A Declarative Approach for First-Order Built-in's of Prolog

Krzysztof R. Apt
CWI
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands
and
Faculty of Mathematics and Computer Science
University of Amsterdam, Plantage Muidergracht 24
1018 TV Amsterdam, The Netherlands

Elena Marchiori
CWI
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

Catuscia Palamidessi
Dipartimento di Informatica e Scienze dell' Informazione
Università di Genova, Viale Benedetto XV 3
16132 Genova, Italy

Abstract

We provide here a framework for studying Prolog programs with various built-in's that include arithmetic operations, and such metalogical relations like *var* and *ground*. To this end we propose a new, declarative semantics and prove completeness of the Prolog computation mechanism w.r.t. this semantics. We also show that this semantics is fully abstract in an appropriate sense. Finally, we provide a method for proving termination of Prolog programs with built-in's which uses this semantics. The method is shown to be modular and is illustrated by proving termination of a number of programs including the `unify` program of Sterling and Shapiro [17].

1991 Mathematics Subject Classification: 68Q40, 68T15,
1991 CR Categories: F.3.2., F.4.1, H.3.3, I.2.3.
Keywords and Phrases: Prolog programs, built-in's, declarative semantics, termination.

Notes. This research was partly done during the third author' s stay at Centre for Mathematics and Computer Science, Amsterdam. The work of K.R. Apt and E. Marchiori was partly supported by ESPRIT Basic Research Action 6810 (Compulog 2). The work of C. Palamidessi was partly supported by ESPRIT Basic Research Action 3020 (Integration) and by the Italian CNR (Consiglio Nazionale delle Ricerche). The work of E. Marchiori was also partly supported by the Italian CNR under Grant No. 89.00026.69.

1 Introduction

1.1 Motivation

Theory of logic programming allows us to treat formally only pure Prolog programs, that is those whose syntax is based on Horn clauses. Any formal treatment of more realistic Prolog programs has to take into account the use of various built-in's. Some of them, like arithmetic relations, seem to be trivial to handle, as they simply refer to some theory of arithmetic. However, the restrictions on the form of their arguments (like the requirement that both arguments of $<$ should be ground) cause complications which the theory of logic programming does not properly account for. In particular, in presence of arithmetic relations the independence of the refutability from the selection rule fails, as the goal $\leftarrow x = 2, 1 < x$ shows.

Further, the use of metalogical relations (like $var, ground$) leads to various additional problems. Clearly, var cannot be handled using the traditional semantics based on first-order logic because $var(x)$ is true whereas some instances of it are not. In presence of $nonvar$ another complication arises: the well-known Lifting Lemma (see Lloyd [14]) used to prove completeness of the SLD-resolution does not hold — for a non-variable term t the goal $\leftarrow nonvar(t)$ can be refuted whereas its more general version $\leftarrow nonvar(x)$ cannot.

Finally, study of termination of Prolog programs in presence of the above built-in's calls for some new insights. For example, the program `list`

```
list([]) ←.
list([X|Xs]) ←
  nonvar(Xs), list(Xs).
```

which recognizes a list, always terminates, whereas its pure Prolog counterpart obtained by dropping the atom $nonvar(Xs)$ may diverge.

The aim of this paper is to provide a systematic account of the class of the above mentioned built-in's of Prolog. This class includes the arithmetic relations (like $+, <$ etc.) and some metalogical relations (like $var, ground$ etc.). To distinguish them from those built-in's which refer to clauses and goals (like *call* and *assert*), we call them *first-order* built-in's. Hence the title.

The main tool in our approach is a new, non-standard declarative semantics which associates with each relation symbol *input* and *output* substitutions. It is introduced in Section 2. We also prove there a completeness result connecting this semantics with the Prolog computational mechanism. We show that this semantics is a *natural extension* of the S-semantics by Falaschi et al. [12], in the sense that it is isomorphic to the S-semantics for *pure* Prolog programs. Moreover we show that our semantics is in a sense *the most simple extension*, by proving that it is fully abstract w.r.t. goals conjunctions.

This semantics is crucial for the study of termination of Prolog programs that use the first-order built-in's. Our approach to this subject combines the use of the level mapping functions (that assign elements from a well-founded set to atoms) with the above semantics. In this respect it is thus similar to that of Apt and Pedreschi [5] which called for the use of level mappings assigning natural numbers to ground atoms, and declarative semantics. However, important differences arise due to the presence of built-in's. First, we have to analyze the original program and not its ground version. Second, in presence of first-order built-in's it seems natural to study programs that terminate for *all* goals and not only for all ground goals as in Apt and Pedreschi [5]. So different characterization results are needed. These issues are dealt with in Section 3 where we also show how termination of Prolog programs with first-order built-in's can be dealt with in a modular way.

In Section 4 we apply our approach to termination to prove termination of the above `list` program, the typed version of the `append` program and a version of the `unify` program of Sterling and Shapiro [17].

We are aware of two other approaches to define the meaning of Prolog *first-order* built-in's, namely that of Börger [7] based on so-called dynamic algebras, and that of Deransart and Ferrand [11] based on an abstract interpreter.

Their aim is to provide semantics to the complete Prolog language whereas ours is to extend the declarative semantics to Prolog programs with *first-order* built-in's so that one can reason about such programs. In this respect our approach has the same aim as that of Hill and Lloyd [13] where all metalogical features of Prolog are represented in a uniform way by means of a representation of the object level in the meta-level, reminiscent of the Gödelization process in Peano arithmetic.

1.2 Preliminaries

In what follows we study logic programs extended by various built-in relations. We call the resulting objects *Prolog programs*, or simply *programs*, and identify *pure Prolog programs* with logic programs. Prolog programs can be executed by means of the *LD-resolution*, which consists of the usual SLD-resolution combined with the leftmost selection rule, that is appropriately extended to deal with the built-in relations. By *length* $l(\xi)$ of an LD-derivation ξ we mean the number of its goals.

Given an expression (term, atom, goal,...) or a substitution E we denote the set of variables occurring in it by $Var(E)$. We often write $\eta \mid E$ to denote $\eta \mid Var(E)$. The set of all variables is denoted by Var. We often manipulate various sets of variables. In general $\mathbf{x}, \mathbf{y}$ stands for sequences of different variables. Sometimes we identify such sequences with sets of variables. Given a substitution η and a set of variables $\mathbf{x}$ we denote by $\eta \mid \mathbf{x}$ the substitution obtained from η by restricting its domain, $Dom(\eta)$, to $\mathbf{x}$. By $Ran(\eta)$ we denote the set of variables that appear in the terms of the range of η. A *renaming* is a substitution that is a permutation of the variables constituting its domain.

Recall that an mgu η of A and B is *idempotent* if $\eta\eta = \eta$ and is *relevant* if $Ran(\eta) \subseteq Var(A, B)$. The relation *more general than* defined on pairs of atoms, terms or substitutions is denoted by $\leq$.

Let s be a term. Then s_i denotes the i-th argument of s, when it is defined, $nodes(s)$ denotes the number of nodes of s in the tree representation, $a(s)$ denotes the arity of the principal functor of s and $funct(s)$ denotes its function symbol.

It is convenient to associate with each pair of terms that unify a unique idempotent (hence relevant) mgu in the sense of Apt [1][page 502]. Given such a pair s, t we denote it by $mgu(s, t)$. Further, we associate with each pair of sequences of terms that unify a unique idempotent (hence relevant) mgu defined as follows.

- $mgu((), ()) = \epsilon$, where $()$ indicates the empty sequence;
- $mgu((s, \mathbf{s}), (t, \mathbf{t})) = \alpha\ mgu((\mathbf{s}\alpha), (\mathbf{t}\alpha))$, where $\alpha = mgu(s, t)$.

We write $mgu(\mathbf{s}, \mathbf{t})$ instead of $mgu((\mathbf{s}), (\mathbf{t}))$. It is not difficult to show that $mgu(\mathbf{s}, \mathbf{t})$ is indeed an idempotent mgu of $\mathbf{s}$ and $\mathbf{t}$. Then we associate with each pair of atoms A and B that unify $mgu(\mathbf{s}, \mathbf{t})$, where $\mathbf{s}$ and $\mathbf{t}$ are the sequences of arguments respectively of A and B. We denote this mgu by $mgu(A, B)$.

Atoms of the form $p(\mathbf{x})$ where p is a relation are called *elementary atoms* and atoms containing a built-in relation are referred to as *built-in atoms.* Finally, atoms containing a relation used in a head of a clause of a program P are said to be *defined in* P. In the context of logic programs, or more generally, Prolog programs, it is convenient to treat sequences of atoms as conjunctions (sometimes called conjuncts). Usually $\mathbf{A}$, $\mathbf{B}$ denote such conjuncts.

The rest of the used notation is more or less standard and essentially follows Lloyd [14]. Recall that, if $\theta_1, \ldots, \theta_n$ are the consecutive mgu's along a refutation of a goal G in the program P, then the restriction $(\theta_1 \ldots \theta_n) \mid Var(G)$ of $\theta_1 \ldots \theta_n$ to the variables of G is called *computed answer substitution* (c.a.s. for short) of $P \cup \{G\}$. In this paper we also associate c.a.s.'s with prefixes of LD-derivations in the obvious way. These prefixes of LD-derivations are also called partial derivations.

2 The declarative semantics

2.1 Motivation

In this section we define a declarative semantics appropriate to describe the operational behaviour of Prolog programs. First, let us see why it is impossible to achieve this goal by simply modifying one of the usually considered declarative semantics.

The standard declarative semantics, based on the (ground) Herbrand models due to van Emden and Kowalski [18], is clearly inadequate to deal with first-order built-in's. Indeed, in this semantics in a given interpretation if an atom is true then all its ground instances are. However, for every ground term t, $var(t)$ should be false in every model whereas $var(x)$ should be true. Therefore we say that *var* is a *non-monotonic relation.*

We conclude that any declarative modeling of non-monotonic relations requires an explicit introduction of non-ground atoms in the Herbrand interpretations, in order to define the truth value of an atom independently from its ground instances. The first declarative semantics based on non-ground atoms was given by Clark [10], with the aim of defining the validity of open atoms (like $p(x)$) in terms of their truth value in the least Herbrand model. Successively, other declarative models based on non-ground atoms were investigated in Falaschi et al. [12]: the C-semantics - which was shown to be equivalent to Clark's semantics, and the S-semantics. However, all these models are not suitable for Prolog programs, because — like the standard semantics of van Emden and Kowalski [18], the resulting definition of truth treats the body of a clause as a logical conjunction - i.e. the ',' is interpreted as an 'and', and this means that the order of the literals in the body is irrelevant. On the other hand, the presence of built-in relations - in particular of the non-monotonic ones, makes this order relevant. Consider for instance

P_1:
```
p(X) ← var(X), q(X).
q(a) ←.
```

and

P_2:
```
p(X) ← q(X), var(X).
q(a) ←.
```

The behavior of the goal $\leftarrow p(x)$ in these programs is different (in P_1 it succeeds, whereas in P_2 it fails). In other words, the *independence from the selection rule*, and the Switching Lemma of Lloyd [14] do not hold for Prolog programs. If we want to characterize declaratively the operational behaviour of goals, we must therefore describe the meaning of ',' in the body of clauses in a non-commutative way, more precisely, we have to mimic the *leftmost selection rule* of Prolog.

However, the intended model cannot be obtained simply by modifying the interpretation of ',' in the C-semantics. The reason is that the domain structure of the C-semantics is too poor: it does not allow us to model the meaning of non-monotonic relations. Indeed, in the C-semantics the interpretations are upward closed, that is, if A belongs to (is true in) an interpretation I, then all its instances belong to I, as well.

On the other hand, in the S-semantics the interpretations are not upward closed. However, the S-semantics is monotonic, that is A is true in an interpretation I if a more general version of A belongs to I.

Moreover, in presence of built-in relations like *nonvar*, another problem arises: the goal $\leftarrow nonvar(x)$ fails whereas for every non-variable term t the goal $\leftarrow nonvar(t)$ succeeds. Therefore we say that *nonvar* is a *non-down-monotonic relation.* Due to the presence of non-down-monotonic relations the Lifting Lemma (see Lloyd [14]) does not hold for Prolog programs. Consider for instance

P_3: `p(X)` $\leftarrow$ `nonvar(X).`

With this program for every non-variable term t, the goal $\leftarrow p(t)$ has a refutation, whereas $\leftarrow p(x)$ fails.

This example shows that it is not sufficient to identify the meaning of a relation p with the set of (computed answer) substitutions η which p is able compute - in a sense, the post-conditions which are verified after the possible executions of the goal $\leftarrow p(\mathbf{x})$. We also need a pre-condition, i.e. information about the substitution θ by which the atom $p(\mathbf{x})$ is instantiated before starting the computation. A possible way to do it is by enriching the domain with another component, thus explicitly representing the substitution before execution.

2.2 Θ-semantics

This leads us to consider objects of the form $\langle \theta, p(\mathbf{x}), \eta \rangle$, where θ represents the *pre-substitution* (or *input substitution*) and η represents the *post-substitution* (or *output substitution*) for the goal $\leftarrow p(\mathbf{x})$. For technical convenience we equivalently represent these triples as pairs of the form $\langle A, \eta \rangle$, where A is the atom obtained by the application of the input substitution θ to the elementary atom $p(\mathbf{x})$, i.e. $A = p(\mathbf{x})\theta$. In Section 2.6 we prove the full abstraction of this model, thus showing that all the information we encode in this semantical structure is in fact necessary.

Of course, we can restrict our attention to pairs $\langle A, \eta \rangle$ in which η does not affect the variables that do not appear in A.

