptyset$ with and the free functors mentioned in the preceding sections. In particular less, color.of, and all other functors introduced to state the axioms in this section, are not in $\Sigma$. We are to assume that $f$, $g$, and all $f, g$ appearing below, belong to $\Sigma$; also, $f$ is to be distinct from both $g$ and $h$. Finally, let $A = \max \{ ar(h) : h \in \Sigma, h \neq \emptyset \}.

Every instance of (OC) results from a finite cyclic graph $G$ labelled over $\Sigma$. The variables $x_{v_0}, \ldots, x_{v_n}$ are in one-to-one correspondence with the nodes $\nu_0, \ldots, \nu_n$ of $G$; we are indicating by $f$ the label of the node $\nu$, by $n$, the arity of $f$, and by $\mu_j^\nu$ the $j$-th son of $\nu$.

Remark. To obtain from the preceding theory of hypersets a corresponding theory of sets, we should drop (D$_E$), (R) and (OC), adopting a classical regularity axiom and a suitable version of occur-check scheme of axioms (cf. [6, 13]).

4 Unification Algorithm
The following notation is used below. Capital letters $X, Y, Z$, etc. represent variables; $f, g$, etc. stand for functional symbols (i.e. elements of $\Sigma$); $\equiv$ denotes the syntactic identity relation between first-order terms over $\Sigma \cup \{ \}$. $\forall \varphi$ denotes the result of replacing every occurrence of the variable $X$ by $Y$ in a quantifier-free first-order expression $\varphi$, and $\text{vars} (\varphi)$ denotes the set of all variables occurring in $\varphi$; $\text{dom} (\lambda)$ and $\text{ran} (\lambda)$ denote the domain and range of a mapping $\lambda$. We will only need substitutions of the following kind:

Definition 5 A (ground) substitution is a mapping $\gamma$ from a finite subset of $V$ to the universe $G$.

One can apply a substitution $\gamma$ to a hollow graph $G$, thereby obtaining a ground $G^{\gamma}$, when $\text{dom} (\gamma) \supseteq \emptyset \cap \text{ran} (\gamma)$: to do that, one will graft an isomorphic copy of $\gamma(X)$ in place of each node $\nu$ labelled $G(\nu) = X$, for all $X$ in $\text{dom} (\gamma)$. After observing that every first-order term $t$ is the concrete rendering of an acyclic rooted graph $G_t$ labelled over $\Sigma \cup \{ \}$ (see ending remarks of Sec. 2), one realizes that the notation $\gamma$ makes sense too, provided $\text{dom} (\gamma) \supseteq \text{vars} (t)$. Thus we are ready to define:

Definition 6 A solution to a Herbrand system $E$ is a substitution $\gamma$ that solves all equations in $E$ at once. That is, for all $\ell = r$ in $E$, both $\text{dom} (\gamma) \supseteq \text{vars} (\ell) \cup \text{vars} (r)$ and $\ell \equiv \gamma (r)$.\hfill $\Box$

There are systems of equations of special forms for which a solution can be determined quite easily.

Definition 7 A Herbrand system $E$ is said to be in solvable form if each equation in it has one of the forms:

1. $X = Y$ and $X$ does not occur elsewhere in $E$;
2. $X = f(Y_1, \ldots, Y_n)$, $f$ different from with, or $X = V$ with $Y, V$ distinct from $X$, and $X$ does not occur as left-hand side of any other equation in $E$.

A solution $\gamma$ of a Herbrand system $E$ in solvable form can be computed in the following way:

1. if $X = Y$ in $E$ then $\gamma(X) = \gamma(Y)$;
2. if $X = f(Y_1, \ldots, Y_n)$ in $E$ then $\gamma(X) = T$, where $T([i]) = f$, and $T([i]) = \gamma(Y_i)$ for $i = 1, \ldots, n$;
3. if $X = V$ with $V (V \neq X)$ is in $E$ then $\gamma(X) = T$, where $T([i]) = \gamma(V)$, and $T([n]) = \gamma(Y)$;
4. if $X = V$ with $V$ is in $E$, for $i = 1, \ldots, n$ in $E$, then $\gamma(X) = T$, where $T([i, \ldots, i]) = \gamma(V)$ for
   - $i = 1, \ldots, n;
   - T([i, \ldots, i]) = \emptyset$;
   - $T([1, \ldots, n, 1, 2]) = \gamma(Y_{i+1})$, for $i = 0, \ldots, n - 1$;
5. $\gamma(X) = T_\emptyset$, where $T_\emptyset([i]) = \emptyset$, whenever $\gamma(X)$ has not been defined by 1-4.

If the graph $G$ obtained by combining all the $\gamma(X)$s for $X$ in $\text{vars} (E)$ does not contain any sequence $v_0, v_1, \ldots, v_n$ of nodes with $G(\nu_i) = \gamma(V)$, for all $i \in \{0, \ldots, n \}$ such that $G[\nu_0] \supseteq \{ \gamma(V), v_0 \}$ (groundness restriction), then $\gamma$ is a solution for $E$. Otherwise a solution $\gamma$ for $E$ can be obtained from $\gamma$ by modifying $G$ so that $G[\nu_0] = T_\emptyset$ and collapsing all the variables $X$ such that $\gamma(X) = G[\nu_0]$ for some $i \in \{0, \ldots, n \}$ into $G[\nu_0]$. We are now ready to state the unification problem in very specific terms. System in solvable form are, in fact, so explicit that we can employ them as 'moulds' of the solutions to a given system:
Definition 8 Given a Herbrand system $E$, solving $E$ amounts to producing a finite set of pairs $(E_1, C_1), \ldots, (E_m, C_m)$ with $E_i$ a Herbrand system in solvable form and $C_i$ a conjunction of inequalities, such that

- for every solution $\gamma$ of $E$, at least one of the $E_i$s has a solution $\sigma$ such that $\gamma(X) = \sigma(X)$ for all $X$ in $\text{vars}(E)$, and all the inequalities in $C_i$ are satisfied by $\gamma$;
- for any solution $\gamma$ of any of the $E_i$s, satisfying $C_i$, every substitution $\sigma$ such that $\text{dom}(\gamma) \supseteq \text{vars}(E)$ and $\gamma(X) = \sigma(X)$ for all $X$ in $\text{vars}(E) \cap \text{dom}(\sigma)$, is a solution of $E$.

The algorithm Unify to be described next, gets an input $E$ which, without loss of generality, is assumed to be flat (cf. Def. 4).

Unify performs a non-deterministic search. Reaching the leaf of a successful branch of the search tree, it will output a pair $(E_i, C_i)$ as in Def. 8. The whole search tree will be finite and while computing $(E_i, C_i)$ an auxiliary Herbrand system $E'$ gathering redundant information will be produced. The system $E'$ can be used for further optimizations in the search of the solutions.

