A Theory of First-Order Built-in's of Prolog

Krzysztof R. Apt [*] [‡], Elena Marchiori [†], Catuscia Palamidessi [‡] [§]

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 [SS86].

Keywords and Phrases: Prolog programs, built-in's, declarative semantics, termination.
1985 Mathematics Subject Classification: 68Q40, 68T15,
CR Categories: F.3.2., F.4.1, H.3.3, I.2.3.

Note: This research was done during the second and third authors' stay at Centre for Mathematics and Computer Science, Amsterdam. The work of K.R. Apt was partly supported by ESPRIT Basic Research Action 3020 (Integration). The work of C. Palamidessi was partly supported by ESPRIT Basic Research Action 3020 and by the Italian CNR (Consiglio Nazionale delle Ricerche). The work of E. Marchiori was partly supported by .....

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 [Llo87]) used to prove completeness

[*]Faculty of Mathematics and Computer Science, University of Amsterdam, Plantage Muidergracht 24, 1018 TV Amsterdam, The Netherlands
[†]Dipartimento di Matematica Pura ed Applicata, Università di Padova, Via Belzoni 7, 35131 Padova, Italy
[‡]Centre for Mathematics and Computer Science, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands
[§]Department of Computer Science, Utrecht University, P.O. Box 80089 3508 TB Utrecht, The Netherlands

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.

These results are 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 [AP91] 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 [AP91]. 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 [SS86].

We are aware of two other approaches to define the meaning of Prolog's built-in's, namely that of Börger [Bör89] based on so-called dynamic algebras, and that of Deransart and Ferrand [FD87] 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 built-in's so that one can reason about such programs.

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.

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.

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. 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. By the *length* of such a conjuction we mean the number of its atoms. Usually, $\mathbf{A}$, $\mathbf{B}$ denote such conjunctions.

It is convenient to associate with each pair of atoms or pair of terms that unify a unique idempotent and relevant mgu in the sense of Apt [Apt90][page 502]. Given such a pair A, B we denote it by $mgu(A, B)$. 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 $\leq$.

The rest of the used notation is more or less standard and essentially follows Lloyd [Llo87]. In particular c.a.s. stands for *computed answer substitution*. In this paper we also associate c.a.s.'s with prefixes of LD-derivations in the obvious way.

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 [vEK76], 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 [Cla79], 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. [FLMP89]: 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 [vEK76], the resulting definition of truth treats the body of a clause as a logical conjunction - i.e. the ',' is intrepreted 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

```
P₁: p(X) ← var(X), q(X).
    q(a) ←.
```

and

```
P₂: 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 [Llo87] 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-monotic relations. In other words, the C-semantics is essentially monotonic. In fact, 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, the S-semantics is not necessarily monotonic: the S-interpretations are not upward closed. In fact, the S-interpretations are isomorphic to arbitrary sets of pairs the form $\langle p(\mathbf{x}), \eta\rangle$, where η represents, roughly, a c.a.s. for the goal $\leftarrow p(\mathbf{x})$; see Falaschi et al. [FLMP91]. However, the S-semantics is not suitable for describing the meaning of Prolog programs, either. In fact, 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 [Llo87]) does not hold for Prolog programs. Consider for instance

```
P₃: p(X) ← 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 2.5 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.

A complication arises when we want to define the meaning of a compound object (clause, goal,...) in terms of the meaning of its components (the atoms). If we want to obtain the c.a.s.'s for a compound goal $\leftarrow A, B$ in terms of the c.a.s.'s for its components, a first naive idea would be to compose each c.a.s. θ for $\leftarrow A$ with each c.a.s. η for $\leftarrow B\theta$. But since the computations of $\leftarrow A$ and $\leftarrow B\theta$ are considered in isolation it can happen that θ introduces variables which are in B, or that η introduces variables which are in $A\theta$. If this is the case, then $\theta\eta$ can be *less general* than the c.a.s. obtained by the refutation of $\leftarrow A, B$ that uses the same clauses. Indeed, due to the *standardization apart*, the intermediate mgu's do not introduce variables present in the original goal or in the previously used input clauses. A similar problem arises in the definition of the meaning of a clause.

In order to overcome these problems we enrich the substitutions by the information about the variables which are considered "auxiliary". For instance, if a c.a.s. for a goal $\leftarrow p(x)$ is of the form $\eta = \{x/f(y)\}$, then the variable y is considered auxiliary. Intuitively, η being a c.a.s. for $\leftarrow p(x)$ expresses the fact that for any object x of the form $f(\ldots)$ the atom $p(x)$ is true. In othe words, the formula $\forall y(x = f(y) \Rightarrow p(x))$ is a logical consequence of the program. On the other hand, y is only mentioned to express the bindings for x, so it is a "dummy" symbol. It is quite natural, then, to separate the roles of x and y above. We will call y a *linked variable* and x a *free variable* of η. Once we have made this distinction, it is easy to avoid the problems mentioned above by imposing restrictions on the notions of application and of composition. For instance, if y is linked in $\eta = \{x/f(y)\}$, then the application of η to an atom $p(x, y)$ will be illegal, since y is "free" in $p(x, y)$. In a sense, this is similar to the restrictions on the notions of substitution in λ-calculus and in predicate calculus. In both formalisms it is disallowed to bring inside the scope of a quantifier variables which are originally free.