First, we deal with built-in relations. For any such relation p we stipulate a set $[\![p]\!]$ of pairs defining its operational behaviour. We list here some cases. In the definition below, "=" is the well-known built-in standing for "is unifiable with".

$$
\begin{array}{lcl}
[\![var]\!] & = & \{\langle var(x), \epsilon\rangle \mid x \in Var\}, \\[2ex]
[\![nonvar]\!] & = & \{\langle nonvar(s), \epsilon\rangle \mid s \notin Var\}, \\[2ex]
[\![=]\!] & = & \{\langle s = t, \eta\rangle \mid \eta = mgu(s,t)\}, \\[2ex]
[\![>]\!] & = & \{\langle s > t, \epsilon\rangle \mid s, t \text{ are integers and } s > t\}, \\[2ex]
[\![constant]\!] & = & \{\langle constant(a), \epsilon\rangle \mid a \text{ is a constant}\}, \\[2ex]
[\![compound]\!] & = & \{\langle compound(s), \epsilon\rangle \mid s \text{ is a compound term}\}, \\[2ex]
[\![functor]\!] & = & \{\langle functor(t,f,n), \eta\rangle \mid Dom(\eta) \subseteq \{f,n\},\ n\eta \text{ is a natural number and} \\
 & & t = (f\eta)(t_1, \ldots, t_{n\eta}) \text{ for some } t_1, \ldots, t_{n\eta},\ \text{ or } n \text{ is a natural number,} \\
 & & f \text{ is a functor symbol, } Dom(\eta) = \{t\} \text{ and } t\eta = f(X_1, \ldots, X_n) \\
 & & \text{where } X_1, \ldots, X_n \text{ are fresh variables}\}, \\[2ex]
[\![:=]\!] & = & \{\langle x := s, \{x/t\}\rangle \mid x \in Var,\ s \text{ is a ground arithmetic expression with value } t\}, \\[2ex]
[\![arg]\!] & = & \{\langle arg(n,s,t), \eta\rangle \mid Dom(\eta) \subseteq \{t\} \text{ and } t\eta = s_n \text{ or } Dom(\eta) = \{s_n\} \text{ and } s_n\eta = t\}, \\[2ex]
[\![\backslash ==]\!] & = & \{\langle s \backslash == t, \epsilon\rangle \mid s \neq t\}.
\end{array}
$$

We assume that the set of pairs associated with a built-in relation describes *correctly* its operational behaviour, in the following sense.

Definition 2.1 Let A be an atom with a built-in relation p. Then for every conjunct $\mathbf{B}$ the goal $\leftarrow \mathbf{B}\eta$ is a resolvent of $\leftarrow A, \mathbf{B}$ iff $\langle A, \eta\rangle \in [\![p]\!]$. □

Notice that in our approach we do not distinguish between failures and errors. For example, in Prolog the evaluation of the goal $\leftarrow X := Y + 1$ will result in an error and not in a (backtrackable) failure. By further refining the structure of the sets $[\![p]\!]$ we could easily incorporate this distinction in the semantics.

We consider now atoms defined by the program. First we introduce the following generalization of Herbrand base and Herbrand intepretation.

Definition 2.2 (Θ-domain and Θ-interpretation)

- Let P be a Prolog program. The Θ-base Θ_P of P is the set of all pairs $\langle A, \eta\rangle$, where A is an atom defined in P, and η is a substitution s.t. $Dom(\eta) \subseteq Var(A)$.
- A Θ-interpretation $\mathcal{I}$ of P is a subset of the Θ-base Θ_P. □

To define the truth in Θ-interpretations we have to model appropriately the proof theoretic properties of the computed answer substitutions. To this end it is important to reflect on them first.

The following lemma relates c.a.s.'s of resolvents of a goal with c.a.s.'s of the goal. It is a consequence of Corollary 3.5 of Apt and Doets [3], to which the reader is referred for the proof.

Lemma 2.3 (C.a.s.) *Consider an atomic goal* $\leftarrow A$ *with input clause* $p(\mathbf{x}) \leftarrow \mathbf{B}$. *Let* $\theta = mgu(A, p(\mathbf{x}))$ *be s.t.* $Dom(\theta) = \mathbf{x}$ *and let* η *be a c.a.s. of* $P \cup \{\leftarrow \mathbf{B}\theta\}$. *Suppose that* $Ran(\eta) \cap Var(p(\mathbf{x})\theta) \subseteq Var(\mathbf{B}\theta)$. *Then* $\eta | A$ *is a c.a.s. of* $P \cup \{\leftarrow A\}$.

This Lemma provides a sufficient condition to guarantee that a c.a.s. of a goal coincides with a c.a.s. of its resolvent on the variables of the goal. Let us give an example showing that this condition is needed. Consider the program P:

```
p(X,Y) ← q(X).
q(X) ← X=f(Y).
```

and the goal $\leftarrow p(X,Y)$. Take as input clause $p(X',Y') \leftarrow q(X')$. Then $\theta = mgu(p(X,Y), p(X',Y')) = \{X'/X, Y'/Y\}$ and $\leftarrow q(X)$ is the corresponding resolvent. Now $\eta = \{X/f(Y)\}$ is a c.a.s. of $P \cup \{\leftarrow q(X)\}$, but η is not a c.a.s. of $P \cup \{\leftarrow p(X,Y)\}$.

Definition 2.4 Let $\mathbf{A}$, $\mathbf{B}$ be conjuncts and let θ and σ substitutions. We say that $(\mathbf{A}, \mathbf{B}, \theta, \sigma)$ is a *good tuple* if the following conditions are satisfied:

(i) $Ran(\theta) \cap Var(\mathbf{B}) \subseteq Var(\mathbf{A})$
(the variables introduced by θ that occur in $\mathbf{B}$ also occur in $\mathbf{A}$),

(ii) $Ran(\sigma) \cap (Var(\mathbf{A}, \mathbf{B}) \cup Ran(\theta)) \subseteq Var(\mathbf{B}\theta)$
(the variables introduced by σ that occur in $\mathbf{A}, \mathbf{B}$ or in $Ran(\theta)$ also occur in $\mathbf{B}\theta$). □

The importance of this, admittedly esoteric, notion is revealed by the following lemma, which characterizes c.a.s.'s of a conjunction of goals in terms of c.a.s.'s of its conjuncts.

Lemma 2.5 (Good Tuple) *Consider a goal* $\leftarrow \mathbf{A}, \mathbf{B}$. *Then* η *is a c.a.s. of* $P \cup \{\leftarrow \mathbf{A}, \mathbf{B}\}$ *iff for some* θ *and* σ

- θ *is a c.a.s. of* $P \cup \{\leftarrow \mathbf{A}\}$,
- σ *is a c.a.s. of* $P \cup \{\leftarrow \mathbf{B}\theta\}$,
- $\eta = (\theta\sigma) | (\mathbf{A}, \mathbf{B})$,
- $(\mathbf{A}, \mathbf{B}, \theta, \sigma)$ *is a good tuple.*

Proof. The proof is lengthy and tedious and can be found in the technical report [4].
□

This lemma shows that the c.a.s.'s for a compound goal $\leftarrow \mathbf{A}, \mathbf{B}$ cannot be obtained by simply composing each c.a.s. θ for $\leftarrow \mathbf{A}$ with each c.a.s. σ for $\leftarrow \mathbf{B}\theta$. The notion of a good tuple formalizes the conditions that θ and σ have to satisfy, due to the standardization apart. Both conditions of Definition 2.4 of Good Tuple are needed: consider for example the program P: `p(Z) ←.` and the goal $G = \leftarrow p(X), p(Y)$. Then $\theta = \{X/Y\}$ is a c.a.s. for $\leftarrow p(X)$, $\sigma = \epsilon$ is a c.a.s. of $P \cup \{\leftarrow p(Y)\theta\}$ but $(\theta\sigma) | G = \{X/Y\}$ is not a c.a.s. of $P \cup \{G\}$. This shows that the first condition in Definition 2.4 of good tuple is needed. Now $\theta = \epsilon$ is also a c.a.s. for $\leftarrow p(X)$, $\sigma = \{Y/X\}$ is a c.a.s. of $P \cup \{\leftarrow p(Y)\theta\}$ but $(\theta\sigma) | G = \{Y/X\}$ is not a c.a.s. of $P \cup \{G\}$. This shows that the second condition in Definition 2.4 of good tuple is needed.

Since we want to model the meaning of a conjunct w.r.t. a post-substitution η in such a way that a precise match with the procedural semantics is maintained, the notion of a good tuple will be crucial also for the semantic considerations.

The next step is dictated by the simplicity considerations. We shall restrict our attention to Prolog programs in a certain form. Then, after proving soundness and completeness for these programs, we shall return to the general case.

Definition 2.6 (Homogeneous Programs)

- A Prolog clause is called *homogeneous* if its head is an elementary atom.
- A Prolog program is called *homogeneous* if all its clauses are homogeneous. □

We now define truth in Θ-interpretations for homogeneous programs. It relies on the notion of good tuple. Given a conjunct $\mathbf{A}$ of atoms we denote by $l(\mathbf{A})$ its length, i.e. the number of atoms in $\mathbf{A}$. If $l(\mathbf{A}) = 0$ we denote $\mathbf{A}$ by *true*.

Definition 2.7 (Truth in Θ-interpretations) Let $\mathcal{I}$ be a Θ-interpretation of a homogeneous Prolog program P.

The *truth* of a conjunct $\mathbf{A}$ in $\mathcal{I}$ w.r.t. a (post-)substitution η, denoted by $\mathcal{I} \models \langle \mathbf{A}, \eta \rangle$, is defined by induction on $l(\mathbf{A})$, the length of $\mathbf{A}$.

- $l(\mathbf{A}) = 0$. Then $\mathbf{A} = true$.

  $\mathcal{I} \models \langle true, \eta \rangle$ iff $\eta = \epsilon$.

- $l(\mathbf{A}) = 1$. Then $\mathbf{A} = A$ for an atom A.

  $\mathcal{I} \models \langle A, \eta \rangle$ iff $\langle A, \eta \rangle \in [\![p]\!]$, where A is a built-in atom with the relation symbol p,

  $\mathcal{I} \models \langle A, \eta \rangle$ iff $\langle A, \eta \rangle \in \mathcal{I}$, where A is defined in P.

- $l(\mathbf{A}) > 1$. Then $\mathbf{A} = A, \mathbf{B}$ for an atom A and a non-empty conjunct $\mathbf{B}$.

  $\mathcal{I} \models \langle A, \mathbf{B}, \eta \rangle$ iff there exist θ, σ s.t. $\eta = (\theta\sigma) \,|\, (A, \mathbf{B})$ and

  - $\mathcal{I} \models \langle A, \theta \rangle$,
  - $\mathcal{I} \models \langle \mathbf{B}\theta, \sigma \rangle$
  - $(A, \mathbf{B}, \theta, \sigma)$ is a good tuple.

The *truth* of a homogeneous clause $H \leftarrow \mathbf{B}$ of P in $\mathcal{I}$, denoted by $\mathcal{I} \models H \leftarrow \mathbf{B}$, is defined as follows.

- $\mathcal{I} \models \langle H \leftarrow \mathbf{B}, \eta \rangle$ iff for all θ s.t. $Dom(\theta) = Var(H)$, $Ran(\theta) \cap Var(H \leftarrow \mathbf{B}) = \emptyset$, $Ran(\eta) \cap Var(H\theta) \subseteq Var(\mathbf{B}\theta)$:

  $\mathcal{I} \models \langle \mathbf{B}\theta, \eta \rangle$ implies $\mathcal{I} \models \langle H\theta, \eta \,|\, H\theta \rangle$,

- $\mathcal{I} \models H \leftarrow \mathbf{B}$ iff for all η, $\mathcal{I} \models \langle H \leftarrow \mathbf{B}, \eta \rangle$.

$\mathcal{I}$ is a Θ-model of P iff all variants of the clauses of P are true in $\mathcal{I}$. □

Notice that in the definition of the truth of a clause the restrictions on θ and σ are needed in order to establish the correspondence with the operational semantics. These restrictions model at the declarative level the restrictions induced by the standardization apart. The following lemmas will be useful to reason about the truth.

Lemma 2.8 (Monotonicity) *Let $\mathcal{I}, J$ be Θ-interpretations,* $\mathbf{A}$ *a conjunct, and* η *a substitution. If* $\mathcal{I} \models \langle \mathbf{A}, \eta \rangle$ *and* $\mathcal{I} \subseteq J$, *then* $J \models \langle \mathbf{A}, \eta \rangle$.

Proof. Straightforward by induction on the length of $\mathbf{A}$. □

Lemma 2.9 (Continuity) *Let $\mathcal{I}_i$ $(i \geq 0)$ be Θ-interpretations such that $\mathcal{I}_0 \subseteq \mathcal{I}_1 \subseteq \ldots$. Then for every conjunct* $\mathbf{A}$ *and substitution* η

$$\cup_{i=0}^{\infty} \mathcal{I}_i \models \langle \mathbf{A}, \eta \rangle \textit{ iff for some } k \geq 0\ \mathcal{I}_k \models \langle \mathbf{A}, \eta \rangle.$$

Proof. Straightforward by induction on the length of $\mathbf{A}$ and the Monotonicity Lemma 2.8. □

Note that the Continuity Lemma strengthens the Monotonicity Lemma.

2.3 Θ-semantics and LD-resolution

The next step is to show that LD-resolution is correct w.r.t. the Θ-semantics. The proof relies on the Good Tuple Lemma 2.5.

The following assumption is convenient.

Whenever in the LD-resolution step the selected atom A is unified with the head H of the input clause where H is a pure atom, then the mgu θ of A and H is s.t. $Dom(\theta) = Var(H)$.

By the previous assumption we have $A = H\theta$.

Theorem 2.10 (Soundness I) *Let P be a homogeneous Prolog program and* $\mathbf{A}$ *a conjunct. If η is a c.a.s. for $P \cup \{\leftarrow \mathbf{A}\}$ then for any Θ-model $\mathcal{I}$ of P we have $\mathcal{I} \models \langle \mathbf{A}, \eta \rangle$.*

Proof. Fix a Θ-model $\mathcal{I}$ of P. Let ξ be a LD-refutation of $P \cup \{\leftarrow \mathbf{A}\}$ with c.a.s. η. We prove the claim by induction on the length $l(\xi)$ of ξ. Three cases arise.

Case 1 $l(\mathbf{A}) = 0$. Then $\mathbf{A} = \mathit{true}$ and $\eta = \epsilon$, so the claim follows directly by Definition 2.7.

Case 2 $l(\mathbf{A}) = 1$. Then $\mathbf{A} = A$ for an atom A.
If A is a built-in atom, then the claim follows directly by Definitions 2.1 and 2.7.
If A is defined in P, then consider the resolvent $\mathbf{B}\theta$ of $\leftarrow A$ in ξ obtained using the input clause $H \leftarrow \mathbf{B}$ and mgu θ. H is a pure atom and by the standardization apart A and $H \leftarrow \mathbf{B}$ have no variable in common, so by Assumption 2.3

$$Dom(\theta) = Var(H),\ Ran(\theta) \cap Var(H \leftarrow \mathbf{B}) = \emptyset, \tag{1}$$

and

$$A = H\theta. \tag{2}$$

Let η' be the c.a.s. for $P \cup \{\leftarrow \mathbf{B}\theta\}$ computed by the suffix ξ' of ξ starting at $\leftarrow \mathbf{B}\theta$. Then

$$\eta = (\theta\eta') \mid A. \tag{3}$$

We have $l(\xi') = l(\xi) - 1$, so by the induction hypothesis $\mathcal{I} \models \langle \mathbf{B}\theta, \eta' \rangle$. But $\mathcal{I}$ is a model of P, so $H \leftarrow \mathbf{B}$ is true in $\mathcal{I}$ and consequently by (1) and Definition 2.7 $\mathcal{I} \models \langle H\theta, \eta' \mid H\theta \rangle$. Thus by (2) $\mathcal{I} \models \langle A, \eta' \mid A \rangle$. However, A and H have no variable in common, so by (1) $\theta \mid A = \epsilon$ and consequently by (3) $\eta = (\theta\eta') \mid A = \eta' \mid A$. So we proved $\mathcal{I} \models \langle A, \eta \rangle$.