Special chains of inclusions, introduced by the following definition, will play an important role in our subsequent discussion:

Definition 9 A path through a Herbrand system $E$ is a sequence $X_0 \rightarrow X_1 \rightarrow \ldots \rightarrow X_n$ such that $X_{i+1} = X_i$ with $Y_i$ is in $E$ for all $i$ in $\{0, \ldots, n\}$.

Unify($E$: Herbrand system; $E'$: $\emptyset$; $C': \emptyset$; repeat
1. for each path $X_0 \rightarrow X_1 \rightarrow \ldots \rightarrow X_n \rightarrow X_0$ in $E \cup E'$, do $E := E \cup \{X_i = X_0 : i \in \{1, \ldots, n\}\}$
2. for each equation $e \equiv X = X$ in $E$, do $E := E \setminus \{e\}$
3. for each equation $e \equiv X = Y$ in $E$ such that $X$ occurs somewhere else in $E \cup E' \cup C$, do $E := E \setminus \{e\} \cup \{X = Y\}$; $E' := E' \setminus \{e\}$; $C := C \setminus \{e\}$
4. for each $U \neq V$ or $V \neq U$ in $C$, infer new inequalities:
   - for each $U \rightarrow Y$, $V \rightarrow Y$, $X$ in $E \cup E'$, do $C := C \cup \{U \neq X\}$, and
   - for each $X \rightarrow Y$, $U \rightarrow V$ in $E \cup E'$, do $C := C \cup \{V \neq X\}$
5. if $X \neq X$ in $E$ then exit with failure;
6. for each $X_n \neq X_{n+1}$ or $X_{n+1} \neq X_n$ in $C$, if there is a path $X_0 \rightarrow X_1 \rightarrow \ldots \rightarrow X_n \rightarrow X_{n+1}$, $X_n \neq X_{n+1}$ then exit with failure;
7. select arbitrarily an equation $e$ in $E$ (if any) enabling one of the following actions:
   - $e \equiv X = f(X_1, \ldots, X_n), f \neq \text{with}$, and there is an $e' \equiv X = f(Y_1, \ldots, Y_n)$ in $E$: $E := (E \setminus \{e\}) \cup \{X_1 = Y_1, \ldots, X_n = Y_n\}$;
   - $e \equiv X = f(X_1, \ldots, X_n)$, and there is an $e' \equiv X = g(Y_1, \ldots, Y_m)$ in $E$ with $f \neq g$: exit with failure;
   - $e \equiv X = V$ with $Y$, and there is an $e' \equiv X = W$ with $Z$ in $E$ such that either $V \neq X$ or $W \neq X$: perform one of the following actions:
     - (a) assume $V = X$, $W = X$, and either $Y = Z$ or $Y \neq Z$, that is, do either

$$E := (E \setminus \{e\}) \cup \{V = X, W = X, Y = Z\} \text{ or }$$

$$E := E \cup \{V = X, W = X\}; C := C \cup \{Y \neq Z\}$$

(b) assume $V = X$, $W \neq X$, and either $Y = Z$ or $Y \neq Z$, that is, do either
   - i. $E := (E \setminus \{e\}) \cup \{V = X, Y = Z\}; C := C \cup \{X \neq W\}$ or
   - ii. $E := (E \setminus \{e\}) \cup \{V = X, W = Y\}; C := C \cup \{X \neq W, Y \neq Z\}$

(c) assume $X \neq V$, $W = X$, and either $Y = Z$ or $Y \neq Z$, that is, do either
   - i. $E := (E \setminus \{e\}) \cup \{W = X, Y = Z\}; C := C \cup \{X \neq V\}$ or
   - ii. $E := (E \setminus \{e\}) \cup \{W = X, V = W\}; C := C \cup \{X \neq V, Y \neq Z\}$

(d) assume $X \neq V$, $X \neq W$, and either $Y = Z$ or $Y \neq Z$, that is, do either
   - i. $E := (E \setminus \{e\}) \cup \{Y = Z, W = V\}; C := C \cup \{X \neq Y\}$ or
   - ii. $E := (E \setminus \{e\}) \cup \{V = N \neq W, Z = W\}; C := E' \cup \{e'\}$

\[(N \text{ a new variable, that intuitively stands for } V \neq W)\]

until $E$ is in solvable form;
exit with success returning $(E, C)$.

It goes without saying that actions 1 and 4 are performed without falling into naive loops. For instance action 1 should be meant as saying:

while there is a path $X_0 \rightarrow X_1 \rightarrow \ldots \rightarrow X_n \rightarrow X_0$ in $E \cup E'$ such that $X_i = X_0$ is not in $E$ for some $i \in \{1, \ldots, n\}$, do $E := E \cup \{X_i = X_0\}$.

4.1 Termination Proof
In sight of proving that every branch of the search tree of Unify eventually breaks down, let us notice that 7(d)ii is the only action introducing new variables. Furthermore, calling element variable any variable that occurs on the right of with inside $E$, note that every variable that eventually plays the role of element variable, either held that role from the beginning, or has become element variable due to a substitution performed by action 3.

Lemma 1 Given a flat system $E$, Unify($E$) terminates unless action 7(d)ii is performed an infinite number of times.

Suppose that the algorithm does not terminate; then, by Lemma 1, there must be a situation which can be depicted as in figure 4. Each variable $N_{i+1}$ has been called into play by action 7(d)ii due to the presence in $E$ of the equations $N_i = V_i$ with $Y_i, N_i = W_i$ with $Z_i$. Observe that every arrow $H \rightarrow T$ in this figure represents the inclusion relation $H \subseteq T$, whose strictness ensues from constraints introduced by action 7(d)ii.

Lemma 2 In the figure 4: (i) for each $i, Y_i \neq Z_i$ is in $C$, and (ii) for every pair $i, j$ with $i \neq j$, it cannot be the case that $A_i \equiv B_j$ with $A_i, B_j \in \{Y_i, Y_j, Z_i, Z_j\}$.

Lemma 3 If $N$, a variable introduced by action 7(d)ii, appears on the right of an occurrence of with after the body of repeat has been performed $k$ times, from that moment on $A = N \in E$, for some $A$ occurring in the initial value of $E$.

Theorem 1 Unify($E$) terminates for any flat system $E$. 

Proof. Suppose that the algorithm does not terminate for some $E$. By Lemma 1, action $T(4)\|i$ has been performed an infinite number of times, generating an infinite path of inclusions as in the figure. By Lemma 2, all variables $Y_1$ and $Z_1$ are distinct. By Lemma 3, we derive a contradiction, since the number of pairwise distinct $Y$s and $Z$s must be less than or equal to the number of initial variables.

The above termination proof contains the main ingredient for a complete study of the complexity of our algorithm. In fact, one can easily show that the unification problem we are dealing with is NP-hard (see, for instance, [6]). On the other hand, the following results, based on the ideas of the above termination proof, allow one to show that the problem can be solved in polynomial time by a suitable non-deterministic Turing machine.

Given a flat Herbrand system $E$, let $v$ be the number of unsolved variables in $E$ and let $s = \text{size}(E)$.