For technical convenience, we assume that the domain of a substitution does not contain linked variables, and we restrict our attention to idempotent substitutions. Furthermore, we indicate explicitly the free variables of a substitution, instead of the linked ones. The advantages of this notation will be clear later.

Definition 2.1 A *linked substitution* is an object of the form $\eta_{\mathbf{x}}$, where η is an idempotent substitution, and $\mathbf{x}$ is a set of variables such that $Dom(\eta) \subseteq \mathbf{x}$. The variables in $\mathbf{x}$ are called *free variables* of $\eta_{\mathbf{x}}$, whereas the variables in $link(\eta_{\mathbf{x}}) = Ran(\eta) - \mathbf{x}$ are called *linked variables* of $\eta_{\mathbf{x}}$. □

For an expression E (term, atom, goal,...), we often write η_E to denote the linked substitution $\eta_{Var(E)}$. We now define the application and composition of linked substitutions.

Definition 2.2 Let E be an expression, and $\eta_{\mathbf{x}}$ a linked substitution. The application $E\eta_{\mathbf{x}}$ is defined only when $link(\eta_{\mathbf{x}}) \cap Var(E) = \emptyset$, and in that case we have $E\eta_{\mathbf{x}} = E\eta$, where $E\eta$ is the standard application of η to E. □

Definition 2.3 Consider two linked substitutions $\theta_{\mathbf{z}}$, $\eta_{\mathbf{w}}$. The composition $\theta_{\mathbf{z}}\eta_{\mathbf{w}}$ is defined only when

(i) $Dom(\theta) \cap Ran(\eta) = \emptyset$ (η does not introduce variables already present in the domain of θ),

(ii) $link(\eta_{\mathbf{w}}) \cap Ran(\theta) = \emptyset$ (the linked variables of η differ from the variables introduced by θ).

In that case we have: $\theta_{\mathbf{z}}\eta_{\mathbf{w}} = \sigma_{\mathbf{v}}$, where

$$\sigma = \theta\eta,$$

$$\mathbf{v} = (\mathbf{z} \cup \mathbf{w}) \cap Var(\sigma).$$

□

Note that due to the restrictions (i) and (ii) above, if $\theta_{\mathbf{z}}\eta_{\mathbf{w}}$ is defined, then it is a linked substitution. The associativity properties still hold for the new notions of application and composition.

Proposition 2.4 *Let E be an expression and $\theta_{\mathbf{x}}, \eta_{\mathbf{y}}$ and $\sigma_{\mathbf{z}}$ be linked substitutions*

- *if $(E\theta_{\mathbf{x}})\eta_{\mathbf{y}}$ and $E(\theta_{\mathbf{x}}\eta_{\mathbf{y}})$ are both defined, then $(E\theta_{\mathbf{x}})\eta_{\mathbf{y}} = E(\theta_{\mathbf{x}}\eta_{\mathbf{y}})$.*
- *if $Var(E) = \mathbf{x}$ then $(E\theta_{\mathbf{x}})\eta_{\mathbf{y}}$ is defined iff $E(\theta_{\mathbf{x}}\eta_{\mathbf{y}})$ is defined, and in that case $(E\theta_{\mathbf{x}})\eta_{\mathbf{y}} = E(\theta_{\mathbf{x}}\eta_{\mathbf{y}})$.*
- *if $(\theta_{\mathbf{x}}\eta_{\mathbf{y}})\sigma_{\mathbf{z}}$ and $\theta_{\mathbf{x}}(\eta_{\mathbf{y}}\sigma_{\mathbf{z}})$ are both defined, then $(\theta_{\mathbf{x}}\eta_{\mathbf{y}})\sigma_{\mathbf{z}} = \theta_{\mathbf{x}}(\eta_{\mathbf{y}}\sigma_{\mathbf{z}})$ holds.*

Proof. Straightforward. □

¿From now on, when we write expressions like $E\theta_{\mathbf{x}}$, $\theta_{\mathbf{x}}\eta_{\mathbf{y}}$ we implicitly assume that they are defined.

We are now ready to define the declarative semantics of Prolog programs. 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".

$$
\begin{aligned}
[\![var]\!] &= \{\langle var(x), \epsilon\rangle \mid x \in Var\},\\
[\![nonvar]\!] &= \{\langle nonvar(s), \epsilon\rangle \mid s \notin Var\},\\
[\![=]\!] &= \{\langle s = t, \eta\rangle \mid \eta = mgu(s,t)\},\\
[\![>]\!] &= \{\langle s > t, \epsilon\rangle \mid s, t \text{ are integers and } s > t\}.
\end{aligned}
$$

Other built-in relations of Prolog, like *ground*, *arg* or *functor*, can be specified in a similar way without any difficulties.

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

Definition 2.5 Let A be an atom with a built-in relation p. Then for every conjunction $\mathbf{B}$ the goal $\leftarrow \mathbf{B}\eta$ is a resolvent of $\leftarrow A, \mathbf{B}$ iff $\langle A, \eta\rangle \in [\![p]\!]$ and $\mathbf{B}\eta_A$ is defined. □

In the previous definition the requirement that $\mathbf{B}\eta_A$ is defined is satisfied by the standard Prolog built-in's, since their execution binds only the local variables of the atom, without introducing any new variable.

Next, we consider atoms defined by the program. First, we introduce the following generalization of the Herbrand base and Herbrand interpretations.