Case 3 $l(\mathbf{A}) > 1$. Then $\mathbf{A} = A, \mathbf{B}$ for an atom A and a non-empty conjunct $\mathbf{B}$. By the Good Tuple Lemma 2.5 there exist θ and σ s.t. $\eta = (\theta\sigma) \mid \mathbf{A}$ and

(i) $P \cup \{\leftarrow A\}$ has an LD-refutation ξ_1 with c.a.s. θ,

(ii) $P \cup \{\leftarrow \mathbf{B}\theta\}$ has an LD-refutation ξ_2 with c.a.s. σ,

(iii) $(A, \mathbf{B}, \theta, \sigma)$ is a good tuple.

Moreover by the proof of the same Lemma it follows that we can choose ξ_1, ξ_2 to be subderivations of ξ. Then $l(\xi_1) < l(\xi)$ so by the induction hypothesis

$$\mathcal{I} \models \langle A, \theta \rangle. \tag{4}$$

Also $l(\xi_2) < l(\xi)$ so by the induction hypothesis

$$\mathcal{I} \models \langle \mathbf{B}\theta, \sigma \rangle. \tag{5}$$

Thus by (iii), (4) and (5) we get $\mathcal{I} \models \langle \mathbf{A}, \eta \rangle$ by Definition 2.7.

□

In order to prove the converse of Theorem 2.10 it is helpful to consider a special Θ-model representing all Θ-models, in the sense that a conjunction is true in it (w.r.t. a given post-substitution) iff it is true in all the Θ-models.

The Θ-interpretations are naturally ordered by the set inclusion. In this ordering the least Θ-interpretation is $\emptyset$, the greatest one is Θ_P. Analogously to standard Herbrand models, the Θ-models are closed w.r.t. arbitrary intersections, from which we deduce the existence of the least Θ-model.

Theorem 2.11 *Let P be a homogeneous program. Let $\mathcal{M}$ be a class of Θ-models of P. Then $M = \bigcap_{\mathcal{M}} \mathcal{I}$ is a model of P.*

Proof. Let $H \leftarrow \mathbf{B}$ be a variant of a clause of P and let η, θ be such that $Dom(\theta) = Var(H)$, $Ran(\theta) \cap Var(H \leftarrow \mathbf{B}) = \emptyset$, $Ran(\eta) \cap Var(H\theta) \subseteq Var(\mathbf{B}\theta)$ and $M \models \langle \mathbf{B}\theta, \eta \rangle$. Fix $\mathcal{I} \in \mathcal{M}$. By the Monotonicity Lemma 2.8 we have $\mathcal{I} \models \langle \mathbf{B}\theta, \eta \rangle$, so since $\mathcal{I}$ is a Θ-model, $\mathcal{I} \models \langle H\theta, \eta \,|\, (H\theta) \rangle$. By Definition 2.7 and the fact that $\mathcal{I}$ is an arbitrary element of $\mathcal{M}$ we conclude $M \models \langle H\theta, \eta \,|\, H\theta \rangle$.
□

Corollary 2.12 (Least Model) *Every homogeneous program P has a least Θ-model, N_P.*

□

This Θ-model is the intended representant of all Θ-models of P in the following sense.

Corollary 2.13 *Let $\mathbf{A}$ be a conjunct and η be a substitution. Then $N_P \models \langle \mathbf{A}, \eta \rangle$ iff for all Θ-models $\mathcal{I}$ of P we have $\mathcal{I} \models \langle \mathbf{A}, \eta \rangle$.*

Proof. By the Monotonicity Lemma 2.8. □

In the theory of Logic Programming the least Herbrand model can be generated as the least fixpoint of the immediate consequence operator T_P on the Herbrand interpretations. This characterization is useful to establish the completeness of SLD-resolution with respect to the least Herbrand model. We now provide an analogous characterization of the least Θ-model N_P in order to show the completeness of the LD-resolution with respect to N_P.

First, we introduce the appropriate operator T_P.

Definition 2.14 Let P be a homogeneous program. The immediate consequence operator T_P on the Θ-interpretations is defined as follows

$$\begin{array}{rcll} T_P(\mathcal{I}) & = & \{\langle H\theta, \eta \,|\, (H\theta)\rangle \mid & \text{for some } \mathbf{B} \\ & & & H \leftarrow \mathbf{B} \ \text{ is a variant of a clause from } P, \\ & & & Dom(\theta) = Var(H),\ Ran(\theta) \cap Var(H \leftarrow \mathbf{B}) = \emptyset, \\ & & & Ran(\eta) \cap Var(H\theta) \subseteq Var(\mathbf{B}\theta),\ \mathcal{I} \models \langle \mathbf{B}\theta, \eta\rangle\}. \end{array}$$

□

Next, we characterize the Θ-models of P as the pre-fixpoints of T_P. The following proposition shows this characterization for programs consisting of one clause only.

Proposition 2.15 *Given a clause C and a Θ-interpretation $\mathcal{I}$, we have that $\mathcal{I}$ is a model of $\{C\}$ iff $T_{\{C\}}(\mathcal{I}) \subseteq \mathcal{I}$.*

Proof. For every H, θ and η we have $\langle H\theta, \eta \mid H\theta\rangle \in T_{\{C\}}(\mathcal{I})$ iff (by Definition 2.14) $H \leftarrow \mathbf{B}$ is a variant of C such that $\mathcal{I} \models \langle \mathbf{B}\theta, \eta\rangle$, $Dom(\theta) = Var(H)$, $Ran(\theta) \cap Var(H \leftarrow \mathbf{B}) = \emptyset$ and $Ran(\eta) \cap Var(H\theta) \subseteq Var(\mathbf{B}\theta)$. Since $\mathcal{I}$ is a model of $\{C\}$ then this holds iff $\mathcal{I} \models \langle H\theta, \eta \,|\, (H\theta)\rangle$, i.e. $\langle H\theta, \eta \,|\, (H\theta)\rangle \in \mathcal{I}$. □

To generalize Proposition 2.15 to non-singleton programs we use the following obvious lemma which states the additivity of the operator T_P.

Lemma 2.16 *Let P, P' be homogeneous programs. Then for every Θ-interpretation $\mathcal{I}$ we have $T_{P \cup P'}(\mathcal{I}) = T_P(\mathcal{I}) \ \cup\ T_{P'}(\mathcal{I})$.* □

Corollary 2.17 (Model Characterization) *$\mathcal{I}$ is a Θ-model of P iff $T_P(\mathcal{I}) \subseteq \mathcal{I}$.* □

Now, we characterize N_P as the least fixpoint of T_P. We need the following observation.

Proposition 2.18 (Monotonicity) *T_P is monotonic, that is $I \subseteq J$ implies $T_P(I) \subseteq T_P(J)$.*

Proof. By the Monotonicity Lemma 2.8. □

Proposition 2.19 (Least Fixpoint) *T_P has a least fixpoint $lfp(T_P)$ which is also its least pre-fixpoint.*

Proof. By the Monotonicity Proposition 2.18 and Knaster-Tarski Theorem. □

We can now derive the desired result.

Corollary 2.20 *$lfp(T_P) = N_P$.*

Proof. By the Least Fixpoint Proposition 2.19, Least Model Corollary 2.12 and Model Characterization Corollary 2.17. □

Finally, we provide a more precise characterization of the Θ-model N_P that will be used in the proof of the completeness of the LD-resolution. We need the following strengthening of the Monotonicity Proposition 2.18.

Proposition 2.21 (Continuity) *T_P is continuous, that is for every sequence $\mathcal{I}_i$ $(i \geq 0)$ of Θ-interpretations such that $\mathcal{I}_0 \subseteq \mathcal{I}_1 \subseteq \ldots$ we have*

$$T_P(\cup_{i=0}^{\infty}\mathcal{I}_i) = \cup_{i=0}^{\infty}T_P(\mathcal{I}_i).$$

Proof. By the Continuity Lemma 2.9. □

We define now a sequence of Θ-interpretations by

$$T_P \uparrow 0 = \emptyset,$$

$$T_P \uparrow (n+1) = T_P(T_P \uparrow n),$$

$$T_P \uparrow \omega = \cup_{i=0}^{\infty} T_P \uparrow i.$$

Proposition 2.22 (Characterization) $N_P = T_P \uparrow \omega$.

Proof. By the Continuity Proposition 2.21 and the Knaster-Tarski Theorem $\mathit{lfp}(T_P) = T_P \uparrow \omega$, so the claim follows by Corollary 2.20. □

We can now prove the completeness of LD-resolution with respect to the Θ-semantics for homogeneous programs.

Theorem 2.23 (Completeness I) *Consider a homogeneous program P and a conjunct $\mathbf{A}$. Suppose that for all Θ-models $\mathcal{I}$ of P we have $\mathcal{I} \models \langle \mathbf{A}, \eta \rangle$. Then there exists an LD-refutation of $P \cup \{\leftarrow \mathbf{A}\}$ with c.a.s. η.*

Proof. In particular we have $N_P \models \langle \mathbf{A}, \eta \rangle$. By the Characterization Proposition 2.22 $T_P \uparrow \omega \models \langle \mathbf{A}, \eta \rangle$. By the monotonicity of T_P we have $T_P \uparrow 0 \subseteq T_P \uparrow 1 \subseteq \ldots$, so by the Continuity Lemma 2.9 $T_P \uparrow k \models \langle \mathbf{A}, \eta \rangle$ for some $k > 0$.
We now prove the claim by induction w.r.t. the lexicographic ordering $<$ defined on pairs $\langle k, l(\mathbf{A}) \rangle$ of natural numbers. In this ordering

$$\langle n_1, n_2 \rangle < \langle m_1, m_2 \rangle \text{ iff } (n_1 < m_1) \text{ or } (n_1 = m_1 \wedge n_2 < m_2).$$

The case when $\mathbf{A}$ is empty, i.e. $l(\mathbf{A}) = 0$ (which covers the base case of the induction) is immediate by Definition 2.7.
Suppose now $\mathbf{A} = A, \mathbf{B}$. There exist substitutions θ, σ such that

$$T_P \uparrow k \models \langle A, \theta \rangle,$$

$$T_P \uparrow k \models \langle \mathbf{B}\theta, \sigma \rangle,$$

$(A, \mathbf{B}, \theta, \sigma)$ is a good tuple and $\eta = (\theta\sigma) \mid (A, \mathbf{B})$.
We first prove that $P \cup \{\leftarrow A\}$ has an LD-refutation with c.a.s. θ. When A is a built-in atom this conclusion follows immediately from Definitions 2.1 and 2.7.
When A is defined in P we have $k > 0$. By Definition 2.14 there exists a variant $H \leftarrow \mathbf{B}'$ of a clause from P, a substitution ψ s.t. $\mathit{Dom}(\psi) = \mathit{Var}(H)$, $\mathit{Ran}(\psi) \cap \mathit{Var}(H \leftarrow \mathbf{B}') = \emptyset$, $A = H\psi$ and a substitution ϕ such that

$$\mathit{Ran}(\phi) \cap \mathit{Var}(H\psi) \subseteq \mathit{Var}(\mathbf{B}\psi), \tag{6}$$

$T_P\uparrow(k-1) \models \langle \mathbf{B}'\psi, \phi\rangle$ and $\theta = \phi \,|\, A$.

Since $\langle k-1, l(\mathbf{B}'\psi)\rangle < \langle k, l(\mathbf{A})\rangle$, by the induction hypothesis there exists an LD-refutation of $P \cup \{\leftarrow \mathbf{B}'\psi\}$ with c.a.s. ϕ. Now notice that $Dom(\psi) = Var(H)$, $\leftarrow \mathbf{B}'\psi$ is a resolvent of $\leftarrow A$ using the mgu ψ and (6) holds. Then by the c.a.s. Lemma 2.3 θ is a c.a.s. of $P \cup \{\leftarrow A\}$.

Since $\langle k, l(\mathbf{B}\theta)\rangle < \langle k, l(\mathbf{A})\rangle$, by the induction hypothesis also there exists an LD-refutation of $P \cup \{\leftarrow \mathbf{B}\theta\}$ with c.a.s. σ. Since $(A, \mathbf{B}, \theta, \sigma)$ is a good tuple and $\eta = (\theta\sigma) \,|\, (A, \mathbf{B})$, we can apply the Good Tuple Lemma 2.5. We conclude that there exists an LD refutation of $P \cup \{\leftarrow \mathbf{A}\}$ with c.a.s. η. □

Corollary 2.24 *Let P be a homogeneous Prolog program. Then*

$$N_P = \{\langle A, \eta\rangle \mid A \text{ is defined in } P \text{ and there exists an LD-refutation of } P \cup \{\leftarrow A\} \text{ with c.a.s. } \eta\}.$$

Proof. By Definition 2.7 and Theorems 2.10 and 2.23. □

This corollary shows that the Θ-model N_P captures precisely the computational meaning of the homogeneous program P.

2.4 Extension to arbitrary programs

Now, every program can be easily transformed into a homogeneous program.

Definition 2.25 (Homogeneous Form) Let P be a Prolog program. Let $x_1, x_2, \ldots$ be distinct variables not occurring in P. Transform each clause

$$p(t_1, \ldots, t_k) \leftarrow \mathbf{B}$$

of P into the clause

$$p(x_1, \ldots, x_k) \leftarrow x_1 = t_1, \ldots, x_k = t_k, \mathbf{B}.$$

Here $=$ is the built-in discussed in Section 2 and interpreted as "is unifiable with". We denote the resulting program by $Hom(P)$ and call it a *homogeneous form* of P. □

We now show that a Prolog program P and its homogeneous form $Hom(P)$ have the same computational behaviour.

Theorem 2.26 (Equivalence I) *Let P be a Prolog program, G a goal. Then $P \cup \{G\}$ has a refutation with c.a.s. η if and only if $Hom(P) \cup \{G\}$ has a refutation with c.a.s. η.*

Proof. See [4]. □

Theorem 2.26 allows to reason about the meaning of Prolog programs by transforming them first to a homogeneous form. Alternatively, we can extend the definition of the truth to arbitrary programs by simply defining a clause to be true iff its homogeneous version is true. By "processing" then the meaning of the introduced calls to the built-in $=$ we obtain the following direct definition of truth of a clause.