Lemma 4 Suppose that action $T(4)\|i$ is performed exactly $k$ times along the branch corresponding to a non-deterministic computation of $\text{Unify}(E)$. Then such branch has length at most $O(v^2k + (v + s)k^2 + k^3)$.

Lemma 5 The number of generated variables in a successful branch of $\text{Unify}(E)$ is $O(s^3)$.

Theorem 2 Let $E$ be a flat Herbrand system. Then every single branch generated during the execution of $\text{Unify}(E)$ has length polynomial in $s + v$.

4.2 Soundness and Completeness

In this section, we will prove the soundness and completeness of the algorithm we have presented. More specifically, we will show that the axiomatic set theory introduced in Sec. 3 is a minimal theory in which the correctness and completeness proof can be carried out.

Theorem 3 Let $(E_1, C_1), \ldots, (E_k, C_k)$ be the solutions returned by $\text{Unify}(E)$. Then

\[ T \vdash \forall (E) \left( \exists N_1 \cdots \exists N_k, (E_1 \wedge C_1, \ldots, E_k \wedge C_k) \right), \]

where $T$ is the finite theory consisting of the following axioms: $(E), (L), (W), (=), (F_0), (F_1)$.

Remark. The equations in $E'$ are used in steps 1 and 4. $E'$ itself can be seen as a global data structure carrying information about mutual (strict) inclusion among variables. As we already observed it is not difficult to see that the constraints in $E'$ follow, in $T$, from those in $E \cup C$. However, notice that in order to apply the tests in steps 1 and 4 exactly in the (simple) way in which they are stated, $E'$ is necessary.

As explained earlier, the input Herbrand system $E$ will be provably equivalent (cf. Thm. 3) to the disjunction $\bigvee_{i=1}^k \exists N_1 \cdots \exists N_k ((E_i \wedge C_i))$, where $N_1, \ldots, N_k$ denote all variables, present in $E_i \wedge C_i$, but not in the original $E$, introduced along the $i$-th branch) taken over all the pairs $(E_i, C_i)$ that result from the successful branches. Somewhat disturbingly, such conjunctions are not guaranteed to be useful in general. Consider the following example: let $E = \{ X = E_1 \wedge Y = E_2 \wedge X = E_3 \wedge Y, E_1 = f(Z), E_2 = f(Z) \}$ from which, by applying action $T(4)\|i$, we obtain $E_i = \{ X = E_1 \wedge Y, E_1 = f(Z), E_2 = f(Z), \ldots \}$. $C_i = \{ X \neq Y, \ldots \}$ which contradicts $\text{AFA}$ (in our restricted axiomatization it contradicts (H)). To eliminate such useless disjuncts, we can apply the decision test present in Sec. 6 of [12]. But, the algorithm in [12] works only in a context in which free functors are not allowed. Therefore, given $(E_i, C_i)$, we test the following formula, implicitly keeping into account the freeness axioms:

\[ C_i \wedge \bigwedge_{f \in \Sigma(\text{with})} \left( Y \neq X \wedge W(E) \wedge \mathcal{D}(E) \wedge \bigwedge_{f \in \Sigma(\text{with})} (\epsilon(E_f) \wedge X = Y \wedge X = f(Y, \ldots, Y)) \right), \]

where:

- $W(E) = \text{def} \bigwedge_{X \neq Y} (Y \in X \wedge X \notin X \wedge X \in Z \wedge Z \in V \wedge V = Y);$ 
- for every $f \in \Sigma(\text{with})$ let $\epsilon(E_f) = \text{def} \in E : e : e, e$, $f(Y, \ldots, Y)$.

Let $E_f = \{ X = f(Y_1(i), \ldots, Y_n(j)), X = f(Y_1(i), \ldots, Y_n(j)) \}$, then

\[ \epsilon(E_f) = \text{def} \bigwedge_{i, j \in \{1, \ldots, k\}} \left( \bigwedge_{i=1}^n \left( Y^{(i)}_1 = Y^{(j)}_1 \right) \wedge X_i = X_j \right) ; \]

- $\mathcal{D}(E) = \text{def} \{ X \neq Y \wedge X = f(V_1, \ldots, V_m), Y = g(Z_1, \ldots, Z_n) \in E, f \neq g \}$.

5 A Direct Application

One of the first applications of hypersets that were proposed (cf., e.g., [3]), was as a means to model (deterministic) finite state automata. Below we show that the notions we have studied in this paper are sufficiently powerful to offer support to such modelling task.

A deterministic finite automaton (DFA for short) consists of a set $Q = \{ Q_0, \ldots, Q_n \}$ of states, a set $S = \{ s_1, \ldots, s_k \}$ of symbols, and a transition function $d : Q \times S \rightarrow Q \cup \{ \bot \}$. One of the states — say $Q_0$ — is called initial state, and there is a set $F \subseteq Q$ of accepting states (for a complete definition of DFAs see, for instance, [7]).

Given a DFA $A$, one may define a corresponding Herbrand system $E_A$ in the signature $\Sigma = \{ \bot \text{with}, \bot, \delta, \delta \}'$, where $\bot$ is a constant symbol and $\delta, \delta'$ are functional
symbols of arity $k$, as follows:

$$E_A = \{A = \emptyset \text{ with } Q_0 \text{ with } \ldots \text{ with } Q_n\} \cup
\{Q_i = \delta(d(Q_i, s_1), \ldots, d(Q_i, s_k)) : Q_i \in Q \setminus F\} \cup
\{Q_i = \delta'(d(Q_i, s_1), \ldots, d(Q_i, s_k)) : Q_i \in F\}.$$ 

This can easily be re-expressed an equivalent flat system, or, if one prefers, as a graph bearing the same information.

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

![DFAs](image)

where the sets of accepting states for $A$ and $B$ are $F_A = \{q_1, q_2\}$ and $F_B = \{q'_1\}$, respectively. In our language these automata can be modelled by the following graphs:

![Graphs](image)

or by means of the two systems $E_A = \{A = X_1 \text{ with } Q_0, X_1 = X_2 \text{ with } Q_1, X_2 = X_3 \text{ with } Q_2, X_3 = \emptyset, Q_0 = \delta(Q_0, 1), Q_1 = \delta'(Q_2, Q_1), Q_2 = \delta'(Q_2, Q_1)\}$, $E_B = \{B = Y_1 \text{ with } Q_0, Y_1 = Y_2 \text{ with } Q'_1, Y_2 = \emptyset, Q'_0 = \delta(Q'_1, 1), Q'_1 = \delta'(Q'_2, Q'_1)\}$.

The function call $\text{Unify}(E_A \cup E_B \cup \{A = B, Q_0 = Q_0, F = E_1 \text{ with } Q_2, E_1 = E_2 \text{ with } Q_1, F = E_2 \text{ with } Q'_1, E_2 = \emptyset\})$ has (at least) one satisfactory branch, reporting the system in solvable form $\{Q_1 = Q_2, Q'_1 = Q_2, \ldots\}$; hence the two automata are unifiable (as holds between their graphs). Notice that bisimulability between graphs (cf. Def. 2), $\approx$, corresponds exactly to equivalence between automata (see [3]).