Definition 2.6 (Θ-domain and Θ-interpretations)

- Let P be a Prolog program. The Θ-base Θ_P for P is the set of all pairs of the form $\langle A, \eta \rangle$, where A is an atom defined in P, and η is an idempotent substitution such that $Dom(\eta) \subseteq Var(A)$.

- A Θ-interpretation I of P is a subset of the Θ-base Θ_P which is closed under renaming, i.e. if $\langle A, \eta \rangle \in I$, then for every renaming substitution ρ such that $Ran(\eta) \cap Var(A\rho) = \emptyset$, we have $\langle A\rho, (\rho^{-1}\eta) \mid A\rho \rangle \in I$. □

We restrict here our attention to atoms defined by the program, because the intended meaning of the built-in atoms is fixed, i.e. it does not depend upon the program. In a sense, our approach is similar to the algebraic semantics, in which the interpretation of the equality symbol is fixed and the proper axioms of an algebraic theory only specify the meaning of other symbols. In our case, the clauses of a program represent the axioms.

Essentially, a Θ-interpretation identifies which pairs $\langle A, \eta \rangle$ are true. We still need to define when a Θ-interpretation is a model of a program. To this end, it will be convenient to give the definion of truth for clauses in *homogeneous form.*

Definition 2.7

- A clause $A \leftarrow \mathbf{B}$ is in homogeneneous form if A is an elementary atom (i.e. it is of the form $p(x_1, \ldots, x_n)$, where $x_1, \ldots, x_n$ are distinct variables).

- Consider a clause $A \leftarrow \mathbf{B}$, where $A = p(t_1, \ldots, t_n)$. Its homogeneous form, denoted by $Hom(A \leftarrow \mathbf{B})$, is given by the clause $p(x_1, \ldots, x_n) \leftarrow x_1 = t_1, \ldots, x_n = t_n, \mathbf{B}$, where $x_1, \ldots, x_n$ are distinct variables, not occurring in $A \leftarrow \mathbf{B}$.

□

Definition 2.8 (Truth in Θ-interpretations)

Let P be a program and I a Θ-interpretation of P.

- The *truth of a conjunction* $A_1, \ldots, A_n$ in I ($n \geq 0$), up to η (a post-substitution), denoted by $I \models \langle A_1, \ldots, A_n, \eta \rangle$, is defined as follows:

  - For $n = 0$ (empty conjunction, which we denote by *true*) $I \models \langle true, \eta \rangle$ iff $\eta = \epsilon$.

  Otherwise:

  - $I \models \langle A, \eta \rangle$ iff A is a built-in atom with relation p and $\langle A, \eta \rangle \in [\![p]\!]$,
  - $I \models \langle A, \eta \rangle$ iff A is an atom defined in P and $\langle A, \eta \rangle \in I$,
  - $I \models \langle A, \mathbf{B}, \eta \rangle$ iff $\exists \theta, \sigma$ such that $I \models \langle A, \theta \rangle$, $I \models \langle \mathbf{B}\theta_A, \sigma \rangle$ and $\eta_{A,\mathbf{B}} = \theta_A \sigma_{\mathbf{B}\theta}$.

- The *truth of a clause* in I is defined as follows:

  - $I \models \langle p(\mathbf{x}) \leftarrow \mathbf{B}, \eta \rangle$ iff for all $\theta_{\mathbf{x}}$, whenever $I \models \langle \mathbf{B}\theta_{\mathbf{x}}, \eta \rangle$ and $(p(\mathbf{x})\theta)\eta_{\mathbf{B}\theta}$ is defined, then $I \models \langle p(\mathbf{x})\theta, \eta \mid \mathbf{x}\theta \rangle$,
  - $I \models p(\mathbf{x}) \leftarrow \mathbf{B}$ iff for all η, $I \models \langle p(\mathbf{x}) \leftarrow \mathbf{B}, \eta \rangle$,
  - $I \models A \leftarrow \mathbf{B}$ iff $I \models \mathbf{C}$, where $\mathbf{C}$ is the homogeneous form of $A \leftarrow \mathbf{B}$. □

Observe that as a consequence of Definition 2.8, if $I \models \langle \mathbf{A}, \eta \rangle$, then $Dom(\eta) \subseteq Var(\mathbf{A})$.

Definition 2.9 A Θ-interpretation I of P is a Θ-model of P iff all the clauses of P are true in I. □

The following lemmas will be useful to reason about the truth of conjunctions.

Lemma 2.10 (Splitting) *Let I be a Θ-interpretation, $\mathbf{A}, \mathbf{B}$ a conjunction, and η a substitution. Then $I \models \langle \mathbf{A}, \mathbf{B}, \eta \rangle$ iff $\exists \theta, \sigma.\ I \models \langle \mathbf{A}, \theta \rangle$, $I \models \langle \mathbf{B}\theta_{\mathbf{A}}, \sigma \rangle$ and $\eta_{\mathbf{A},\mathbf{B}} = \theta_{\mathbf{A}} \sigma_{\mathbf{B}\theta}$.*