Definition 2.27 $\mathcal{I} \models \langle H \leftarrow \mathbf{B}, \eta\rangle$ iff for any atom A and a variant $H' \leftarrow \mathbf{B}'$ of $H \leftarrow \mathbf{B}$ disjoint from A the following implication holds: $\theta = mgu(A, H')$, $\mathcal{I} \models \langle \mathbf{B}'\theta, \eta\rangle$ and $Ran(\eta) \cap (Var(A) \cup Var(H' \leftarrow \mathbf{B}')) \subseteq Var(\mathbf{B}'\theta)$ implies $\mathcal{I} \models \langle A, \theta\eta \,|\, A\rangle$. □

We now establish the semantics equivalence of a program and its homogeneous form.

Theorem 2.28 (Equivalence II) *$\mathcal{I}$ is a model of a Prolog program P iff it is a model of $Hom(P)$.*

Proof. See [4]. $\square$

¿From the two previous results on operational and semantic equivalence of P and $Hom(P)$ the soundness and completeness of the LD-resolution for Prolog programs directly follows.

Theorem 2.29 (Soundness II) *Let P be a Prolog program and $\mathbf{A}$ a conjunct. If η is a c.a.s. for $P \cup \{\leftarrow \mathbf{A}\}$ then for any Θ-model $\mathcal{I}$ of P we have $\mathcal{I} \models \langle \mathbf{A}, \eta \rangle$.*

Proof. By the Equivalence I Theorem 2.26 and the Equivalence II Theorem 2.28. $\square$

Theorem 2.30 (Completeness II) *Consider a Prolog program P and a conjunct $\mathbf{A}$. Suppose that for all Θ-models $\mathcal{I}$ of P we have $\mathcal{I} \models \langle \mathbf{A}, \eta \rangle$. Then there exists an LD-refutation of $P \cup \{\leftarrow \mathbf{A}\}$ with c.a.s. η.*

Proof. By the Equivalence II Theorem 2.28 and the Equivalence I Theorem 2.26. $\square$

2.5 Relation between the Θ-semantics and the S-semantics

In this section we show that the Θ-semantics is the natural extension to Prolog programs of the S-semantics defined in Falaschi et al. [12] for logic programs, in the sense that if P is a *pure* Prolog program (i.e. it does not contain built-in atoms) then the least Θ-model N_P coincides with the least S-model S_P. To this purpose, it will be helpful to consider the following operational characterization of S_P (cf. Falaschi et al. [12]).

$$S_P = \{p(\mathbf{x})\eta \mid \mathbf{x} \in Var \text{ and } \leftarrow p(\mathbf{x}) \text{ has an LD-refutation with c.a.s. } \eta\}$$

or, equivalently,

$$S_P = \{\langle p(\mathbf{x}), \eta \rangle \mid \mathbf{x} \in Var \text{ and } \leftarrow p(\mathbf{x}) \text{ has an LD-refutation with c.a.s. } \eta\} \tag{7}$$

We define now some properties on Θ-interpretations which will be shown to hold for N_P when P is pure, and which will be useful for proving the correspondence stated above.

Definition 2.31 Let $\mathcal{I}$ be a Θ-interpretation. $\mathcal{I}$ is called

- upward-closed iff $\forall \langle A, \eta \rangle \in \mathcal{I}$, $\forall \theta$ such that $\exists \sigma = mgu(A\theta, A\eta)$, we have $\langle A\theta, \sigma' \rangle \in \mathcal{I}$, where σ' is the restriction of σ to $A\theta$.
- downward-closed iff $\forall \langle A\theta, \sigma \rangle \in \mathcal{I}$. $\exists \eta$. $\exists \sigma' = mgu(A\theta, A\eta)$. $\langle A, \eta \rangle \in \mathcal{I}$ and σ is the restriction of σ' to $A\theta$. $\square$

Proposition 2.32 *Let P be a pure Prolog program. Then N_P is upward-closed and downward-closed.*

Proof. By using the characterization of N_P expressed by Corollary 2.24 it is sufficient to prove the operational counterparts of upward and downward closedness, which when extended to arbitrary conjunctions, are expressed by the following lemma.

Lemma 2.33

1. *If the goal* $\leftarrow \mathbf{Q}$ *has a LD-refutation with c.a.s.* η*, then, for each* θ *such that* $\exists\sigma = mgu(\mathbf{Q}\theta, \mathbf{Q}\eta)$*, the goal* $\leftarrow \mathbf{Q}\theta$ *has an LD-refutation with a computed answer substitution* σ' *which is the restriction of* σ *to* $\mathbf{Q}\theta$*.*

2. *If the goal* $\leftarrow \mathbf{Q}\theta$ *has a LD-refutation with c.a.s.* σ *then there exists* η *and* $\sigma' = mgu(\mathbf{Q}\theta, \mathbf{Q}\eta)$ *such that* $\leftarrow \mathbf{Q}$ *has a LD-refutation with c.a.s.* η *and* σ *is the restriction of* σ' *to* $\mathbf{Q}\eta$*.*

Proof. See [4]. $\square$

Note the analogy between Lemma 2.33(2) and the Lifting Lemma. Actually, Lemma 2.33(2) (which can obviously be generalized to arbitrary selection rules) is stronger than Lifting Lemma, because not only it ensures the existence of η, but it also gives more precise information about the relation between θ, σ and η (from the Lifting Lemma we would only know that $\mathbf{Q}\eta \leq \mathbf{Q}\theta\sigma$).

If P contains built-in relations, then N_P could be non upward-closed or non downward-closed.

Example 2.34 Consider the program P:

```
p(X) ← var(X), q(X).
q(a) ← .
```

The goal $\leftarrow p(x)$ has an LD-refutation with c.a.s. $\eta = \{x/a\}$, but the goal $\leftarrow p(x)\eta$ has no refutations. Thus, N_P is not upward-closed.

Consider the program P:

```
p(X) ← nonvar(X).
```

The goal $\leftarrow p(a)$ has an LD-refutation, but the goal $\leftarrow p(x)$ has no refutations. Thus, N_P is not downward-closed.

We show now that if P is a pure Prolog program, then N_P is isomorphic to the least S-model S_P, in the sense that there exist a mapping α from S-interpretations to Θ-interpretations, and a mapping β from Θ-interpretations to S-interpretations such that for every program P $N_P = \alpha(S_P)$ and $S_P = \beta(N_P)$.

Note that α and β are *abstraction operators*, i.e. they do not depend upon P: if $S_{P_1} = S_{P_2}$ then $\beta(S_{P_1}) = \beta(S_{P_2})$ and if $N_{P_1} = N_{P_2}$ then $\alpha(N_{P_1}) = \alpha(N_{P_2})$.

Definition 2.35 The mappings *Up* from S-interpretations to Θ-interpretations, and *Kernel* from Θ-interpretations to S-interpretations are defined as follows.

$$Up(\mathcal{I}) = \{\langle A\theta, \sigma\rangle \mid \exists\eta.\ \langle A, \eta\rangle \in \mathcal{I} \text{ and } \exists\sigma' = mgu(A\theta, A\eta).\ \sigma \text{ is the restriction of } \sigma' \text{ to } A\theta\},$$
$$Kernel(\mathcal{I}) = \{\langle p(\mathbf{x}), \eta\rangle \mid \mathbf{x} \in Var \text{ and } \langle p(\mathbf{x}), \eta\rangle \in \mathcal{I}\}$$

$\square$

Note that the definition of *Up* and *Kernel* does not depend upon P. We prove that *Up* and *Kernel* are the intended α and β satisfying the property described above.

Proposition 2.36 *If* P *is a pure Prolog program, then*

1. $N_P = Up(S_P)$*, and*

2. $S_P = Kernel(N_P)$.

Proof. The equality $S_P = Kernel(N_P)$ follows immediately by the definition of *Kernel* and by (7). Therefore we have only to prove that $N_P = Up(Kernel(N_P))$.

$\subseteq$) Let $\langle A, \sigma\rangle \in N_P$. Assume $A = p(\mathbf{x})\theta$. Then by Proposition 2.32(2) there exist η and $\sigma' = mgu(A, p(\mathbf{x})\eta)$ such that $\langle p(\mathbf{x}), \eta\rangle \in N_P$ (and therefore $\langle p(\mathbf{x}), \eta\rangle \in Kernel(N_P)$), and σ is the restriction of σ' to A. The rest follows by Proposition 2.32(1).

$\supseteq$) Let $\langle A, \sigma\rangle \in Up(Kernel(N_P))$. Then there exists $\langle p(\mathbf{x}), \eta\rangle \in Kernel(N_P) \subseteq N_P$ such that, for some θ, $A = p(\mathbf{x})\theta$ and $\exists\sigma' = mgu(A, p(\mathbf{x})\eta)$ such that σ is the restriction of σ' to A. By Proposition 2.32(1), we conclude $\langle A, \sigma\rangle \in N_P$. $\square$

2.6 Full abstraction of the Θ-semantics

In the previous sections we have seen that N_P coincides with the set of computational pairs $\langle A, \eta\rangle$ s.t. there exists an LD-refutation of $P \cup \{\leftarrow A\}$ with c.a.s. η and that the Θ-semantics is and-compositional, in the sense that the truth value of a conjunction of atoms (possibly sharing variables) can be derived by the truth value of the atoms.

We argue that a declarative semantics should provide such a compositional interpretation of conjuncts. We focus on conjuncts of the form

$$p_1(\mathbf{x}_1), \ldots, p_n(\mathbf{x}_n)$$

where the $p_i(\mathbf{x}_i)$'s are either elementary atoms or atoms of the form $x = t$, and $\mathbf{x}_1, \ldots, \mathbf{x}_n$ are possibly not disjoint. Every conjunct can be equivalently transformed into a conjunct of this form.

One might wonder whether it is possible to develop a declarative semantics for Prolog based on a simpler (i.e. more abstract) domain than the Θ-domain, possibly encoding less information concerning the computational behavior of goals. One might for instance be interested in *observing* only the *non-ground success set* of a program P, defined as:

$$NGSS_P = \{A\eta \mid \; \leftarrow A \text{ has an LD-refutation with c.a.s. } \eta\}$$

(which corresponds to the least C-model when P is a pure program (cf. Falaschi et al. [12])). This notion can be considered the most abstract interesting one, since, as we already have seen in the introduction, the *ground* success set is not suitable for programs containing built-in relations. So the question is:

> is it possible to give a declarative, hence and-compositional, characterization of $NGSS_P$?

If we want to have a declarative model which *coincides* with $NGSS_P$, then the answer is no. In fact, it is easy to show that $NGSS_P$ is not and-compositional (in the sense that the $NGSS_P$ information about a goal in P cannot be derived from the $NGSS_P$ information about its atomic subgoals). An example of this fact will be given below.

We have therefore to be content with a declarative semantics from which it is possible to derive $NGSS_P$, but which contains more information than $NGSS_P$ necessary to achieve and-compositionality. The main result of this section is that the information encoded in N_P is the least one which is necessary to model $NGSS_P$ and to provide an and-compositional notion of truth. In other words, N_P is the *fully abstract* semantics with respect to and-compositionality and $NGSS_P$, which means that N_P is the simplest declarative semantics for Prolog programs with first-order built-in's.

We first introduce the notions of semantical mappings associated to N_P and $NGSS_P$ (which we will still denote by N_P and $NGSS_P$).

Definition 2.37 Let $\mathbf{x}_1, \ldots, \mathbf{x}_n$ be sequences of variables, possibly not disjoint. Let $p_1(\mathbf{x}_1), \ldots, p_n(\mathbf{x}_n)$ be either elementary atoms, or atoms of the form $\mathbf{x} = \mathbf{t}$.

- The mapping N_P from conjunctions of elementary atoms to pairs of substitutions is defined as follows:

$$N_P[\![p_1(\mathbf{x}_1), \ldots, p_n(\mathbf{x}_n)]\!] = \{\langle\theta, \eta\rangle \mid \begin{array}{l} Dom(\theta) \subseteq \{\mathbf{x}_1\} \cup \ldots \cup \{\mathbf{x}_n\} \text{ and} \\ \leftarrow (p_1(\mathbf{x}_1), \ldots, p_n(\mathbf{x}_n))\theta \\ \text{has an LD-refutation with c.a.s. } \eta\} \end{array}$$

- The mapping $NGSS_P$ from conjunctions of elementary atoms to substitutions is defined as follows:

$$NGSS_P[\![p_1(\mathbf{x}_1), \ldots, p_n(\mathbf{x}_n)]\!] = \{\theta\eta \mid \begin{array}{l} Dom(\theta) \subseteq \{\mathbf{x}_1\} \cup \ldots \cup \{\mathbf{x}_n\} \text{ and} \\ \leftarrow (p_1(\mathbf{x}_1), \ldots, p_n(\mathbf{x}_n))\theta \\ \text{has an LD-refutation with c.a.s. } \eta\} \end{array}$$

$\square$

The correspondence with the standard notions of N_P, $NGSS_P$ is immediate, since

$$N_P[\![p(\mathbf{x})]\!] = \{\langle\theta, \eta\rangle \mid \langle p(\mathbf{x})\theta, \eta\rangle \in N_P\}$$

and

$$NGSS_P[\![p(\mathbf{x})]\!] = \{\sigma \mid p(\mathbf{x})\sigma \in NGSS_P\}.$$

The semantics $NGSS_P$ is more abstract than N_P, i.e. the information encoded in $NGSS_P$ can be retrieved from the one in N_P (correctness of N_P w.r.t. $NGSS_P$). This is shown by the following fact.

Fact 1 $NGSS_P[\![\mathbf{Q}]\!] = \{\theta\eta \mid \langle\theta, \eta\rangle \in N_P[\![\mathbf{Q}]\!]\}$.

On the other hand, it is not possible to retrieve the information encoded in N_P from the one encoded in $NGSS_P$, i.e N_P and $NGSS_P$ are not equivalent . This because the mapping N_P is and-compositional and $NGSS_P$ is not. In fact $N_P[\![\mathbf{Q}, \mathbf{R}]\!]$ can be derived from $N_P[\![\mathbf{Q}]\!]$ and $N_P[\![\mathbf{R}]\!]$:

$$\langle\theta, \eta\rangle \in N_P[\![\mathbf{Q}, \mathbf{R}]\!] \quad \text{iff} \quad \begin{array}{l} \exists\sigma.\ \langle\theta, \sigma\rangle \in N_P[\![\mathbf{Q}]\!], \text{ and} \\ \langle\theta\sigma, \eta\rangle \in N_P[\![\mathbf{R}]\!], \text{ and} \\ (\mathbf{Q}\theta, \mathbf{R}\theta, \sigma, \eta) \text{ is a good tuple.} \end{array}$$

On the contrary, $NGSS_P$ is not and-compositional, as it is shown in the following example.