**Acknowledgements**

We are grateful to Giorgio Levi who encouraged us in pursuing to an end the research reported about in this paper. Davide Aliffi actively contributed to the early phases of the discussion. Enrico Pontelli took part in our attempts to prove the termination of a preliminary version of the unification algorithm. Marino Miculan gave us invaluable advice concerning \LaTeX.
1 Introduction

1.1 Motivations

A logic program is, by all means, a first order theory. It is therefore legitimate to ask ourselves: which is the first order language underlying a logic program? This issue is important for several reasons.

- Most of (declarative) semantics are affected by the choice of the language. For instance, the least Herbrand model and the least term model semantics are language-dependent.

- In modular programming, a module refers to objects that are generally defined in other modules.

- A query may use symbols that do not occur in the program.

Consider the least Herbrand model semantics. Although it is generally accepted as the standard meaning of a pure logic program, it is inadequate to address such issues like compositionality, modularity, and verification. To see this point, let us adhere for a moment to the assumption of Apt [Apt90] and Lloyd [Llo87], where the language underlying a logic program is formed by the symbols occurring in it (with the proviso that a constant is added if none occurs in the program.) When reasoning on a program in isolation, this choice is natural and fruitful for many purposes. But if later it is needed either to extend the program by adding clauses with new function symbols, or to evaluate a query where new symbols occur, no support is available for reasoning in an incremental way, as we have to reconsider the overall semantics from scratch.

A brute force solution to this problem is to fix a common language $L$ in which every program of interest is written. This is the approach taken by Apt and Pedreschi in [AP94]. This leads to a semantics that solves the modularity problem, but causes unnecessary complications when reasoning about a program in isolation, as some unimportant details must be taken into account.

In this paper, we propose to use a combination of the mentioned two approaches, which retains the advantages of both while avoiding the drawbacks, and reveals useful to support modular reasoning in termination proofs. We assume that a program $P$ is written with reference to any language that extends $L$. More generally, we study termination (and semantics) properties with reference to any language which extends $L$, and show that certain useful properties holds independently of the underlying language. On this basis, we provide natural proof obligations to preserve the desired properties when combining programs together in a modular way. As a consequence, it is possible to consider $L$ only when reasoning on a program $P$ in isolation, and to directly reuse such proofs when performing modular reasoning.

We concentrate here on termination properties of logic programs. A query $Q$ universally terminates if every SLD-tree with root $Q$ is finite. $Q$ existentially terminates if there exists either a computed answer or a finitely failed SLD-tree for $Q$. Bezem [Bez89] introduced a proof method for universal termination based on the notion of recurrent programs. Apt and Pedreschi [AP90] extended the method to pure Prolog programs, where LD-resolution is adopted, that is SLD-resolution with the leftmost selection rule. Their method is based on the notion of acceptable programs. Both the cited methods adopt the $L$ language. In this paper, we extend the cited results with reference to a generic extension of $L$.

As an application of our results, we obtain a proof method for termination which is better suited for modular reasoning, thus improving over the results of Apt and Pedreschi [AP94]. Moreover, it is desirable to reason with decidable semantics. In [Bez93] it is proved that recurrent programs compute all recursive functions, and in [Bez89, AP90] that the least Herbrand model semantics is decidable. As another application of our approach, we prove that also the least term model semantics and the $S$-semantics are decidable in the case of acceptable (and recurrent) programs.

1.2 Preliminaries

We use in this paper the standard notation of Apt [Apt90] and Lloyd [Llo87]. A (first order) language $L$ is a pair $(\Sigma_L, \Pi_L)$ of disjoint, non-empty sets: the set of function symbols $\Sigma_L$, and the set of predicate symbols $\Pi_L$. Every symbol in a language is assigned a non-negative arity. Given two languages $L = (\Sigma_L, \Pi_L)$ and $M = (\Sigma_M, \Pi_M)$, we say that $M$ extends $L$ if $\Sigma_M \supseteq \Sigma_L$ and $\Pi_M \supseteq \Pi_L$.

An atom is called pure if it is of the form $p(x_1, \ldots, x_n)$ where $x_1, \ldots, x_n$ are...
different variables. With \( N \) we denote the set of natural numbers and with \( Z \) that of integers. We use \( \text{Term}_L \) to denote the set of terms on \( L \), \( U_L \) the set of ground terms on \( L \), \( \text{Atom}_L \) the set of atoms on \( L \), \( B_L \) the Herbrand base on \( L \), and \( B_P \) the Herbrand base on \( L_P \). \( \text{rel}(A) \) denotes the relation symbol occurring in an atom \( A \), \( \{A\}_L \) the set of ground instances of \( A \) on \( L \), and \( \text{Vars}(A) \) the set of variables in \( A \). A relation \( P \) is defined in a program \( P \) if \( P \) occurs in the head of a clause from \( P \). Given programs \( P \) and \( Q \), we say that \( P \) extends \( Q \) if no relation defined in \( P \) occurs in \( Q \). Boldface letters denote sequences of atoms.

Finally, in this paper we concentrate on terminating and left terminating programs, according to the following definitions from Bezem [Bez89] and Apt and Pedreschi [AP90]:

**Definition 1** A program \( P \) is terminating (resp., left terminating) iff all its SLD-derivations (resp., LD-derivations) starting with a ground goal in \( L_P \) are finite.

All proof methods considered in this paper are sound and complete characterizations of the class of terminating or left terminating programs. Based on empirical investigations, it is commonly accepted that most (if not all) practical Prolog programs are left terminating (e.g., most pure programs in Sterling and Shapiro's book [SS86].)

## 2 Recurrent programs

The notion of a recurrent program was introduced by Bezem in [Bez89], where it is shown that a program is recurrent iff it is terminating. Therefore, the notion of a recurrent program gives rise to a complete method for proving termination. Here, we generalize such a notion w.r.t. any language which extends the language \( L_P \) of a program \( P \).

**Definition 2** Let \( P \) be a program, and \( L \) be a language which extends \( L_P \).

- A level mapping w.r.t. \( L \) is a function \( |: B_L \rightarrow N \) of ground atoms (in \( B_L \)) to natural numbers. \( |A| \) is called the level of \( A \).

- \( \text{ground}_L(P) \) denotes the set of ground instances on \( L \) of clauses from \( P \).

In the sequel, we write \( \text{ground}(P) \) as an abbreviation for \( \text{ground}_{L_P}(P) \).

**Definition 3** Let \( P \) be a program, and \( L \) be a language which extends \( L_P \). Then

- \( P \) is recurrent w.r.t. \( L \) by \( |: B_L \rightarrow N \) iff, for every clause \( A \leftarrow A', B, B \) in \( \text{ground}_L(P) \)

\[ |A| > |B|. \]

- \( P \) is recurrent w.r.t. \( L \) by some level mapping \( |: B_L \rightarrow N \).