Proof. By induction. The case in which $\mathbf{A}$ is a single atom coincides with Definition 2.8. Consider now the case $\mathbf{A} = A, \mathbf{A}'$. Then $I \models \langle A, \mathbf{A}', \mathbf{B}, \eta \rangle$ iff (by Definition 2.8) there exist ϕ, γ such that $I \models \langle A, \phi \rangle$, $I \models \langle (\mathbf{A}', \mathbf{B})\phi_A, \gamma \rangle$ and $\eta_{\mathbf{A},\mathbf{B}} = \phi_A \gamma_{(\mathbf{A}',\mathbf{B})\phi}$. By the induction hypothesis, this is true iff there exist ϕ, ψ, σ such that $I \models \langle A, \phi \rangle$, $I \models \langle \mathbf{A}'\phi_A, \psi \rangle$, $I \models \langle \mathbf{B}\theta_{\mathbf{A}}, \sigma \rangle$, $\gamma_{(\mathbf{A}',\mathbf{B})\phi} = \psi_{\mathbf{A}'\phi} \sigma_{\mathbf{B}\theta}$ and $\eta_{\mathbf{A},\mathbf{B}} = \phi_A \psi_{\mathbf{A}'\phi} \sigma_{\mathbf{B}\theta}$. By Definition 2.8, this is true iff there exists θ, σ such that $I \models \langle \mathbf{A}, \theta \rangle$, $I \models \langle \mathbf{B}\theta_{\mathbf{A}}, \sigma \rangle$ and $\eta_{\mathbf{A},\mathbf{B}} = \theta_{\mathbf{A}} \sigma_{\mathbf{B}\theta}$. □

Lemma 2.11 (Monotonicity) *Let I, J be Θ-interpretations, $\mathbf{A}$ a conjunction, and η a substitution. If $I \models \langle \mathbf{A}, \eta \rangle$ and $I \subseteq J$, then $J \models \langle \mathbf{A}, \eta \rangle$.*

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

Lemma 2.12 (Continuity) *Let I_i ($i \geq 0$) be Θ-interpretations such that $I_0 \subseteq I_1 \subseteq \ldots$. Then for every conjunction $\mathbf{A}$ and substitution η*

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

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

Note that the Continuity Lemma strengthens the Monotonicity Lemma.

We show now that the meaning of a clause can be defined directly, without the use of homogeneous form, albeit in a complicated way. It will be needed to establish the Soundness Theorem below.

Next definition splits the definition of truth of a clause in two parts, separating the interpretation of the antecedent and the consequent in two different Θ-interpretations. It will help to prove the characterization of truth of a clause in such a format that will be useful also to prove the characterization of the immediate-consequence operator (see Section 2.3).

Definition 2.13 (θ-pair) Let I, J be Θ-interpretations. We define

- $\langle I, J \rangle \models \langle p(\mathbf{x}) \leftarrow \mathbf{B}, \eta \rangle$, iff for all $\theta_{\mathbf{x}}$, whenever $I \models \langle \mathbf{B}\theta_{\mathbf{x}}, \eta \rangle$ and $(p(\mathbf{x})\theta)\eta_{\mathbf{B}\theta}$ is defined, then $J \models \langle p(\mathbf{x})\theta, \eta \mid p(\mathbf{x})\theta \rangle$,
- $\langle I, J \rangle \models p(\mathbf{x}) \leftarrow \mathbf{B}$ iff for all η, $\langle I, J \rangle \models \langle p(\mathbf{x}) \leftarrow \mathbf{B}, \eta \rangle$,
- $\langle I, J \rangle \models A \leftarrow \mathbf{B}$ iff $\langle I, J \rangle \models C$, where C is the homogeneous form of $A \leftarrow \mathbf{B}$. □

Given a clause C, a pair of Θ-interpretations $\langle I, J \rangle$ such that $\langle I, J \rangle \models C$ is called Θ-pair w.r.t. C. The correspondence with the notion of truth of a clause is immediate:

Lemma 2.14 *Let I be a Θ-interpretation and C be a clause. Then C is true in I iff $\langle I, I\rangle$ is a Θ-pair w.r.t. C.* $\square$

The property of being a Θ-pair is of course monotonic with respect to the second component.

Lemma 2.15 *Let I, J, K be Θ-interpretations such that $J \subseteq K$, and let C be a clause. Assume that $\langle I, J\rangle \models C$. Then $\langle I, K\rangle \models C$.* $\square$

Now we give a direct characterization of Θ-pair, without the use of homogeneous form.

Lemma 2.16 *Let I, J be Θ-interpretations and $A \leftarrow \mathbf{B}$ a clause of P. Then $\langle I, J\rangle \models A \leftarrow \mathbf{B}$ iff for every atom A' with no variables in common with $A \leftarrow \mathbf{B}$, if $\theta = mgu(A, A')$ exists, and there exists η such that $I \models \langle \mathbf{B}\theta, \eta\rangle$, and $(A'\theta)\eta_{\mathbf{B}\theta}$ is defined, then $J \models \langle A', (\theta\eta) \mid A'\rangle$.*

Proof. To prove this lemma we use the following property.