Example 2.38 Consider the program P:

```
p(X) ← X=a .
q(X) ← var(X) , X=a .
```

We have $NGSS_P[\![p(x)]\!] = NGSS_P[\![q(x)]\!] = \{\{x/a\}\}$, but $NGSS_P[\![p(x), p(x)]\!] = \{\{x/a\}\}$ whereas $NGSS_P[\![q(x), q(x)]\!] = \emptyset$. Note that the key point of this counterexample is the presence of shared variables.

The next theorem shows, however, that N_P is the most abstract and-compositional semantics which is correct w.r.t. $NGSS_P$. We first need the following lemma.

Lemma 2.39 *If $\langle\theta, \eta\rangle \in N_P[\![\mathbf{Q}]\!] \setminus N_P[\![\mathbf{R}]\!]$ then there exists $\langle\theta', \eta'\rangle \in N_P[\![\mathbf{Q}]\!] \setminus N_P[\![\mathbf{R}]\!]$ s.t. θ' is idempotent.*

Proof.

¿From $\langle\theta, \eta\rangle \in N_P[\![\mathbf{Q}]\!]$ it follows that there exists an LD-refutation ξ of $P \cup \{\leftarrow \mathbf{Q}\theta\}$ with c.a.s. η. Let $Dom(\theta) \cap Ran(\theta) = \{x_1, \dots, x_n\}$ and let $y_1, \dots, y_n$ distinct variables that do not occur in ξ. Let $\rho = \{x_1/y_1, \dots, x_n/y_n\}$, $\rho^{-1} = \{y_1/x_1, \dots, y_n/x_n\}$. Let $\theta' = (\theta\rho) \mid \mathbf{Q}$ and $\eta' = (\rho^{-1}\eta) \mid (\mathbf{Q}\theta')$. Then $\xi' = \xi\rho$, is an LD-refutation of $P \cup \{\leftarrow \mathbf{Q}\theta'\}$ with c.a.s. η'. Hence $\langle\theta', \eta'\rangle \in N_P[\![\mathbf{Q}]\!] \setminus N_P[\![\mathbf{R}]\!]$ and θ' is idempotent. $\square$

Theorem 2.40 (Full Abstraction) *If $N_P[\![\mathbf{Q}]\!] \neq N_P[\![\mathbf{R}]\!]$ then there exists a conjunction $\mathbf{A}$ such that $NGSS_P[\![\mathbf{A}, \mathbf{Q}]\!] \neq NGSS_P[\![\mathbf{A}, \mathbf{R}]\!]$.*

Proof.

Assume, without loss of generality, that there exist $\langle\theta, \eta\rangle \in N_P[\![\mathbf{Q}]\!] \setminus N_P[\![\mathbf{R}]\!]$. By Lemma 2.39 we can assume θ idempotent. Let $\theta = \{x_1/t_1, \dots, x_m/t_m\}$, $Var(\mathbf{Q}\theta) = \{y_1, \dots, y_n\}$.

Define now:

$$\mathbf{A}_1 = x_1 = t_1, \dots, x_m = t_m,$$

$$\mathbf{A}_2 = var(y_n), \dots, var(y_n),$$

$$\mathbf{A}_3 = (y_{k_1} \backslash == y_{l_1}), \dots, (y_{k_r} \backslash == y_{l_r}),$$

where $\{\{k_1, l_1\}, \dots, \{k_r, l_r\}\}$ are all possible combinations of two indexes in the set $\{1, \dots, n\}$ (r is the cardinality of such combinations: $r = (n-1)n/2$). Finally, define $\mathbf{A} = \mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3$. By the definition of $\mathbf{A}$ we derive immediately that $\theta\eta \in NGSS_P[\![\mathbf{A}, \mathbf{Q}]\!]$. We show now that $\theta\eta \notin NGSS_P[\![\mathbf{A}, \mathbf{R}]\!]$. Assume, by contradiction, that there exists $\sigma \in NGSS_P[\![\mathbf{A}, \mathbf{R}]\!]$ such that

$$\sigma = \theta\eta. \tag{8}$$

Then, there exist ϕ, ψ such that $\leftarrow (\mathbf{A}, \mathbf{R})\phi$ has an LD-refutation with c.a.s. ψ and

$$\phi\psi = \sigma. \tag{9}$$

We show that in this case $\leftarrow \mathbf{R}\theta$ has an LD-refutation with c.a.s. η, i.e. $\langle\theta, \eta\rangle \in N_P(\mathbf{R})$, against the hypothesis. Consider an LD-refutation for $\leftarrow (\mathbf{A}, \mathbf{R})\phi$ with a c.a.s. ψ which satisfies (9). Then there exists γ, τ such that

$$\leftarrow \mathbf{A}_1\phi \text{ has an LD-refutation with c.a.s. } \gamma, \tag{10}$$

$$\leftarrow \mathbf{A}_2\phi\gamma \text{ has an LD-refutation (with c.a.s. } \epsilon), \tag{11}$$

$$\leftarrow \mathbf{A}_3\phi\gamma \text{ has an LD-refutation (with c.a.s. } \epsilon), \tag{12}$$

$$\leftarrow \mathbf{R}\phi\gamma \text{ has an LD-refutation with c.a.s. } \tau, \tag{13}$$

and

$$\gamma\tau = \psi. \tag{14}$$

Since θ is a mgu for $\{x_1 = t_1, \ldots, x_n = t_n\}$, by (10) we have $\mathbf{A}_1\theta \leq \mathbf{A}_1\phi\gamma$. Moreover, by (11), (12) and the definition of $[\![var]\!]$ and $[\![\backslash ==]\!]$ we also have $\mathbf{A}_1\phi\gamma \leq \mathbf{A}_1\theta$, and therefore, since the domains of $\phi\gamma$ and θ are restricted to $\mathbf{A}_1$, we can derive (up to renaming)

$$\phi\gamma = \theta. \tag{15}$$

By (13), we have that

$$\leftarrow \mathbf{R}\theta \text{ has an LD-refutation with c.a.s. } \tau$$

furthermore, by (15), (14), (9), and (8),

$$\mathbf{R}\theta\tau = \mathbf{R}\phi\gamma\tau = \mathbf{R}\phi\psi = \mathbf{R}\sigma = \mathbf{R}\theta\eta$$

i.e. (since both the domains of τ and η are restricted to $\mathbf{Q}\theta$), $\tau = \eta$. □

3 Termination of Prolog Programs

In this section we show that the Θ-semantics is helpful when studying termination of Prolog programs. The presence of built-in's allows us to better control the execution of the programs and consequently it is not surprising that most "natural" programs with built-in's terminate for *all* goals. This motivates the following definition.

Definition 3.1 We say that a Prolog program P *strongly terminates* if for all goals G all LD-derivations of $P \cup \{G\}$ are finite. □

Traditionally, the main concept used to prove termination of Prolog programs is that of a level mapping. Level mapping was originally defined to be a function from ground atoms to natural numbers (see Bezem [6], Cavedon [9], Apt and Pedreschi [5]).

In our case it is more natural to consider level mappings defined on non-ground atoms. Such level mappings were already considered in Bossi, Cocco and Fabris [8] and subsequently in Plümer [16] but they were applied only to prove termination of pure Prolog programs. In our case it is convenient to allow a level mapping yielding values in a well-founded ordering.

Definition 3.2 A *level mapping* $|\ |$ is a function from atoms to a well-founded ordering with a smallest element 0 such that $|A| = |B|$ if A and B are variants. □

The following auxiliary notion will be used below.

Definition 3.3 C' is called a *head instance* of a clause C if $C' = C\theta$ for some substitution that instantiates only variables of C that appear in its head. □

First we provide a method for proving (strong) termination of Prolog programs in homogeneous form. Our key concept is the following one.

Definition 3.4 A homogeneous Prolog program P is called *acceptable* w.r.t. a *level mapping* $|\ |$ and a Θ-*model* I *of* P if for all head instances $A \leftarrow B_1, \ldots, B_n$ of a clause of P the following implication holds for $i \in [1, n]$:

if $I \models \langle B_1, \ldots, B_{i-1}, \eta\rangle$ then $|A| > |B_i\eta|$.

P is called *acceptable* if it is acceptable w.r.t. some level mapping and a Θ-model of P. □

The relevance of the notion of acceptability is clarified by the following theorem.

Theorem 3.5 (Soundness III) *Let P be a homogeneous Prolog program. If P is acceptable then it strongly terminates.*

The following notion will be useful in the proof.

Definition 3.6 Consider an LD-derivation ξ. Let G be a goal in ξ. Let k be the minimum length of a goal in the suffix of ξ starting at G and let H be the first goal in this suffix with length k. We call H *the shortest goal of ξ under G*. □

Proof of Theorem 3.5. Suppose by contradiction that there exists an infinite LD-derivation of $P \cup \{G\}$. Call it ξ. Denote G by H_0. We first define two infinite sequences $G_1, G_2, \ldots$ and $H_1, H_2, \ldots$ of goals of ξ by the following formula for $j \geq 1$:

G_j is the shortest goal of ξ under H_{j-1},
H_j is the direct descendant of G_j in ξ.

Fix $j \geq 1$. Let $A \leftarrow B_1, \ldots, B_n$ be the input clause and θ the *mgu* used to obtain H_j from G_j. By the choice of G_j and H_j we have $l(G_j) \leq l(H_j)$, so $n \geq 1$. G_j is of the form $\leftarrow C_1, \ldots, C_k$ where $k \geq 1$ and H_j is of the form $\leftarrow (B_1, \ldots, B_n, C_2, \ldots, C_k)\theta$. By definition, no goal of ξ under G_j is of length less than k, so G_{j+1} is of the form $\leftarrow (B_i, \ldots, B_n, C_2, \ldots, C_k)\theta\eta$ for some η, where $i \in [1, n-1]$. This means that there exists an LD-refutation of $P \cup \{\leftarrow (B_1, \ldots, B_{i-1})\theta\}$ with c.a.s. η. This refutation is obtained by deleting from all goals of ξ between and including H_j and G_{j+1} all occurrences of the instantiated versions of $B_i\theta, \ldots, B_n\theta, C_2\theta, \ldots, C_n\theta$.

By the Soundness Theorem 2.10 we have $I \models \langle (B_1, \ldots, B_{i-1})\theta, \eta \rangle$. By the acceptability of P

$$|A\theta| > |B_i\theta\eta|. \tag{16}$$

By Assumption 2.3 the *mgu* μ used to obtain H_{j+1} from G_{j+1} does not bind the variables of the selected atom $B_i\theta\eta$. So $B_i\theta\eta = B_i\theta\eta\mu$ and consequently

$$|B_i\theta\eta| = |B_i\theta\eta\mu|. \tag{17}$$

Thus assuming $j > 1$, we have

$$|C_1| = |C_1\theta|, \tag{18}$$

(C_1 is the first atom of G_j and $B_i\theta\eta$ is the first atom of G_{j+1}). But θ unifies A and C_1, so

$$|C_1\theta| = |A\theta|. \tag{19}$$

By (16), (18), and (19) we conclude, assuming $j > 1$,

$$|C_1| > |B_i\theta\eta|.$$

Thus applying the level mapping $|\ |$ to the first atoms of the goals $G_2, G_3, \ldots$ we obtain an infinite descending sequence of elements of a well-founded ordering. This yields a contradiction. □

We now prove a converse of the Soundness III Theorem 3.5.

For a Prolog program P that strongly terminates and a goal G, denote by $nodes_P(G)$ the number of nodes in the LD-tree of $P \cup \{G\}$. The following lemma summarizes the relevant properties of $nodes_P(G)$.

Lemma 3.7 (LD-tree) *Let P be a Prolog program that strongly terminates. Then*

(i) $nodes_P(G) = nodes_P(H)$ *if G and H are variants,*

(ii) $nodes_P(H) < nodes_P(G)$ *for all non-root nodes H in the LD-tree of $P \cup \{G\}$,*

(iii) $nodes_P(H) \leq nodes_P(G)$ *for all prefixes H of G.*

Proof. (i) By a simple generalization of the Variant Lemma 2.8 of Apt [1] to the class of Prolog programs, an isomorphism between the LD-trees of $P \cup \{G\}$ and $P \cup \{H\}$ can be established. (ii), (iii) Immediate by the definition. □

We are now in position to prove the desired result.

Theorem 3.8 (Completeness III) *Let P be a homogeneous Prolog program. Suppose that P strongly terminates. Then P is acceptable.*

Proof. Put for an atom A

$$|A| = nodes_P(\leftarrow A).$$

By Lemma 3.7 (i) $|\ |$ is a level mapping. We now prove that P is acceptable w.r.t. $|\ |$ and N_P, the least Θ-model of P. To this end consider a clause C with head A_0 and its head instance $C\theta = A \leftarrow B_1, \ldots, B_n$ where $Dom(\theta) \subseteq Var(A_0)$. Let us assume that $C\theta$ is disjoint with C. Then A is disjoint with A_0, $A = A_0\theta$ and $Dom(\theta) \subseteq Var(A_0)$, so θ is idempotent and $A\theta = A$. Thus θ unifies A and A_0 and it is easy to see that in fact θ is an *mgu* of A and A_0. Thus $\leftarrow B_1, \ldots, B_n$ is a resolvent of $\leftarrow A$ with the input clause C. By Lemma 3.7 (ii)

$$nodes_P(\leftarrow A) > nodes_P(\leftarrow B_1, \ldots, B_n). \tag{20}$$

This conclusion was reached under the assumption that $C\theta$ is disjoint with C but Lemma 3.7 (i) allows us to dispense us with this assumption. Suppose now that $N_P \models \langle B_1, \ldots, B_{i-1}, \eta\rangle$ for some $i \in [1, n]$ and substitution η. Then by the Completeness Theorem 2.23 there exists an LD-refutation of $\leftarrow B_1, \ldots, B_{i-1}$ with c.a.s. η, so $\leftarrow (B_i, \ldots, B_n)\eta$ is a node in the LD-tree of $P \cup \{\leftarrow B_1, \ldots, B_n\}$. By Lemma 3.7 (ii)

$$nodes_P(\leftarrow B_1, \ldots, B_n) \geq nodes_P(\leftarrow (B_i, \ldots, B_n)\eta) \tag{21}$$

and by Lemma 3.7 (iii)

$$nodes_P(\leftarrow (B_i, \ldots, B_n)\eta) \geq nodes_P(\leftarrow B_i\eta). \tag{22}$$

By (20), (21), and (22) we now conclude

$$nodes_P(\leftarrow A) > nodes_P(\leftarrow B_i\eta),$$

i.e. $|A| > |B_i\eta|$.