- \( P \) is recurrent iff it is recurrent w.r.t. \( L_P \).

Observe that our definition of recurrence coincides with that of Bezem [Bez89]. We now prove that the various notions of recurrence are equivalent. To this purpose, we introduce a notion of extension of a level mapping w.r.t. \( L \) on another language \( M \) which extends \( L \). The idea is to map any atom \( A \in B_M \) in an atom \( A' \in B_L \) by replacing every maximal subterm of \( A \) which has its principal functor in \( \Sigma_M \setminus \Sigma_L \) with some fixed term from \( U_L \).

**Definition 4** Consider two languages \( L, M \) such that \( M \) extends \( L \), and a term \( t \in U_L \).

- Given a term \( u \in U_M \), \( H(u) \) denotes a term \( u' \in U_L \) obtained from \( u \) by replacing with \( t \) all maximal subterms of \( u \) with principal functor in \( \Sigma_M \setminus \Sigma_L \).

- Analogously, given an atom \( A \in B_M \), \( H(A) \) denotes an atom \( A' \in B_L \) obtained from \( A \) as above.

- Given a level mapping \( |: B_L \rightarrow N \), an extension of \( | \) on \( M \) by \( t \) is any level mapping \( |' : B_M \rightarrow N \) satisfying the following:

\[ |' A' | = | H(A) | \]

for any \( A \in B_M \) such that \( \text{rel}(A) \in \Pi_L \).

With reference to given languages \( L \) and \( M \) such that \( M \) extends \( L \), and term \( t \in U_L \), the mapping \( H \) can be equivalently specified as follows, for any term \( f(t_1, \ldots, t_n) \in U_M \) and atom \( p(t_1, \ldots, t_n) \in B_M \):

\[
H(f(t_1, \ldots, t_n)) = \begin{cases} f(H(t_1), \ldots, H(t_n)) & \text{if } f \in \Sigma_L \\ t & \text{if } f \in \Sigma_M \setminus \Sigma_L \end{cases}
\]

\[
H(p(t_1, \ldots, t_n)) = p(H(t_1), \ldots, H(t_n)).
\]

The choice of the term \( t \in U_L \) in the definition of the mapping \( H \) is immaterial to the purpose of the achieved results; we therefore omit to explicitly mention it in the sequel. The following lemma points out two immediate properties of the mapping \( H \) and of extended level mappings.

**Lemma 2.1** Let \( L, M \) be two languages such that \( M \) extends \( L \), \( A \) an atom in \( L \), and \( \theta \) a ground substitution such that \( A \theta \in B_M \). Then

(i) \( H(A \theta) = A \theta^H \) where \( \theta^H = \{ z/H(u) | z/u \in \theta \} \)

(ii) \( |A \theta^H|' = |A \theta^H| \)

The next result shows that all the proposed notions of recurrence w.r.t. different languages are equivalent.
Theorem 2.2 Let $P$ be a program. Then the following statements are equivalent:

(i) $P$ is recurrent w.r.t. some language that extends $L_P$,

(ii) $P$ is recurrent,

(iii) $P$ is strongly recurrent.

Proof. To prove (i) $\Rightarrow$ (ii) it suffices to consider the restriction of the level mapping for $P$ on $B_P$. (iii) $\Rightarrow$ (i) is immediate. We now prove (ii) $\Rightarrow$ (iii).

Let $P$ be a recurrent program w.r.t. $L_P$ by $| B_P | \rightarrow N$, and $L$ a language that extends $L_P$. Let $f : B_L | N$ be an extension of $|$ on $L$, and $C_0 = (A \leftarrow A, B, B) \in \text{ground}_L(P)$, where $C$ is a clause of $P$. We calculate:

$$A^H | = \begin{cases} | \text{Lemma 2.1(ii)} | \\ | A^H | \\ | \text{Lemma 2.1(ii)} | \\ | B_0^H | \\ | \text{Lemma 2.1(ii)} | \end{cases} = | B_0^H |.$$

The previous theorem justifies the fact that the termination proof for a program $P$ can be constructed with reference to the language $L_P$ of the symbols occurring in $P$ only. In fact, the previous result shows how such a proof can be readily extended to any language which extends $L_P$. In this sense, a proof method based on this result is more geared to support modular reasoning.

A more direct way of defining the extension of a level mapping can be given considering compositional level mappings. We say that a level mapping $| |$ w.r.t. $L$ is compositional if, for any symbol $s$ in $L$ of arity $k$, there exists a function $s : N^k \rightarrow N$ such that, for $p \\ s \\ \in \Sigma_L$:

$$| p(t_1, \ldots, t_k) | = \bar{s}(|t_1|, \ldots, |t_k|)$$

where $| | : U_L \rightarrow N$ is such that, for $f \in \Sigma_L$:

$$| f(t_1, \ldots, t_k) | = \bar{f}(|t_1|, \ldots, |t_k|).$$

In this case, an extension of $| |$ on $M$ is defined as follows:

$$| p(t_1, \ldots, t_k) |' = \begin{cases} \bar{s}(|t_1|', \ldots, |t_k|') & \text{if } p \in \Pi_L \\ 0 & \text{if } p \in \Pi_M \setminus \Pi_L \end{cases}$$

where $| |' : U_M \rightarrow N$ is defined as follows:

$$| f(t_1, \ldots, t_k) |' = \begin{cases} \bar{f}(|t_1|', \ldots, |t_k|') & \text{if } f \in \Sigma_L \\ m & \text{if } f \in \Sigma_M \setminus \Sigma_L \end{cases}$$

where $m$ is the minimum of $| |$ on $U_L$.

We note that most realistic programs can be proven terminating by using compositional level mappings. A direct application of the previous results is showed in the next simple example.

Example 1 Consider the program $\text{APPEND}$:

$\text{APPEND} = \{ X_1, Y_2, Z_3 \} \rightarrow \text{Z_3}$ is the result of concatenating the lists $X_1$ and $Y_2$.

$\text{APPEND} = \{ X_1 | Y_2, Y_3, [X] | Z_4 \} \rightarrow \text{APPEND}(X_1, Y_2, Z_3)$.

$\text{APPEND}$ is recurrent by defining:

$$| \text{APPEND}(X_1, Y_2, Z_3) |' = \min(\|X_1\|', \|Z_3\|').$$

where

$$\|X_1\|' = 0$$

and

$$\|X_1\|' = \|X_1\|' + 1.$$

Observe that this level mapping is compositional. Applying Theorem 2.2, we conclude that $\text{APPEND}$ is recurrent in every extension of $\text{APPEND}$ by an extended level mapping defined as follows:

$$| \text{APPEND}(X_1, Y_2, Z_3) |' = \min(\|X_1\|', \|Z_3\|').$$

where $\| |'$ is the list-length function of Ullmann and Van Gelder [UvG88]:

$$\| f(t_1, \ldots, t_n) |' = 0 \text{ for } f \neq [ . ],$$

$$\| f(t_1, \ldots, t_n) |' = \| t_1 \|' + 1.$$

This justifies the fact that the local termination proof for $\text{APPEND}$ can be reused when reasoning about larger programs. We note that this is a natural condition.