Lemma 2.17 *For every Θ-interpretation I we have $I \models \langle t_1 = u_1, \ldots, t_n = u_n, \theta\rangle$ iff $\theta = mgu(p(t_1, \ldots, t_n), p(u_1, \ldots, u_n))$.*

Proof. For $i \in [0, n]$ let θ_i be defined as follows:

$$\theta_0 = \epsilon,$$

and, for $i > 0$,

$$\theta_i = \theta_{i-1} mgu(t_i\theta_{i-1}, u_i\theta_{i-1}).$$

We first show by induction that $\theta = \theta_n$. For $n = 0$ we have $\theta = \epsilon = \theta_0$. For $n \geq 1$ we have, by definition,

$$\theta_n = \theta_{n-1} mgu(t_n\theta_{n-1}, u_n\theta_{n-1}).$$

By the induction hypothesis,

$$\theta_{n-1} = mgu(p(t_1, \ldots, t_{n-1}), p(u_1, \ldots, u_{n-1}))$$

holds. Then, by Palamidessi [Pal90], we obtain

$$\theta_n = mgu(p(t_1, \ldots, t_n), p(u_1, \ldots, u_n)).$$

Again by induction, we show now that $I \models \langle t_1 = u_1, \ldots, t_n = u_n, \eta\rangle$ iff $\eta = \theta_n$. For $n = 0$ we have $I \models \langle true, \eta\rangle$ iff $\eta = \epsilon$. For $n > 0$ we have, by induction hypotesis,

$$I \models \langle t_1 = u_1, \ldots, t_{n-1} = u_{n-1}, \sigma\rangle$$

iff $\sigma = \theta_{n-1}$. By definition of $[\![=]\!]$ and definition of truth, we know that

$$I \models \langle (t_n = u_n)\theta_{n-1}, \psi\rangle$$

iff $\psi = mgu(t_n\theta_{n-1}, u_n\theta_{n-1})$. By the Splitting Lemma 2.10, and since the conditions on definidness are satisfied (because we deal with relevant mgu's) we have

$$I \models \langle t_1 = u_1, \ldots, t_n = u_n, \phi\rangle$$

iff there exist σ, ψ such that $\sigma = \theta_{n-1}$, $\psi = mgu(t_n\theta_{n-1}, u_n\theta_{n-1})$ and $\phi = \sigma\psi$ holds. Finally, apply the definition of θ_n. $\square$

We now prove Lemma 2.16. For the sake of simplicity we do not mention explicitly the conditions under which the application and composition of substitutions is defined.

Let A be the atom $p(t_1, \ldots, t_n)$ and let $p(x_1, \ldots, x_n) \leftarrow x_1 = t_1, \ldots, x_n = t_n, \mathbf{B}$ be the homogeneous form of $A \leftarrow \mathbf{B}$. Then

$$\langle I, J \rangle \models A \leftarrow \mathbf{B}$$
iff
$$\langle I, J \rangle \models p(x_1, \ldots, x_n) \leftarrow x_1 = t_1, \ldots, x_n = t_n, \mathbf{B}$$
iff
$$\forall \theta_{\mathbf{x}}, \eta. \text{ if } I \models \langle (x_1 = t_1, \ldots, x_n = t_n, \mathbf{B})\theta_{\mathbf{x}}, \eta \rangle \text{ then } J \models \langle p(x_1, \ldots, x_n)\theta, \eta \mid (x_1, \ldots, x_n)\theta \rangle$$
iff (by Lemma 2.17)
$$\forall \theta_{\mathbf{x}}, \sigma, \psi \text{ if } \sigma = mgu(p(x_1\theta_{\mathbf{x}} \text{ and } I \models \langle \mathbf{B}\theta_{\mathbf{x}}\sigma, \psi \rangle \text{ then } J \models \langle p(x_1, \ldots, x_n)\theta, (\sigma\psi) \mid (x_1, \ldots, x_n)\theta \rangle$$
iff (since the x_i's do not occur in $\mathbf{B}$)
$$\forall A' = p(x_1, \ldots, x_n)\theta_{\mathbf{x}}, \forall \sigma, \psi \text{ if } \sigma = mgu(A, A') \text{ and } I \models \langle \mathbf{B}\sigma, \psi \rangle \text{ then } J \models \langle A', (\sigma\psi) \mid A' \rangle.$$

□

Finally we give a direct characterization of the truth of a clause.

Corollary 2.18 (Truth Characterization) *Let I be a Θ-interpretation and $A \leftarrow \mathbf{B}$ a clause of P. Then $I \models A \leftarrow \mathbf{B}$ iff for every atom A' with no variables in common with $A \leftarrow \mathbf{B}$, if $\theta = mgu(A, A')$ exists, and there exists η such that $I \models \langle \mathbf{B}\theta, \eta \rangle$, and $(A'\theta)\eta_{\mathbf{B}\theta}$ is defined, then $I \models \langle A', (\theta\eta) \mid A' \rangle$.*

Proof. Immediate by Lemma 2.14 and Lemma 2.16. □

The following lemma will be helpful in the proof of completeness. If C is a clause, by $Ren(C)$ we denote the set of clauses obtained by applying to C any renaming substitution.