This shows that P is acceptable. □

Thus we proved an equivalence between the notions of acceptability and strong termination for homogeneous Prolog programs.

Now, every Prolog program can be easily transformed into a homogeneous program with the same termination behaviour.

Theorem 3.9 *Let P be a Prolog program and G a goal. Then the LD-tree of $P \cup \{G\}$ is finite iff the LD-tree of $Hom(P) \cup \{G\}$ is finite.*

The following lemma is useful.

Lemma 3.10 *Let G be a goal and C a clause. G and C have LD-resolvent $\leftarrow \mathbf{Q}\theta$ with mgu θ iff G and $Hom(C)$ have resolvent $\leftarrow x_1\alpha = t_1, \ldots, x_n\alpha = t_n, Q$ with mgu α and θ is the c.a.s. of $\leftarrow x_1\alpha = t_1, \ldots, x_n\alpha = t_n$, where $t_1, \ldots, t_n$ (resp. $x_1, \ldots, x_n$) are the arguments of the head of C (resp. $Hom(C)$).*

Proof. See [4]. □

Proof of Theorem 3.9
The LD-trees (in P and in $Hom(P)$) are finitely branching, so by König Lemma it suffices to show that G has an infinite derivation in P iff G has an infinite derivation in $Hom(P)$. The result follows by Lemma 3.10. □

Corollary 3.11 *Let P be a Prolog program. Then P strongly terminates iff $Hom(P)$ strongly terminates.* □

This allows us to reason about termination of Prolog programs by transforming them first to a homogeneous form and then using the notion of acceptability. We offer now an alternative, direct way of reasoning about termination. To this end the following auxiliary notion will be needed.

Definition 3.12 Let P be a Prolog program and $| \ |$ a level mapping. An atom A is called *stable* w.r.t. $| \ |$ if $|A| \geq |A\theta|$ for every *mgu* θ of A and a disjoint with A variant of a head of a non-unit clause of P. □

Intuitively, an atom A is stable w.r.t. a level mapping $| \ |$ if A is sufficiently instantiated so that the value of $| \ |$ on every instance of A can be defined by means of the arguments of A.
Note that atoms with built-in relations are automatically stable w.r.t. every level mapping.

The following is a generalization of Definition 3.4 to arbitrary Prolog programs.

Definition 3.13 A Prolog program P is called *acceptable* w.r.t. a *level mapping* $| \ |$ and a Θ-*model* I *of* P if for all head instances $A \leftarrow B_1, \ldots, B_n$ of a clause of P the following implication holds for $i \in [1, n]$:
if $I \models \langle B_1, \ldots, B_{i-1}, \eta \rangle$ then
(i) $|A| > |B_i\eta|$,
(ii) $B_i\eta$ is stable w.r.t. $| \ |$.
P is called *acceptable* if it is acceptable w.r.t. some level mapping and a Θ-model of P. □

It is important to note the following.

Lemma 3.14 *Let P be a homogeneous Prolog program and $| \ |$ a level mapping. Then every atom is stable w.r.t. $| \ |$.*

Proof. Suppose an atom A unifies with a disjoint with A variant B of a head of a non-unit clause of P. B is an elementary atom, so A is an instance of B, say $A = B\eta$ with η such that $Dom(\eta) = Var(B)$. Then $A\eta = A$, so η unifies A and B.
Let now θ be an *mgu* of A and B. Then $A\theta$ is more general than $A\eta$, i.e. $A\theta$ is more general than A. Also A is more general than $A\theta$, so A and $A\theta$ are variants and consequently $|A| = |A\theta|$. □

Corollary 3.15 *For homogeneous programs both definitions of acceptability coincide.* □

The following theorem is a generalization of the Soundness III Theorem 3.5.

Theorem 3.16 (Soundness IV) *Let P be a Prolog program. Suppose P is acceptable. Then P strongly terminates.*

Proof. The proof is completely analogous to that of the Soundness III Theorem 3.5. The only difference is that instead of (17) we can now only claim by condition (ii) of acceptability

$$|B_i\theta\eta| \geq |B_i\theta\eta\mu|,$$

so assuming $j > 1$ we now only have

$$|C_1| \geq |C_1\theta|.$$

instead of (18). However, this weaker conclusion is still sufficient to yield the same contradiction as in the proof of Theorem 3.5. □

Ideally, we would like to prove the converse of the Soundness IV Theorem 3.16, that is Prolog programs that strongly terminate are acceptable. Unfortunately this is not the case.

Theorem 3.17 *There exists a Prolog program P that strongly terminates but is not acceptable.*

Proof. Consider the following program P:

```
p(f(X)) ← nonvar(X), p(X).
p(f(f(X))) ← nonvar(X), p(X).
```

It is easy to see that all LD-derivations of P terminate. In fact, in every LD-derivation of P a goal of the form $\leftarrow p(y)$ leads to a failure in two steps and a goal of the form $\leftarrow p(f^n(y))$, where $n \geq 1$, leads to a goal of the form $\leftarrow p(f^k(y))$, where $k < n$, in two steps.

Suppose now that P is acceptable w.r.t. some level mapping $|\ |$ and a Θ-model I. Then due to condition (i)

$$|p(f(f(Y)))| > |p(f(Y))|$$

because $nonvar(f(Y))$ holds. Also $p(f(Y))$ is stable w.r.t. $|\ |$, so

$$|p(f(Y))| \geq |p(f(f(X)))|$$

which gives a contradiction. □

It may seem disappointing that we opted here for a notion of acceptability that didn't allow us to prove its equivalence with strong termination for all Prolog programs. Clearly, it is possible to characterize strong termination by means of well-founded relations for all Prolog programs. To this end it suffices to use the concept of a level mapping defined on *goals*, with the condition that $|H| < |G|$ whenever H is a direct descendant of G in an LD-derivation. However, such a characterization of strong termination is hardly of any use when proving termination because it requires an analysis of arbitrary goals. In contrast, the definition of acceptability refers only to the program clauses and calls for the use of a level mapping defined only on atoms, so it is simpler to use.

On the other hand, the introduction of homogeneous programs allows us to draw the following conclusion.

Theorem 3.18 *Let P be a Prolog program. Then P strongly terminates iff $Hom(P)$ is acceptable.*

Proof. By the Soundness III Theorem 3.5 and Completeness III Theorem 3.8 applied to $Hom(P)$, and Corollary 3.11. $\square$

4 Applications

We illustrate the use of the results established in the previous section to prove strong termination of some Prolog programs. We start by considering the program `list` given in Section 1.

Then we show how a relation that strongly terminates can be treated as a built-in relation when proving strong termination of a program depending on this relation. This allows us to prove strong termination in a modular way. We illustrate this method by proving strong termination of two well-known Prolog programs.

First, we define by structural induction the function $|\,|$ on terms by putting:

$$\begin{aligned} |x| &= 0 \text{ if } x \text{ is a variable}, \\ |f(x_1,\ldots,x_n)| &= 0 \text{ if } f \neq [\,.\,|\,.\,], \\ |[x|xs]| &= |xs| + 1. \end{aligned}$$

It is useful to note that for a list xs, $|xs|$ equals its length. This function will be used in the examples below.

List

Consider the program `list` from Section 1:

(l_1) `list([]) ←` .
(l_2) `list([X | Xs]) ←`
`nonvar(Xs), list(Xs).`

To prove that `list` strongly terminates we show that it is acceptable. We define a level mapping $|\,|$ by putting

$$\begin{aligned} |list(xs)| &= |xs| \\ |nonvar(xs)| &= 0. \end{aligned}$$

Clearly, $|A| = |B|$ if A and B are variants, so $|\,|$ is indeed a level mapping. Next, we take the Θ-base Θ_P as the Θ-model of `list`.

Theorem 4.1 `list` *is acceptable w.r.t.* $|\,|$ *and* Θ_P.

Proof. Consider a head instance $C = A \leftarrow B_1, B_2$ of (l_2). It is of the form

$$list([x|xs]) \leftarrow nonvar(xs), list(xs).$$

Claim 1 $|A| > |B_1\eta|$.

Proof. Note that $|list([x|xs])| > \ 0 = \ |nonvar(xs\eta)|$. $\square$

Suppose now $\Theta_p \models \langle B_1, \eta\rangle$. Then $\eta = \epsilon$ and $B_1\eta = nonvar(xs)$ with $xs \notin Var$.

Claim 2 $|A| \ > \ |B_2\eta|$.

Proof. Note that $|A| = |list([x|xs])| = \ |[x|xs]| > \ |xs| = \ |list(xs)| = |B_2\eta|$. $\square$

Claim 3 *$B_2\eta$ is stable w.r.t.* $|\ |$.

Proof. Suppose $B_2\eta$ unifies with a variant $list([x'|xs'])$ of the head of the clause (l_2). Since $x_s \notin Var$, $B_2\eta$ is an instance of $list([x'|xs'])$. As in the proof of Lemma 3.14 this implies that for any mgu θ of $B_2\eta$ and $list([x'|xs'])$ we have $|B_2\eta| = |B_2\eta\theta|$. $\square$

$\square$

Modularity

In the proof of Theorem 3.5 the level mapping of built-in relations is not used. This is due to the fact that the built-in relations always terminate and never occur in the head of a clause. So we can assume that $|A| = 0$ if A is a built-in atom.

This observation provides an idea of how to prove the strong termination of a Prolog program in a modular way. Before formalizing this idea we show how the relation `list` previously defined can be treated as a built-in in the proof of the strong termination of a Prolog program.

Example 4.2 Consider the following program `APPEND`:

```
(a1)  a([], Ys, Ys) ←
        list(Ys).
(a2)  a([X | Xs], Ys, [X | Zs]) ←
        nonvar(Xs), a(Xs, Ys, Zs).
```

augmented by the clauses (l_1) and (l_2) defining the `list` program.

To prove that `APPEND` strongly terminates we regard `APPEND` as union of the program `append`, containing only the clauses (a_1) and (a_2) of `APPEND`, with the program `list`. In `append` the relation `list` does not occur in the head of any clause. We already proved that `list` strongly terminates. Thus the relation `list` can be treated as a built-in with the semantics given by an arbitrary Θ-model I_0 of the program `list`. Hence, to show that `APPEND` strongly terminates, it is sufficient to prove that `append` is acceptable w.r.t. a a model of `APPEND` and a level mapping $|\ |$ s.t. $|A|$ is 0 if A is a built-in or is an atom of the form $lis(x_s)$. We choose the following level mapping:

$$|a(x,y,z)| = \ |x|,$$

$$|A| = 0 \text{ if } A \text{ is a built-in or } list(xs).$$

Next, we define a Θ-interpretation for the relation a by putting

$$I = \{\langle a(xs, ys, zs), \eta\rangle \mid |xs\eta| \ + \ |ys\eta| = \ |zs\eta|\}.$$

Lemma 4.3 *$I \cup I_0$ is a Θ-model of* `APPEND`*.*

Proof. Clearly $I \cup I_0$ is a model of `list`.

Let $A = a(r, s, t)$ and let $a([\,], Y_s', Y_s') \leftarrow list(Y_s')$ be a variant of (a_1) disjoint with A. Suppose that $\theta = mgu(A, a([\,], Y_s', Y_s'))$ exists and suppose that $I \cup I_0 \models \langle list(Y_s')\theta, \eta\rangle$, with η satisfying the restriction of Definition 2.27. We have to show that $I \cup I_0 \models \langle A, (\theta\eta) \mid A\rangle$. We have that $r\theta = [\ ]$, $s\theta = t\theta = Y_s'\theta$. Then $|r\theta| + |s\theta| = |t\theta|$ and so $|r\theta\eta| + |s\theta\eta| = |t\theta\eta|$. Hence $I \cup I_0 \models \langle A, (\theta\eta) \,|\, A\rangle$ holds.

Let now $a([X'|X_s'], Y_s', [X'|Z_s']) \leftarrow nonvar(X_s'), a(X_s', Y_s', Z_s')$ be a variant of (a_2) disjoint with A. Suppose that $\theta = mgu(A, a([X'|X_s'], Y_s', [X'|Z_s']))$ exists and suppose that $I \cup I_0 \models \langle (nonvar(X_s'), a(X_s', Y_s', Z_s'))\theta, \eta\rangle$, with η satisfying the restriction of Definition 2.27. We have to show that $I \cup I_0 \models \langle A, (\theta\eta) \,|\, A\rangle$. Clearly $(nonvar(X_s')\theta, a(X_s', Y_s', Z_s')\theta, \epsilon, \eta)$ is a good tuple. Then, by the semantics of *nonvar*, it follows that $I \cup I_0 \models \langle (nonvar(X_s'), a(X_s', Y_s', Z_s'))\theta, \eta\rangle$ iff $I \cup I_0 \models \langle a(X_s', Y_s', Z_s')\theta, \eta\rangle$, with $X_s'\theta \notin Var$. Then we have $|X_s'\theta\eta| + |Y_s'\theta\eta| = |Z_s'\theta\eta|$ and, by $r\theta = [X'|X_s']\theta$, $s\theta = Y_s\theta$ and $t\theta = [X'|Z_s']\theta$ it follows that $|r\theta\eta| + |s\theta\eta| = |t\theta\eta|$. Hence $I \cup I_0 \models \langle A, (\theta\eta) \,|\, A\rangle$ holds.

This concludes the proof that $I \cup I_0$ is a Θ-model of `APPEND`. □

Theorem 4.4 `append` *is acceptable w.r.t.* $|\ |$ *and* $I \cup I_0$*.*

Proof. Analogous to that of Theorem 4.1, due to the similarity between clauses (a_2) and (l_2). □

We can now formulate our modular approach to termination.

Definition 4.5 Let P_1 and P_2 be two Prolog programs. We say that P_2 extends P_1, and write $P_1 < P_2$, if

(i) P_1 and P_2 define different relations,

(ii) no relation defined in P_2 occurs in P_1. □

Informally, P_2 extends P_1 if P_2 defines *new* relations, possibly using the relations defined already in P_1. For example the program `APPEND` extends the program `list`.

The following theorem formalizes the idea used to prove termination of the `APPEND` program.

Theorem 4.6 (Modularity) *Suppose P_2 extends P_1. Assume that*

(i) P_1 is acceptable,

(ii) P_2 is acceptable w.r.t. a Θ-model I of $P_1 \cup P_2$ and a level mapping $|\ |$ such that $|A| = 0$ if A contains a relation defined in P_1.

Then $P_1 \cup P_2$ strongly terminates.