3 Semi-recurrent Programs

Semi-recurrent programs were introduced in Apt and Pedreschi [AP94] in order to obtain more modular termination proofs. We recall here the basic definitions, and show how the notion of semi-recurrence can be directly generalized in our framework.

Definition 5 Let $P$ be a program and $p, q$ relations in $\Pi_P$.

(i) We say that $p$ refers to $q$ in $P$ iff there is a clause in $P$ that uses $p$ in its head and $q$ in its body.

(ii) We say that $p$ depends on $q$ in $P$, and write $p \sqsupset q$, if $(p, q)$ is in the reflexive, transitive closure of the relation refers to.
(iii) We write \( p \preceq q \) iff \( p \supseteq q \) and \( q \supseteq p \).

(iv) We write \( p \supseteq q \) iff \( p \supseteq q \) and \( q \npreceq p \).

**Definition 6** Let \( P \) be a program and \( L \) a language which extends \( L_P \).

- \( P \) is semi-recurrent w.r.t. \( L \) by \( \mid : B_L \rightarrow N \) iff for every \( A \vdash A, B, B \) in \( \text{ground}_L(P) \):
  
  (i) \( |A| > |B| \) if \( \text{rel}(A) \preceq \text{rel}(B) \).
  
  (ii) \( |A| \geq |B| \) if \( \text{rel}(A) \supseteq \text{rel}(B) \).

- \( P \) is semi-recurrent w.r.t. \( L \) iff it is semi-recurrent w.r.t. \( L \) by some level mapping w.r.t. \( L \).

- \( P \) is semi-recurrent iff it is semi-recurrent w.r.t. \( L_P \).

- \( P \) is strongly semi-recurrent iff it is semi-recurrent w.r.t. every language that extends \( L_P \).

**Proof.** Analogous to Theorem 2.2.

**Corollary 3.2** A program is recurrent w.r.t. a language \( L \) iff it is semi-recurrent w.r.t. \( L \).

**Proof.** In [AP94] it is proved that a program is recurrent iff it is semi-recurrent.

We show some applications to modular logic programming, by extending the results of Apt and Pedreschi [AP94]. We thus obtain simple and general proof obligations for modular proofs of semi-recurrence. The aim is to reuse the proof of (semi-)recurrence of a subprogram in the proof for the whole program. A natural way to decompose a program into two subprograms \( P \) and \( Q \) is to choose them in such a way that \( P \) extends \( Q \). Indeed, this simplifies the proofs and allows for incremental reasoning.

- A level mapping \( \mid \) for \( P \cup Q \) is required to satisfy, for the clauses from \( Q \), the same conditions which already (an extension of, or a restriction of) a level mapping for \( Q \) satisfies.

- For all ground instances \( A \vdash A, B, B \) of a clause from \( P \) we have to fulfill conditions (i), (ii) of Definition 6. To this purpose, we propose to specify \( |A| \) as the sum of \( |A|_P \) and \( |A|_Q \), where \( |.\) is a function to integers. This function adds (or subtracts) to \( |A|_P \) a value to majorize \( |B|_Q \), when \( \text{rel}(B) \) is defined in \( Q \).

**Theorem 3.3** Let \( P \) and \( Q \) be programs such that \( P \) extends \( Q \), and \( L \) a language that extends \( L_P \cup Q \). Assume that:

1. \( P \) is semi-recurrent w.r.t. \( \Sigma_L, \Pi_P \) by \( \mid \).
2. \( Q \) is semi-recurrent w.r.t. \( \Sigma_L, \Pi_Q \) by \( \mid \).
3. there exists a function \( \mid : B(\Sigma_L, \Pi_P) \rightarrow Z \) such that

   - for every \( A \vdash A, B, B \) in \( \text{ground}_L(P) \)
     
     \( (a) \mid A \mid \geq \mid B\mid \) if \( \text{rel}(B) \) is defined in \( P \),
     
     \( (b) \mid A \mid \geq |B|_Q - |A|_P \) if \( \text{rel}(B) \) is defined in \( Q \),

   - for every \( A \in B_L \)
     
     \( (c) \mid A|_P + |A| \geq 0 \) if \( \text{rel}(A) \) is defined in \( P \).

Then \( P \cup Q \) is semi-recurrent w.r.t. \( L \) by \( \mid : B_L \rightarrow N \) defined as follows:

\[
|A| = \begin{cases} |A|_P + |A| & \text{if } \text{rel}(A) \text{ is defined in } P, \\ |A|_Q & \text{if } \text{rel}(A) \text{ is defined in } Q, \\ 0 & \text{otherwise} \end{cases}
\]

**Proof.** Consider \( A \vdash A, B, B \in \text{ground}_L(P \cup Q) \). If \( \text{rel}(A) \) is defined in \( Q \) then the conclusion is immediate. Otherwise, we consider the following cases.

**Case 1.** \( \text{rel}(A) \preceq \text{rel}(B) \).

By Definition 5, \( \text{rel}(B) \) is defined in \( P \):

\[
|A| = |A|_P + |A| \geq |B|_P + |A| \geq |B|_P + |B| = |B|.
\]

**Case 2.** \( \text{rel}(A) \supseteq \text{rel}(B) \).

If \( \text{rel}(B) \) is defined in \( P \) then the proof is analogous to that of the previous case. If \( \text{rel}(B) \) is neither defined in \( P \) nor in \( Q \), then \( |B| = 0 \), so the thesis holds trivially.

Finally, if \( \text{rel}(B) \) is defined in \( Q \):

\[
|A| = |A|_P + |A| \geq |B|_Q = |B|.
\]
Observe that condition (a) may be weakened by requiring:

\[ |A|_P + |A| > |B|_P + |B| \]

i.e., by directly applying Definition 6 to \(| |\). However, based on application, we found the proposed version for deriving \(| |\) more fruitful. Moreover, we note as condition (c) may be verified a fortiori.

### 4 Acceptable programs

The notion of an acceptable program was introduced by Apt and Pedreschi in [AP90], where it is shown that a program is acceptable iff it is left terminating. Therefore, the notion of an acceptable program gives raise to a complete method for proving left termination. Here, we follow the lines of Section 2, and generalize such a notion w.r.t. any language which extends the language \(L_P\) of a program \(P\).

**Definition 7** Let \(P\) be a program, \(L\) a language which extends \(L_P\), and \(I \subseteq B_L\) a Herbrand interpretation.

- \(P\) is acceptable w.r.t. \(L\) by \(| |\) if \(B_L \rightarrow N\) and \(I\) if \(I\) is a model of \(P\), and for every \(A \leftarrow A, B, B\) in \(\text{ground}_L(P)\):
  \[ I \models A \text{ implies } |A| > |B| \]  \((1)\)

- \(P\) is acceptable w.r.t. \(L\) iff it is acceptable w.r.t. \(L\) by some level mapping \(| |\) and interpretation \(I\) on \(L\).