Lemma 2.19 *Let I, J be Θ-interpretations and C a clause. Then $\langle I, J \rangle$ is a Θ-pair for C iff $\langle I, J \rangle$ is a Θ-pair for each clause in $Ren(C)$.* □

Proof. . The if part is straightforward, since the empty substitution is a renaming. We prove the only-if part by using the characterization of Θ-pair offered by Lemma 2.16. Assume C to be the clause $A \leftarrow \mathbf{B}$. Let I be a Θ-interpretation and ρ a renaming. Assume that $\langle I, J \rangle \models A \leftarrow \mathbf{B}$. We prove that $\langle I, J \rangle \models A\rho \leftarrow \mathbf{B}\rho$. Let A' be an atom with no variables in common with $A\rho \leftarrow \mathbf{B}\rho$. Assume that there exist $\theta = mgu(A\rho, A')$ and η such that $I \models \langle \mathbf{B}\rho\theta, \eta \rangle$ and $(A'\theta)\eta_{\mathbf{B}\rho\theta}$ is defined. We have that $A'\rho^{-1}$ has no variables in common with A (otherwise $A' = A'\rho^{-1}\rho$ would have variables in common with $A\rho$) and that $\rho\theta = mgu(A, A'\rho^{-1})$. Since $\langle I, J \rangle$ is a Θ-pair for C, by Lemma 2.16 we have that $J \models \langle A'\rho^{-1}, (\rho\theta\eta) \mid A'\rho^{-1} \rangle$. This implies that $J \models \langle A', (\theta\eta) \mid A' \rangle$ because J is closed under renaming. □

As a consequence, the truth of a clause C in a Θ-interpretation I is invariant under renaming, i.e. C is true in I iff all the clauses of $Ren(C)$ are true in I.

2.3 Θ-semantics and LD-resolution

In this section we prove the correctness of the LD-resolution with respect to the introduced semantics.

The next proposition and corollary show that the automatic renaming of variables done by the resolution procedure with ensures that the appropriate operations on the substitutions generated by the resolution steps are defined.

Proposition 2.20 *Let $G_0, \ldots, G_n$ be a prefix of an LD-derivation, and $\theta_1, \ldots, \theta_n$ the corresponding mgu's. Then for all $i \in [1, n]$*

(i) $(G_0\theta_1 \ldots \theta_{i-1})\theta_{i\,A}$ is defined,

(ii) $\eta_{G_0}\theta_{i\,A}$ is defined,

where A is the selected literal of G_{i-1} and $\eta = (\theta_1 \ldots \theta_{i-1}) \mid G_0$.

Proof.

1. If A is a built-in atom it follows from Definition 2.5. If A is an atom defined in the program it follows from the *standardization apart*, i.e. from the fact that the clause used for the i-th resolution step is renamed so to avoid clashes with the variables of $G_0\theta_1 \ldots \theta_{i-1}$. Therefore since θ_i is the relevant mgu between A and the head of the clause, it contains no variables in $G_0\theta_1 \ldots \theta_{i-1}$ others than the ones in A.

2. ¿From part (i) of this Proposition and part (ii) of Proposition 2.4. □

Corollary 2.21 *Consider a goal $\leftarrow \mathbf{A}, \mathbf{B}$ and an LD-refutation ξ of $\leftarrow \mathbf{A}, \mathbf{B}$. Let θ be the intermediate substitution computed in the prefix of ξ which is an LD-refutation of the subgoal $\leftarrow \mathbf{A}$, and let η be the substitution computed in the remaining suffix of ξ (which is an LD-refutation of the subgoal $\leftarrow \mathbf{B}\theta$). Then $\mathbf{B}\theta_{\mathbf{A}}$ and $\theta_{\mathbf{A}}\eta_{\mathbf{B}\theta}$ are both defined.*

Proof. By induction using Proposition 2.20. □

We are now in position to establish correctness of our semantics w.r.t. the LD-resolution.

Theorem 2.22 (Soundness) *Consider a program P. and a conjunction $\mathbf{A}$. Suppose there exists an LD-refutation of $P \cup \{\leftarrow \mathbf{A}\}$ with c.a.s. η. Then for all Θ-models I of P we have $I \models \langle \mathbf{A}, \eta \rangle$.*

To prove it we need the following lemma which shows the correctness of a single LD-derivation step.

Lemma 2.23 *Consider a program P and a goal $\leftarrow \mathbf{A}$. Suppose that $\leftarrow R$ is a resolvent of $\leftarrow \mathbf{A}$ and let θ be the associated mgu. Then for every Θ-model I of P and a c.a.s. η of $P \cup \{\leftarrow R\}$, if $I \models \langle R, \eta \rangle$ then $I \models \langle \mathbf{A}, (\theta\eta) \mid \mathbf{A} \rangle$.*

Proof. Fix a Θ-model I of P, and a substitution η such that $I \models \langle R, \eta \rangle$. $\mathbf{A}$ is of the form $A, \mathbf{B}$. We distinguish two cases.

Case 1 A is a built-in. By Definition 2.5 we have

$$I \models \langle A, \theta \rangle. \tag{1}$$

By assumption and by Proposition 2.20 $\mathbf{B}\theta_A$ is defined. But $R = \mathbf{B}\theta_A$, so

$$I \models \langle \mathbf{B}\theta_A, \eta \rangle. \tag{2}$$

By Corollary 2.21 $\theta_A\eta_{\mathbf{B}\theta}$ is defined. Furthermore, by Definition 2.3 $\theta_A\eta_{\mathbf{B}\theta} = (\theta\eta) \mid A, \mathbf{B}$, hence from (1) and (2), and by Definition 2.8 we conclude $I \models \langle \mathbf{A}, (\theta\eta) \mid \mathbf{A} \rangle$.