Proof. P_2 extends P_1. Thus $P_1 \cup P_2$ strongly terminates iff P_1 strongly terminates and P_2 strongly terminates when the relations defined in P_1 are treated as built-in's defined by

$$[\![p]\!] = \{\langle A, \eta\rangle \mid A \text{ contains } p \text{ and there exists an LD-refutation of } P_1 \cup \{\leftarrow A\} \text{ with c.a.s. } \eta\ \}.$$

Now, by (i) and the Soundness IV Theorem 3.16 P_1 strongly terminates. To deal with the other conjunct consider $N_{P_1 \cup P_2}$, the least Θ-model of $P_1 \cup P_2$. By (ii) and Corollary 2.12 P_2 is acceptable w.r.t. $N_{P_1 \cup P_2}$ and the level mapping $|\ |$. Moreover, by Corollary 2.24 and the fact that P_2 extends P_1 we have for all atoms A containing a relation p defined in P_1

$$N_{P_1 \cup P_2} \models \langle A, \eta \rangle \ \text{ iff } \ \langle A, \eta \rangle \in [\![p]\!].$$

Thus by the Soundness IV Theorem 3.16 P_2 strongly terminates when the relations defined in P_1 are treated as built-in's defined as above.

This concludes the proof of the theorem. □

We illustrate the use of this theorem in the example below.

Unification

Consider the program `UNIFY` (for unification without occurs check) from Sterling and Shapiro [page 150][17]. In this program several built-in's, namely *var*, *nonvar*, =, *constant*, *compound*, *functor*, > are used. The meaning of them was already given in Section 2. Additionally, the function "−" (minus) is used on terms. Its meaning is implicitly referred within the description of the meaning of ":=". For instance, $\langle x := 3 - 1, \{x/2\} \rangle \in [\![:=]\!]$.

The program `UNIFY` consists of the following clauses.

```
(u1)    unify(X,Y) ←
          var(X), var(Y), X = Y.
(u2)    unify(X,Y) ←
          var(X), nonvar(Y), X = Y.
(u3)    unify(X,Y) ←
          var(Y), nonvar(X), Y = X.
(u4)    unify(X,Y) ←
          nonvar(X), nonvar(Y), constant(X), constant(Y), X = Y.
(u5)    unify(X,Y) ←
          nonvar(X), nonvar(Y), compound(X), compound(Y), term-unify(X,Y).
(tu)    term-unify(X,Y) ←
          functor(X,F,N), functor(Y,F,N), unify-args(N,X,Y).
(uar1)  unify-args(N,X,Y) ←
          N > 0, unify-arg(N,X,Y), N1 := N-1, unify-args(N1,X,Y).
(uar2)  unify-args(0,X,Y).
(ua)    unify-arg(N,X,Y) ←
          arg(N,X,ArgX), arg(N,Y,ArgY), unify(ArgX,ArgY).
```

We assume that `UNIFY` operates on the domain of natural numbers over which the built-in relation > and the function −, both written in infix notation, are defined.

In Pieramico [15] it was proved that `UNIFY` terminates for ground goals by showing that the program obtained by deleting all built-in relations is acceptable in the sense of Apt and Pedreschi [5].

We prove here a stronger statement, namely that `UNIFY` strongly terminates by showing that it is acceptable in the sense of Definition 3.13.

For the subsequent analysis it is important to understand how this program operates. Intuitively, the goal $\leftarrow$ `unify(s,t)` yields an *mgu* of s and t as a computed answer substitution if s and t unify, and otherwise it fails. It is evaluated as follows. If either s or t is a variable, then the built-in relation $=$ is called (clauses (u_1) - (u_3)). It assigns to the term out of s, t which is a variable the other term. If both s and t are variables (clause (u_1)) then s is chosen. If neither s nor t is a variable, but both are constants, then it is tested - again by means of $=$ - whether they are equal (clause (u_4)). The case when both s and t are compound terms is handled in clause (u_5) by calling the relation `term-unify`. This relation is defined by clause (tu).

The goal $\leftarrow$ `term-unify(s,t)` is evaluated by first identifying the form of s and t by means of the built-in relation `functor`. If for some function symbol f and $n \geq 0$, the term s is of the form $f(s_1, \ldots, s_n)$ and the term t is of the form $f(t_1, \ldots, t_n)$, then the relation `unify-args` is called. This relation is defined by clauses (uar_1) and (uar_2).

The goal $\leftarrow$ `unify-args(n,s,t)` succeeds if the sequence of the first n arguments of s can be unified with the sequence of the first n arguments of t. When $n > 0$, clause (uar_1) is used and these arguments are unified pairwise starting with the last pair. This last pair is dealt with by calling the relation `unify-arg` which is defined by clause (ua).

The goal $\leftarrow$ `unify-arg(n,s,t)` is evaluated by first extracting the n-th arguments of s and t by means of the built-in relation `arg`, and then calling `unify` recursively on these arguments. If this call succeeds, *the produced* c.a.s. *modifies* s and t, and the recursive call of `unify-args` in clause (uar_1) operates on this modified pair of s and t. Finally, when $n = 0$, `unify-args(n,s,t)` succeeds immediately (clause (uar_2)). It is clear from this description what is the intended meaning of the defined relations `unify`, `term-unify`, `unify-args` and `unify-arg`. In the proof of the strong termination of `UNIFY` only partial information about the meaning of these relations is needed. This information is captured in the Θ-model I we use.

Let us first define a level mapping $|\,|$. To this end we use the lexicographic ordering $<$ defined on triples of natural numbers. In this ordering $\langle n_1, n_2, n_3\rangle < \langle m_1, m_2, m_3\rangle$ iff $(n_1 < m_1)$ or $(n_1 = m_1 \wedge n_2 < m_2)$ or $(n_1 = m_1 \wedge n_2 = m_2 \wedge n_3 < m_3)$.

For brevity we write $Var(s,t)$ instead of $Var(s) \cup Var(t)$. We put

$$
\begin{aligned}
&|unify(s,t)| = \langle\ card(Var(s,t)),\ nodes(s) + nodes(t),\ 1\ \rangle,\\
&|term - unify(s,t)| = \langle\ card(Var(s,t)),\ nodes(s) + nodes(t),\ 0\ \rangle,\\
&|unify - args(n,s,t)| = \langle\ card(Var(s,t)),\ f(n,s,t),\ 3\ \rangle,\\
&|unify - arg(n,s,t)| = \langle\ card(Var(s,t)),\ nodes(s_n) + nodes(t_n),\ 2\ \rangle,\\
&|A| = \langle\ 0,\ 0,\ 0\ \rangle \text{ if } A \text{ built-in},
\end{aligned}
$$

where $card(S)$ indicates the cardinality of the set S and $f(n,s,t)$ denotes the sum of the number of nodes of the i-th component of s and t for $i \in [1,n]$, that is

$$f(n,s,t) = \sum_{i=1}^{n} (nodes(s_i) + nodes(t_i)).$$

Next we define the Θ-interpretation I.

$I = \{\langle unify(s,t), \theta\rangle,\ \langle term - unify(s,t), \theta\rangle \ \mid\ Inv(s,t,\theta)\} \cup$
$\{\langle unify - args(n,s,t), \theta\rangle,\ \langle unify - arg(n,s,t), \theta\rangle \ \mid\ n \text{ natural number}, Inv(s,t,\theta))\},$

where $Inv(s,t,\theta)$ is the assertion below:

$$Ran(\theta) \subseteq Var(s,t) \wedge (Var(s\theta, t\theta) = Var(s,t) \Rightarrow nodes(s) + nodes(t) = nodes(s\theta) + nodes(t\theta)).$$

The following example clarifies the Θ-interpretation and level mapping we have chosen. Consider the goal $G_1 = \leftarrow unify(f(s), f(t))$.
1) G_1 calls $G_2 = \leftarrow term - unify(f(s), f(t))$ using clause (u_5);
2) G_2 calls $G_3 = \leftarrow unify - args(1, f(s), f(t))$ using clause (tu);
3) G_3 calls $G_4 = \leftarrow unify - arg(1, f(s), f(t))$ and $G_5 = \leftarrow unify - args(0, f(s)\theta, f(t)\theta)$ using clause (uar_1), where θ is a c.a.s. of G_4;
4) G_4 calls $G_6 = \leftarrow unify(s, t)$ using clause (ua).

Let $|unify(f(s), f(t))| = \langle m_1, m_2, 1\rangle$,
$|unify - args(1, f(s), f(t))| = \langle k_1, k_2, 3\rangle$,
$|unify - args(1, f(s)\theta, f(t)\theta)| = \langle k_1', k_2', 3\rangle$. Then
$|term - unify(f(s), f(t))| = \langle m_1, m_2, 0\rangle$,
$|unify - arg(1, f(s), f(t))| = \langle k_1, k_2, 2\rangle$ and
$|unify(s, t)| = \langle k_1, k_2, 1\rangle$.

We now show that when G_i calls G_j, $i, j \in [1, 6]$, $|G_i| > |G_j|$.

For 1) we need $|unify(f(s), f(t))| > |term - unify(f(s), f(t))|$, which holds because $1 > 0$.
For 2) we need $|term - unify(f(s), f(t))| > |unify - args(1, f(s), f(t))|$, which is satisfied because $m_1 = k_1$ and $m_2 > k_2$.
For 3) we need $|unify - args(1, f(s), f(t))| > |unify - arg(1, f(s), f(t))|$, which is satisfied because $3 > 2$, and $|unify - args(1, f(s), f(t))| > |unify - args(0, f(s)\theta, f(t)\theta)|$, which is satisfied when $k_1 > k_1'$, or $k_1 = k_1'$ and $k_2 = k_2'$. These conditions are satisfied if
$Ran(\theta) \subseteq Var(f(s), f(t))$ and
$Var(f(s)\theta, f(t)\theta) = Var(f(s), f(t)) \Rightarrow nodes(f(s)) + nodes(f(t)) = nodes(f(s)\theta) + nodes(f(t)\theta)$,
i.e. if $Inv(f(s), f(t), \theta)$ holds.
For 4) we need $|unify - arg(1, f(s), f(t))| > |unify(s, t)|$, which holds because $2 > 1$.

We prove now that I is a Θ-model of `UNIFY`. The following definition is useful.

Definition 4.7 Let I be a Θ-interpretation. We say that I is *good* if for all $\langle A, \theta\rangle \in I$ we have $Ran(\theta) \subseteq Var(A)$. □

In good interpretations the truth of a conjunct (see Definition 2.7) can be checked, as the condition that $(A, \mathbf{B}, \theta, \sigma)$ is a good tuple is not needed. Indeed this condition holds for atoms defined in the program if the interpretation is good and for built-in atoms it follows by Definition 2.1 and the Good Tuple Lemma 2.5.

Lemma 4.8 *I is a Θ-model of* `UNIFY`.

Proof. The condition $Ran(\theta) \subseteq Var(s, t)$ that occurs in I implies that I is good.
Consider clauses (u_1) - (u_4). $\langle s = t, \theta\rangle \in [\![=]\!]$ iff $\theta = mgu(s, t)$, with θ relevant. Then $Ran(\theta) \subseteq Var(s, t)$ and $Inv(s, t, \theta)$ hold. This implies that I is a Θ-model of (u_1) - (u_4). I is a Θ-model of (u_5), since the relations $unify$ and $term - unify$ are equivalent w.r.t. I. I is a Θ-model of (uar_1), since the condition $\langle unify - args(n, s, t), \theta\rangle \in I$ does not depend on the value of n. I is a Θ-model of (uar_2), because for an atom $A = unify - args(n, s, t)$ and a variant $unify - args(0, X', Y') \leftarrow$ of (uar_2) s.t. $\theta = mgu(A, unify - args(0, X', Y'))$ exists, we have that $n = 0$ and $Inv(s, t, \theta \,|\, A)$ holds. Consider now the clause (tu). Let $A = term - unify(x, y)$ and let
$term - unify(X', Y') \leftarrow functor(X', F', N'), functor(Y', F', N'), unify - args(N', X', Y')$
be a variant of (tu) disjoint with A. Suppose $\alpha = mgu(A, term - unify(X', Y'))$ exists and assume

$$I \models \langle (functor(X', F', N'), functor(Y', F', N'), unify-args(N', X', Y'))\alpha, \eta\rangle. \qquad (23)$$

We need to show that $I \models \langle A, (\alpha\eta) \,|\, A\rangle$.
Since $F'\alpha$ and $N'\alpha$ are in Var, then by the semantics of $functor$ we have that (23) implies that $N'\alpha\theta = a(x)$, $F'\alpha\theta = funct(x) = funct(y)$, $I \models \langle unify - args(N', X', Y')\alpha\theta, \beta\rangle$, $\eta = \theta\beta$ and $(\alpha\eta) \,|\, A = \beta$. But for compound terms x and y we have that $I \models \langle term - unify(x, y), \eta\rangle$ iff $I \models \langle unify - args(a(x), x, y), \eta\rangle$. Then $I \models \langle term - unify(x, y), (\alpha\eta) \,|\, A\rangle$.
It remains to check that I is a model of (ua). Let $A = unify - arg(n, x, y)$ and let
$unify - arg(N', X', Y') \leftarrow arg(N', X', ArgX'), arg(N', Y', ArgY'), unify(ArgX', argY')$
be a variant of (ua) disjoint with A. Suppose $\alpha = mgu(A, unify - arg(N', X', Y'))$ exists and assume

$$I \models \langle (arg(N', X', ArgX'), arg(N', Y', ArgY'), unify(ArgX', argY'))\alpha, \eta\rangle. \qquad (24)$$

We need to show that $I \models \langle A, (\alpha\eta) \,|\, A\rangle$. Since $ArgX'\alpha$ and $ArgY'\alpha$ are in Var, then by the semantics of arg we have that (24) implies that $N'\alpha = n$, with $n > 0$, $ArgX'\alpha\theta = x_n$, $ArgY'\alpha\theta = y_n$, $I \models \langle unify(ArgX', argY')\alpha\theta, \beta\rangle$, $\eta = \theta\beta$ and $(\alpha\eta) \,|\, A = \beta$. Now notice that $Dom(\beta) \subseteq Var(x_n, y_n)$ and $Inv(x_n\alpha, y_n\alpha, \beta)$ imply $Inv(x\alpha, y\alpha, \beta)$. Then $I \models \langle unify - arg(x, y), (\alpha\eta) \,|\, A\rangle$.

This concludes the proof that I is a Θ-model of `UNIFY`. $\square$

We can now prove the desired result.

Theorem 4.9 `UNIFY` *is acceptable w.r.t.* $|\;|$ *and* I.