- \(P\) is acceptable iff it is acceptable w.r.t. \(L_P\).

- \(P\) is strongly acceptable iff it is acceptable w.r.t. every language that extends \(L_P\).

It is clear that a program is recurrent w.r.t. a language \(L\) by a level mapping iff it is acceptable w.r.t. \(L\) by the same level mapping and \(B_L\). Notice that our notion of acceptability is slightly different from that of Apt and Pedreschi [AP90], as the latter uses general first order interpretations instead of Herbrand ones. We now provide a notion of extension of an interpretation w.r.t. a larger language, following the lines of Definition 4.

**Definition 8** Consider be two languages \(L, M\) such that \(M\) extends \(L\), an interpretation \(I \subseteq B_L\), and a term \(t \in U_L\). Let \(H\) the mapping of Definition 4.

The extension of \(I\) on \(M\) by \(t\) is a Herbrand interpretation \(J \subseteq B_M\) defined as follows:

\[ J = \{ A \in B_M \mid H(A) \in I \} \]

Observe that, by Lemma 2.1(i), if \(A\) is a sequence of atoms in \(L\) and \(A\theta\) a sequence of ground atoms in \(M\), then:

\[ J \models A\theta \iff I \models A\theta^M \]  \((2)\)

The next result shows that all the proposed notions of acceptability are equivalent, and coincide with the original notion of Apt and Pedreschi.

**Theorem 4.1** Let \(P\) be a program. The following statements are equivalent:

(i) \(P\) is acceptable w.r.t. some language that extends \(L_P\),

(ii) \(P\) is acceptable in the sense of Apt and Pedreschi,

(iii) \(P\) is acceptable,

(iv) \(P\) is strongly acceptable.

**Proof.** To prove (i) \(\Rightarrow\) (ii) it suffices to consider the restriction of the level mapping and the Herbrand interpretation on \(B_P\). (iv) \(\Rightarrow\) (i) is immediate. We now prove (ii) \(\Rightarrow\) (iii). We have to show that if (1) holds for a generic first order interpretation \(I\) then it holds for a Herbrand interpretation as well. Let \(P\) be acceptable w.r.t. \(L_P\) by \(| |\) and a generic first order interpretation \(I\) on \(L\). Consider the following Herbrand interpretation

\[ J = \{ A \in B_P \mid I \models A \} \]

Clearly, \(J\) is a model of \(P\), and for all \(A \leftarrow A, B, B \in \text{ground}(P)\):

\[ J \models A \Rightarrow I \models A \Rightarrow |A| > |B| \]

Finally, we prove (iii) \(\Rightarrow\) (iv). Let \(P\) be an acceptable program w.r.t. \(L_P\) by \(| |\) and \(I \subseteq B_P\), and \(L\) a language which extends \(L_P\). Let \(| |\) be an extension of \(| |\) on \(L\), \(J\) the extension of \(I\) on \(L\), and \(A\theta^M = (A \leftarrow A, B, B)\theta \in \text{ground}_L(P)\), where \(C\) is a clause of \(P\). To prove that \(J\) is a model of \(P\), we calculate:

\[ J \models (A, B, \theta) = \begin{cases} (2) \\ I \models (A, B, \theta)^M \{ I \text{ is a model of } P, \text{ and } A\theta^M \in \text{ground}(P) \} \\ (2) \end{cases} \]

Finally, if \(J \models A\theta\):

\[ |A\theta|' = \begin{cases} \text{Lemma 2.1(ii)} \end{cases} \]
Semi-acceptable programs

Semi-acceptable programs were introduced by Apt and Pedreschi in [AP94] in order to obtain modular proofs of left termination.

Definition 9 Let $P$ be a program and $L$ a language such that $L$ extends $L_P$ and $I \subseteq B_L$ a Herbrand interpretation.
- $P$ is semi-acceptable w.r.t. $L$ by $|I| : B_L \rightarrow N$ and $I$ iff $I$ is a model of $P$, and for every $A \rightarrow A$, $B$, $B$ in $\text{ground}_L(P)$ such that $I \models A$:
  (i) $|A| > |B|$ if $\text{rel}(A) \succeq \text{rel}(B)$,
  (ii) $|A| \geq |B|$ if $\text{rel}(A) \supset \text{rel}(B)$.
- $P$ is semi-acceptable w.r.t. $L$ iff it is semi-acceptable w.r.t. $L$ by some level mapping w.r.t. $L$ and $I \subseteq B_L$.
- $P$ is semi-acceptable if and only if it is semi-acceptable w.r.t. $L_P$.
- $P$ is strongly semi-acceptable if and only if it is semi-acceptable w.r.t. every language that extends $L_P$. □

As for recurrence and semi-recurrence, we have that the notions of acceptability and semi-acceptability coincide. We now upgrade Theorem 4.1 to acceptable programs.

Theorem 5.1 Let $P$ and $Q$ be programs such that $P$ extends $Q$, and $L$ a language that extends $L_P \cup Q$. Assume that:

1. $P$ is semi-recurrent w.r.t. $(\Sigma_L, \Pi_P)$ by $|I| : B_L \rightarrow N$ and $I$.
2. $Q$ is semi-acceptable w.r.t. $(\Sigma_L, \Pi_Q)$ by $|Q| : B_L \rightarrow N$.
3. There exists a function $|| | : B_{(\Sigma_L, \Pi_P \cup \Pi_Q)} \rightarrow Z$ such that
   - for every $A \rightarrow A$, $B$, $B$ such that $P \rightarrow P$,$\text{rel}(A) \in \Pi_P \cup \Pi_Q$ and $I \models A$ then
     (a) $|A| \geq |B|$ if $\text{rel}(B)$ is defined in $P$,
     (b) $|A| \geq |B| - |A_P|$ if $\text{rel}(B)$ is defined in $Q$.
   - for every $A \rightarrow A$, $B$, $B$ such that $P \rightarrow P$,$\text{rel}(A) \in \Pi_P \cup \Pi_Q$ and $I \models A$ then
     (c) $|A| + |A| \geq |A_P|$ if $\text{rel}(A)$ is defined in $P$.

Then $P \cup Q$ is semi-acceptable w.r.t. $L$ by $|I| : B_L \rightarrow N$ defined as follows:

$$|A| = \begin{cases} |A| + |A| & \text{if } \text{rel}(A) \text{ is defined in } P, \\ |A| & \text{if } \text{rel}(A) \text{ is defined in } Q, \\ 0 & \text{otherwise} \end{cases}$$ □

An immediate corollary relating semi-recurrent and semi-acceptable programs is the following.