Case 2 A is defined in P. In this case there is a clause $A' \leftarrow \mathbf{B}'$ in $Ren(P)$ having no variables in common with $A, \mathbf{B}$ such that there exists $\sigma = mgu(A, A')$, $\theta = \sigma \mid A$, and $R = (\mathbf{B}', \mathbf{B})\sigma$. By the assumption and by the Splitting Lemma 2.10 there exist γ, ϕ such that

$$I \models \langle \mathbf{B}'\sigma, \gamma \rangle, \tag{3}$$

$$I \models \langle \mathbf{B}\sigma\gamma_{\mathbf{B}'\sigma}, \phi \rangle, \tag{4}$$

and $\eta_R = \gamma_{\mathbf{B}'\sigma}\phi_{\mathbf{B}\sigma\gamma}$. By Definition 2.3 we derive $\eta = (\gamma\phi) \mid R$. By Corollary 2.21 $(A\sigma)\gamma_{\mathbf{B}'\sigma}$ is defined. Moreover I is a model of P, so from (3) and Corollary 2.18 we derive

$$I \models \langle A, \psi \rangle, \tag{5}$$

where $\psi = (\sigma\gamma) \mid A$. Furthermore, $A' \leftarrow \mathbf{B}'$ has no variables in common with $A, \mathbf{B}$, and $Dom(\gamma) \subseteq Var(\mathbf{B}'\sigma)$, so $\mathbf{B}\sigma\gamma = \mathbf{B}\psi$ and $\mathbf{B}\psi_A$ is defined. Therefore we can rewrite (4) as

$$I \models \langle \mathbf{B}\psi_A, \phi \rangle. \tag{6}$$

Combining (6) and (5) we now conclude by Definitions 2.3 and 2.8

$$I \models \langle A, \mathbf{B}, (\psi\phi) \mid_{A,\mathbf{B}} \rangle.$$

Finally, observe that $(\psi\phi) \mid_{A,\mathbf{B}} = (\sigma\gamma\phi) \mid_{A,\mathbf{B}} = (\theta\eta) \mid \mathbf{A}$. □

Proof of Theorem 2.22. Consider an LD-refutation ξ for $P \cup \{\leftarrow \mathbf{A}\}$. We argue by induction on the length n of ξ.

$n = 1$. In this case $\mathbf{A} = true$, $\eta = \epsilon$, so by Definition 2.8, $I \models \langle \mathbf{A}, \eta \rangle$.

$n > 1$. Let $\leftarrow R$ be the direct descendant of $\leftarrow \mathbf{A}$ in ξ, and let θ be the associated substitution. Then $\leftarrow R$ has an LD-refutation ξ' of length $n - 1$ which is a suffix of ξ. Let σ be the c.a.s. in ξ'. By the induction hypothesis we have $I \models \langle R, \sigma \rangle$. Furthermore $\eta = (\theta\sigma) \mid \mathbf{A}$. Thus by Lemma 2.23 we conclude $I \models \langle \mathbf{A}, \eta \rangle$. □

In order to prove the converse of Theorem 2.22 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 2.24 *Let P be a program. Let $\mathcal{M}$ be a class of Θ-models of P. Then $M = \bigcap_{\mathcal{M}} I$ is a model of P.*

Proof. Let $p(\mathbf{x}) \leftarrow \mathbf{B}$ be a clause in $Hom(P)$ and Let $\mu, \theta_{\mathbf{x}}$ be such that $M \models \langle \mathbf{B}\theta_{\mathbf{x}}, \eta \rangle$ and $(p(\mathbf{x})\theta)\eta_{\mathbf{B}\theta}$ is defined. Fix $I \in \mathcal{M}$. By the Monotonicity Lemma 2.11 we have $I \models \langle \mathbf{B}\theta_{\mathbf{x}}, \eta \rangle$, so since I is a Θ-model, $I \models \langle p(\mathbf{x})\theta, \eta \mid \mathbf{x}\theta \rangle$. By Definition 2.8 and the definition of $\mathcal{M}$ we conclude $M \models \langle p(\mathbf{x})\theta, \eta \mid \mathbf{x}\theta \rangle$. □

Corollary 2.25 (Least Model) *Every program P has a least Θ-model, N_P.* □

This Θ-model is the intended representant of all Θ-models of P:

Corollary 2.26 *Let $\mathbf{A}$ be a conjunction and η be a substitution. Then $N_P \models \langle \mathbf{A}, \eta \rangle$ iff for all Θ-models I of P we have $I \models \langle \mathbf{A}, \eta \rangle$.*

Proof. By the Monotonicity Lemma 2.11. □

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 2.27 The immediate consequence operator T_P on the Θ-interpretations is defined as follows

$$\begin{array}{rcll} T_P(I) & = & \{\langle p(\mathbf{x})\theta, \eta \mid \mathbf{x}\theta \rangle \mid & \exists C \in P, \exists p(\mathbf{x}) \leftarrow \mathbf{B} \in Ren(Hom(C)), \\ & & & I \models \langle \mathbf{B}\theta_{\mathbf{x}}, \eta \rangle, \\ & & & (p(\mathbf{x})\theta)\eta_{\mathbf{B}\theta} \text{ is defined}\}. \end{array}$$

□

It is easy to see that $T_P(I)$ is closed under renaiming, hence it is a Θ-interpretation. We prove now a characterization of T_P in terms of Θ-pairs.