Proof. Notice that any atom in the body of an instance of a clause in `UNIFY` satisfies property (ii) of Definition 3.13, since each clause with nonempty body is in homogeneous form. Any instance of (u_1), (u_2), (u_3), (u_4) satisfies the appropriate requirement since $|unify(s, t)| > \langle 0, 0, 0\rangle$. Consider now a head instance $C = A \leftarrow B_1, B_2, B_3, B_4, B_5.$ of (u_5). C is of the form
$unify(s, t) \leftarrow nonvar(s), nonvar(t), compound(s), compound(y), term - unify(s, t).$
We prove two claims which obviously imply that C satisfies the appropriate requirement.

Claim 1 $|A| > |B_i|$ *for* $i = 1, \ldots, 4$.

Proof. Note that $|A| > \langle 0, 0, 0\rangle = |B_i|$ for $i = 1, \ldots, 4$. $\square$

Claim 2 *Suppose that* $I \models \langle B_1, B_2, B_3, B_4, \eta\rangle$. *Then* $|A| > |B_5\eta|$.

Proof. By the semantics of the built-in's $nonvar$ and $compound$ it follows that $s\eta = s$, $t\eta = t$. So $|unify(s, t)| = \langle\; card(Var(s, t)),\; nodes(s) + nodes(t),\; 1\;\rangle > \langle\; card(Var(s, t)),\; nodes(s) + nodes(t),\; 0\;\rangle = |term - unify(s, t)|$. $\square$

Consider a head instance $C = A \leftarrow B_1, B_2, B_3.$ of (tu). C is of the form
$term - unify(s, t) \leftarrow functor(s, F, N), functor(t, F, N), unify - args(N, s, t).$
We prove two claims which obviously imply that C satisfies the appropriate requirement.

Claim 1 $|A| > |B_i|$ *for* $i = 1, 2$.

Proof. Note that $|A| > \langle 0, 0, 0\rangle = |B_i|$ for $i = 1, 2$. $\square$

Claim 2 *Suppose that* $I \models \langle B_1, B_2, \eta\rangle$. *Then* $|A| > |B_3\eta|$.

Proof. By assumption $s\eta = s$, $t\eta = t$ and $n = a(s) = a(t)$. Notice that $nodes(s) + nodes(t) > f(n, s, t)$. So $|term - unify(s, t)| > |unify - args(n, s, t)|$. $\square$

Consider a head instance $C = A \leftarrow B_1, B_2, B_3, B_4$. of (uar_1). C is of the form

$unify - args(n, s, t) \leftarrow n > 0, unify - arg(n, s, t), N1 := n - 1, unify - args(N1, s, t)$.

We prove three claims which obviously imply that C satisfies the appropriate requirement.

Claim 1 $|A| > |B_i|$ *for* $i = 1, 3$.

Proof. Note that $|A| > \langle 0, 0, 0\rangle = |B_i|$ for $i = 1, 3$. $\square$

Claim 2 *Suppose that* $I \models \langle B_1, \eta\rangle$. *Then* $|A| > |B_2\eta|$.

Proof. By the semantics of the built-in $>$ it follows that $s\eta = s$, $t\eta = t$, $n > 0$. Notice that $f(n, s, t) \geq nodes(s_n) + nodes(t_n)$. So $|unify - args(n, s, t)| > |unify - arg(n, s, t)|$. $\square$

Claim 3 *Suppose that* $I \models \langle B_1, B_2, B_3, \eta\rangle$. *Then* $|A| > |B_4\eta|$.

Proof. By the semantics of the built-in's $>$, $:=$ and of the relation $unify - arg$ it follows that $n > 0$, $a(s) \geq n > 0$, $N1\eta = n - 1$, $Var(s\eta, t\eta) \subseteq Var(s, t)$.
If $Var(s\eta, t\eta) \subset Var(s, t)$ then $card(Var(s, t)) > card(Var(s\eta, t\eta))$; if $Var(s\eta, t\eta) = Var(s, t)$ then $nodes(s\eta) + nodes(t\eta) = nodes(s) + nodes(t)$, hence $f(n, s, t) > f(n-1, s\eta, t\eta)$. So in both cases we have $|unify - args(n, s, t)| > |unify - args(n - 1, s\eta, t\eta)|$. $\square$

Consider a head instance $C = A \leftarrow B_1, B_2, B_3$ of (ua). C is of the form

$unify - arg(n, s, t) \leftarrow arg(n, s, ArgX), arg(n, t, ArgY), unify(ArgX, ArgY)$.

We prove two claims which obviously imply that C satisfies the appropriate requirement.

Claim 1 $|A| > |B_i|$ *for* $i = 1, 2$.

Proof. Note that $|A| > \langle 0, 0, 0\rangle = |B_i|$ for $i = 1, 2$. $\square$

Claim 2 *Suppose that* $I \models \langle B_1, B_2, \eta\rangle$. *Then* $|A| > |B_3\eta|$.

Proof. Since in the clause C the third argument of arg is a fresh variable, then from the semantics of arg it follows $s\eta = s$, $t\eta = t$, $n > 0$, $a(s) \geq n > 0$, $ArgX\eta = s_n$, $ArgY\eta = t_n$. So $|unify - arg(n, s, t)| > |unify(s_n, t_n)|$. $\square$

$\square$

Consider now the program `UNIFYoc` for the unification with occur check (see Sterling and Shapiro [page 152][17]). Let `UNIFY'` be the program obtained by `UNIFY` introducing the atom `not-occurs-in(X,Y)` before the last atom in the bodies of clauses (u_2) and (u_3). Then `UNIFYoc` is the union of `UNIFY'` with the following program `not-occur` defining the relation `not-occurs-in`/2:

```
(noc1) not-occurs-in(X,Y) ←
         var(Y), X \== Y.
(noc2) not-occurs-in(X,Y) ←
         nonvar(Y), constant(Y).
(noc3) not-occurs-in(X,Y) ←
         nonvar(Y), compound(Y), functor(Y,F,N), not-occurs-in(N,X,Y).
(no1)  not-occurs-in(N,X,Y) ←
         N > 0, arg(N,Y,Arg), not-occurs-in(X,Arg), N1 := N-1,
         not-occurs-in(N1,X,Y).
(no2)  not-occurs-in(0,X,Y).
```

By the Modularity Theorem 4.6 and the Soundness IV Theorem 3.5 to prove that `UNIFYoc` strongly terminates it suffices to prove that `not-occur` is acceptable and then prove that `UNIFY'` is acceptable w.r.t a Θ-model of `UNIFYoc` and a level mapping $|\ |$ such that $|not-occurs(s,t)| = 0$ for all s, t.

To prove that `not-occur` is acceptable we define an appropriate level mapping with

$$\begin{aligned}|not-occurs-in(x,y)| =&\ nodes(y)+1,\\ |not-occurs-in(n,x,y)| =&\ nodes(y) - \textstyle\sum_{i=n+1}^{a(y)} nodes(y_i) \ \text{ if } n>0,\\ |not-occurs-in(0,x,y)| =&\ 0,\\ |A| =&\ 0 \text{ if } A \text{ is a built-in.}\end{aligned}$$

Next, we define a Θ-interpretation of `not-occur` by putting

$I' = \{\langle not-occurs-in(s,t),\epsilon\rangle\} \cup \{\langle not-occurs-in(n,s,t),\epsilon\rangle \mid 0 \le n \le a(t)\}$.

Lemma 4.10 *I' is a Θ-model of* `not-occur` .

Proof. Notice that $1 \le n-1 \le a(t)$ implies $0 \le n \le a(t)$. This implies that I' is a Θ-model of `not-occur` . □

Lemma 4.11 `not-occur` *is acceptable w.r.t. $|\ |$ and I'.*

Proof. Notice that condition (ii) of Definition 3.4 is satisfied since `not-occur` is stable. Any instance of (noc_1) and (noc_2) satisfies the appropriate requirement since $|not-occurs-in(s,t)| > 0$. Consider an instance $C = A \leftarrow B_1, B_2, B_3, B_4$ of (noc_3). C is of the form
$not-occurs-in(s,t) \leftarrow nonvar(t), compound(t), functor(t,F,N), not-occurs-in(N,s,t)$.
We prove two claims which obviously imply that C satisfies the appropriate requirement.

Claim 1 $|A|\ >\ |B_i|$ *for* $i = 1,2,3$.

Proof. Notice that $|A| > 0 = |B_i|\ for\ i = 1,2,3$. □

Claim 2 *Suppose that* $I' \models \langle B_1, B_2, B_3, \eta\rangle$. *Then* $|A| > |B_4\eta|$.

Proof. By the semantics of the buil-in's *nonvar*, *compound* and *functor* we have $s\eta = s$, $t\eta = t$ and $N\eta = a(t)$. So $|not-occurs-in(s,t)| = nodes(t)+1 > nodes(t) = |not-occurs-in(N\eta,s,t)|$. □

Consider now an instance $C = A \leftarrow B_1, B_2, B_3, B_4, B_5$ of (no_1). C is of the form
$not-occurs-in(n,s,t) \leftarrow n>0, arg(n,t,Arg), not-occurs-in(s,Arg),$
$N1 := n-1, not-occurs-in(N1,s,t)$.
We prove three claims which obviously imply that C satisfies the appropriate requirement.

Claim 1 $|A|\ >\ |B_i|$ *for* $i = 1,2,4$.

Proof. Notice that $|A| > 0 = |B_i|$ for $i = 1,2,4$. □

Claim 2 *Suppose that* $I' \models \langle B_1, B_2, \eta\rangle$. *Then* $|A| > |B_3\eta|$.

Proof. By the semantics of the built-in's $>$ and *arg* we have $s\eta = s$, $t\eta = t$, $Arg\eta = t_n$ and $1 \le n \le a(t)$. So $|not-occurs-in(n,s,t)| = nodes(t) - \sum_{i=n+1}^{a(t)} nodes(t_i)$
$> nodes(t_n)+1 = |not-occurs-in(s,t_n)|$. □

Claim 3 *Suppose that* $I' \models \langle B_1, B_2, B_3, B_4, \eta\rangle$. *Then* $|A| > |B_5\eta|$.

Proof. By the semantics of the built-in's $>$, arg, $:=$ and of the relation $not-occurs-in$ we have $s\eta = s$, $t\eta = t$, $Arg\eta = t_n$, $N1\eta = n-1(\geq 0)$ and $(1 \leq n \leq a(t))$. So $|not-occurs-in(n,s,t)| = nodes(y) - \sum_{i=n+1}^{a(t)} nodes(t_i) > nodes(t) - \sum_{i=n-1}^{a(t)} nodes(t_i) = |not - occurs - in(n-1,s,t)|$.
□
□

To prove that `UNIFY'` is acceptable we consider the level mapping and Θ-model defined in `UNIFY` and we treat `not-occurs-in` as built-in relation whose semantics is given by I'.

Lemma 4.12 `UNIFY'` *is acceptable w.r.t.* $|\ |$ *and* I.

Proof. Notice that if $C = A \leftarrow B_1, \ldots, B_4$ is an instance of (u_2') (resp. of (u_3') exchanging the positions of s and t in the body of the clause), then C is of the form
$unify(s,t) \leftarrow var(s), nonvar(t), not - occurs - in(s,t), s = t$.
If $I \models \langle B_1, B_2, B_3, \eta\rangle$ then by the semantics of the built-in's var, $nonvar$ and of the relation $not - occurs - in$ we have $s\eta = s$ and $t\eta = t$, i.e. `not-occurs-in`(s,t) does not modify s and t. It follows that the proof that `UNIFY'` is acceptable w.r.t. $|\ |$ and I is analogous to the one for `UNIFY` given in Theorem 4.9. □

Acknowledgements We thank Annalisa Bossi and Kees Doets for helpful discussions on the subject of the Good Tuple Lemma 2.5. Also, we thank the referees for useful suggestions on the subject of this paper.

References

[1] Apt K. R.: Logic Programming. In: J. van Leeuwen (ed.) Handbook of Theoretical Computer Science. Vol. B. Elsevier 1990

[2] Apt K. R., Bezem M.: Acyclic Programs. New Generation Computing 9, 335-363 (1991)

[3] Apt K. R. , Doets H. C.: A new definition of SLDNF-resolution. Tech. Rep. CS-R9242. CWI, Amsterdam, NL (1992)

[4] Apt K. R., Marchiori E., Palamidessi C.: A declarative approach for first-order built-in's of Prolog. Tech. Rep. CS-R9246. CWI, Amsterdam, NL (1992).

[5] Apt K. R., Pedreschi D.: Studies in pure Prolog: termination. In: Lloyd J.W. (ed.) Symposium on Computional Logic. Berlin: Springer-Verlag (1990)

[6] Bezem M.: Characterizing termination of logic programs with level mappings. In: Lusk E.L., Overbeek R.A. (eds.) Proceedings of the North American Conference on Logic Programming, 69-80. The MIT Press (1989)

[7] Börger E.: A logical operational semantics of full Prolog, Part III: Built-in predicates for files, terms, arithmetic and input-output. In: Moschovakis Y. (ed.), Proceedings Workshop on Logic from Computer Science. Springer MSRI Publications (1989)

[8] Bossi A., Cocco N., Fabris M.: Proving termination of logic programs by exploiting term properties. In: Proceedings of Tapsoft '91, 153-180. (1991).

[9] Cavedon L.: Continuity, consistency, and completeness properties for logic programs. In: Levi G., Martelli M. (eds.) Proceedings of the Sixth International Conference on Logic Programming, 571-584. The MIT Press (1989)

[10] Clark K. L.: Predicate logic as a computational formalism. Tech. Rep. DOC 79/59. ico, London, GB (1979)

[11] Deransart P., Ferrand G.: An operational formal definition of Prolog. In: Proceedings of the 4th Symposium on Logic Programming, 162-172. Computer Society Press (1987)

[12] Falaschi M., Levi G., Martelli M., Palamidessi C.: Declarative modeling of the operational behaviour of logic languages. Theoretical Computer Science 69, 289-318 (1989)

[13] Hill P. M., Lloyd J.W.: Analysis of meta-programs. In: Abramson H.D., Rogers M.H. (eds.) Proceedings of the Meta88 Workshop, 23-52. MIT Press (1988)

[14] Lloyd J. W.: Foundations of Logic Programming. Second edition, Berlin: Springer-Verlag 1987

[15] Pieramico C.: Metodi formali di ragionamento sulla terminazione di programmi prolog. Tech. Rep. Tesi di Laurea. Universitá degli Studi di Pisa. I (1991)

[16] Plümer L.: Automatic termination proofs for prolog programs operating on nonground terms. In: Proceedings of the 1991 International Logic Programming Symposium. San Diego (1991)

[17] Sterling L., Shapiro E.: The Art of Prolog. MIT Press 1986

[18] van Emden M. H., Kowalski R. A.: The semantics of predicate logic as a programming language. Journal of the ACM 23, 733-742 (1976)