Corollary 5.2 Let $P$ and $Q$ be programs such that $P$ extends $Q$, and $L$ a language that extends $L_P \cup Q$. Assume that:

1. $P$ is semi-recurrent w.r.t. $(\Sigma_L, \Pi_P)$ by $|I| : B_L \rightarrow N$ and $I$.
2. $Q$ is semi-acceptable w.r.t. $(\Sigma_L, \Pi_Q)$ by $|Q| : B_L \rightarrow N$.
3. There exists a function $|| | : B_{(\Sigma_L, \Pi_P \cup \Pi_Q)} \rightarrow Z$ such that
   - for every $A \rightarrow A$, $B$, $B$ such that $P \rightarrow P$,$\text{rel}(A) \in \Pi_P \cup \Pi_Q$ and $I \models A$ then
     (a) $|A| \geq |B|$ if $\text{rel}(B)$ is defined in $P$,
     (b) $|A| \geq |B| - |A_P|$ if $\text{rel}(B)$ is defined in $Q$.
   - for every $A \rightarrow A$, $B$, $B$ such that $P \rightarrow P$,$\text{rel}(A) \in \Pi_P \cup \Pi_Q$ and $I \models A$ then
     (c) $|A| + |A| \geq |A_P|$ if $\text{rel}(A)$ is defined in $P$.

Then $P \cup Q$ is semi-acceptable w.r.t. $L$ by $|I| : B_L \rightarrow N$ defined as follows:

$$|A| = \begin{cases} |A| + |A| & \text{if } \text{rel}(A) \text{ is defined in } P, \\ |A| & \text{if } \text{rel}(A) \text{ is defined in } Q, \\ 0 & \text{otherwise} \end{cases}$$ □

Proof. The result follows from the observation that $P$ is semi-acceptable w.r.t. $(\Sigma_L, \Pi_P)$ by $|I| : B_L \rightarrow N$ and $I$, and Theorem 5.1. □

Example 2 Consider the program $\text{SELECTS} = \text{SEL} \cup \text{SELS}$

$\text{SELECTS} (\text{IS} , \text{YS}) \leftarrow \text{The list IS is a subset of the list YS}$

$\text{SELECTS} ( \text{I} , \text{YS})$. 

$\text{SELECTS} ( \text{IS} \setminus \text{IS})$, $\text{YS} \leftarrow \text{select}(\text{I}, \text{YS}, \text{YS1})$. 

$\text{SELECTS}(\text{IS}, \text{YS})$.
selects(\(X, Y\)).
select(\(X, [X|X], X\)).
select(X, [Y | X], [Y | Y]) -> select(X, X, Y).

where SEL is the definition of select and SELS is that of selects. It is readily
checked that:
- SELS is recurrent by \(|\text{select}(X, Y)|_\text{SELS} = |X|\) where \(||\) is defined as in
Example 1
- SEL is recurrent by \(|\text{select}(X, X, Y)|_\text{SELS} = |X|\).

Unfortunately, program SELECTS is not recurrent, as it is not terminating: it admits
infinite derivations. We now use Corollary 5.2 to conclude that it is acceptable.
To this end, we need more information on the behavior of select. Noting that a
program \(P\) which is recurrent by a level mapping is (semi-)acceptable by the same
level mapping and any model of \(P\), we try to fulfill the conditions of Corollary 5.2 by
taking \(|P|_\text{SEL} = |\text{select}(X, X, Y)| = |X| + 1\). We have to find a function \(||\)
such that:
\[|Y| = |Y| + 1 \Rightarrow \{ |\text{select}(X, X, Y)| \geq |\text{select}(X, Y, Y)|,\]
\[|\text{select}(X, X, Y)| \geq |Y| - |X| \].

As suggested by the second constraint, we put
\[|\text{select}(X, Y)| = |Y| - |X|.

Note that \(||\) ranges on all integers. For \(|Y| = |Y| + 1\), we have:
\[|\text{select}(X, X, Y)| = |Y| - |X| - 1 = |Y| - |X| - |\text{select}(X, Y, Y)|].

Finally, we find that SELECTS is semiacceptable by \(\text{select}(X, Y) = \text{SELS} \cup J_\text{SEL} = |Y| - |X| + |X| = |Y|

We conclude this example by observing that we reached an expressive final level
mapping despite the initial mistake of choosing \(\text{select}(X, X, Y)|_\text{SELS} = |X|\)
instead of \(\text{select}(X, X, Y)|_\text{SELS} = |Y|\). In general, however, a modular method
cannot guarantee the optimal choice.

6 Decidability results

In this section, we study some recursion-theoretic aspects of the extensions of recurrence
and acceptability proposed in this paper. We show that most common declarative
semantics are decidable in the case of recurrent and acceptable programs. Bezem
For an acceptable program $P$, the sets $C_P$, $MC_P$ and $SP_P$ are recursive.

Proof. The fact that $C_P$ is recursive is an immediate consequence of Corollary 6.3. The fact that $MC_P$ is recursive follows from Corollary 6.3 and the observation that there are finitely many atoms (modulo renaming) which are more general of a given atom. We now prove that $SP_P$ is recursive, by providing a decision procedure for the problem $A \in SP_P$, for $A \in Atom_{P}$. Let $\theta$ be a substitution mapping all variables of $A$ into distinct new constants $c_1, \ldots, c_n$. The query $\theta A$ is ground and therefore, by Lemma 6.2, $\theta A$ has a finite (modulo renaming) set $\{e_1, \ldots, e_m\}$, $m \geq 0$, of (finite) D-refutations in $P$. Consider the pure atom $\theta A$ for $A$; by the Lifting Lemma [Apt90], for $i \in [1, m]$, there exists an L-refutation $\xi_i$ for $\theta A$, which uses the same sequence of clauses of $\xi_i$, with computed $A_i$. To conclude the proof, it suffices to prove the following:

Claim. $A \in SP_P$ iff $A$ is a renaming of $A_i$ for some $i \in [1, m]$.

Proof. The if part is trivial. To prove the only-if part, we notice that, by [Apt90, Lemma 3.20] or [Lio87, Lemma 8.5], if $A$ is a computed instance of $\theta A$ in $P$ with an L-refutation $\xi$, then $A$ is a computed instance of $A$ in $P$ with an L-refutation $\xi'$, which uses the same sequence of clauses of $\xi$. The conclusion then follows from the observation that, again by the same Lemma, $\theta A$ has an L-refutation $\xi''$ in $P$, which uses the same sequence of clauses of $\xi'$.

Indeed, the above proof implies the following slightly stronger result.

Corollary 6.5 Let $P$ be an acceptable program, and $Q, Q'$ queries. Then it is decidable if $Q'$ is a computed instance of $Q$ in $P$.

Final remarks

In this paper, we showed that proving universal (left) termination of a logic program is unaffected by the first order language underlying the program. On the basis of this fact, we proposed a proof method for universal termination which supports modular reasoning. As a by-product, we also obtained decidability results for several declarative semantics.

A natural direction for future research is to extend our method beyond (left) terminating programs. In fact, some programs are designed to behave properly only on certain ground queries, rather than on all ground queries. A generalization along these lines would hopefully lead to simpler termination proofs, and larger classes of programs with decidable semantics. Also, it is needed to clarify the relations with the other methods, based on pre- and post-conditions.