Proposition 2.28 *Given a clause C and a Θ-interpretation I, we have*

$$T_P(I) = min\{J \mid \forall C \in P.\ \langle I, J \rangle \models C\}.$$

Proof. By Definition 2.27, for each $C \in P$ we have $\langle I, T_P(I) \rangle \models Hom(C)$ and therefore $\langle I, T_P(I) \rangle \models C$. We show now that $T_P(I)$ is the minimum. Consider a Θ-interpretation J such that for each $C \in P$, $\langle I, J \rangle \models C$ holds. By definition $\langle I, J \rangle \models Hom(C)$, hence by Lemma 2.19 for each $C' \in Ren(Hom(C))$ we have $\langle I, J \rangle \models C'$. Therefore $T_P(I) \subseteq J$. □

Next, we characterize the Θ-models of P as the pre-fixed points of T_P.

Proposition 2.29 (Model Characterization) *I is a Θ-model of P iff $T_P(I) \subseteq I$.* □

Proof. By Lemma 2.14, I is a Θ-model of P iff, for each clause $C \in P$, $\langle I, I \rangle \models C$ holds. By Proposition 2.28 if $\langle I, I \rangle \models C$ then $T_P(I) \subseteq I$. Viceversa, if $T_P(I) \subseteq I$ then by Lemma 2.15, and since $\langle I, T_P(I) \rangle \models C$ holds by definition, we get $\langle I, I \rangle \models C$. □

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

Proposition 2.30 (Monotonicity) *T_P is monotonic, that is $I \subseteq J$ implies $T_P(I) \subseteq T_P(J)$.*

Proof. By the Monotonicity Lemma 2.11. □

Proposition 2.31 (Least Fixed Point) *T_P has a least fixed point $lfp(T_P)$ which is also its least pre-fixed point.*

Proof. By the Monotonicity Proposition 2.30 and Knaster-Tarski Theorem. $\square$

We can now derive the desired result.

Corollary 2.32 *$lfp(T_P) = N_P$.*

Proof. By the Least Fixed Point Proposition 2.31, Least Model Corollary 2.25 and Model Characterization Proposition 2.29. $\square$

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 2.30.

Proposition 2.33 (Continuity) *T_P is continuous, that is for every sequence I_i $(I \geq 0)$ of Θ-interpretations such that $I_0 \subseteq I_1 \subseteq \ldots$ we have*

$$T_P(\cup_{i=0}^{\infty} I_i) = \cup_{i=0}^{\infty} T_P(I_i).$$

Proof. By the Continuity Lemma 2.12. $\square$

Define now a sequence of Θ-interpretations by

$$\begin{aligned} T_P \uparrow 0 &= \emptyset, \\ T_P \uparrow (n+1) &= T_P(T_P \uparrow n), \\ T_P \uparrow \omega &= \cup_{i=0}^{\infty} T_P \uparrow i. \end{aligned}$$

Proposition 2.34 (Characterization) *$N_P = T_P \uparrow \omega$.*

Proof. By the Continuity Proposition 2.33 and the Knaster-Tarski Theorem $lfp(T_P) = T_P \uparrow \omega$, so the claim follows by Corollary 2.32. $\square$

The following proposition is the counterpart for T_P of Corollary 2.18.

Proposition 2.35 *The definition of T_P given in Definition 2.27 is equivalent to the following one*

$$\begin{array}{rcll} T_P(I) & = & \{\langle A, (\theta\eta) \mid A\rangle \mid & \exists A' \leftarrow \mathbf{B} \in Ren(P), \\ & & & Var(A' \leftarrow \mathbf{B}) \cap Var(A) = \emptyset, \\ & & & \theta = mgu(A, A'), \\ & & & I \models \langle \mathbf{B}\theta, \eta\rangle, \\ & & & (A\theta)\eta_{\mathbf{B}\theta} \textit{ is defined}\} \end{array}$$

Proof. Immediate from Lemma 2.16 and 2.28. $\square$

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

Theorem 2.36 (Completeness) *Consider a program P and a conjunction $\mathbf{A}$. Suppose that for all Θ-models I of P we have $I \models \langle \mathbf{A}, \eta \rangle$. 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 Characterizatin Proposition 2.34 $T_P \uparrow \omega \models \langle \mathbf{A}, \eta \rangle$. By the monotonicity of T_P we have $T_P \uparrow 0 \subseteq T_P \uparrow 1 \subseteq \ldots$, so by the Continuity Lemma 2.12 $T_P \uparrow k \models \langle \mathbf{A}, \eta \rangle$ for some $k > 0$.

Given a conjunction $\mathbf{A}$, denote by $l(\mathbf{A})$ its length. 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 2.8.

Suppose now $\mathbf{A} = A, \mathbf{B}$. There exist substitutions θ, σ such that

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

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

and $\eta_{A,\mathbf{B}} = \theta_A \sigma_{\mathbf{B}\theta}$.

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 2.5 and 2.8.

When A is defined in P we have $k > 0$. By Proposition 2.35 there exists a clause $A' \leftarrow \mathbf{B}'$ in $Ren(P)$, there exists $\psi = mgu(A, A')$, there exists ϕ such that $T_P \uparrow (k-1) \models \langle \mathbf{B}'\psi, \phi \rangle$, and $\theta_{A,\mathbf{B}'} = \psi_A \phi_{\mathbf{B}'\psi}$. Since $\langle k-1, l(\mathbf{B}') \rangle < \langle k, l(\mathbf{A}) \rangle$, by the induction hypothesis there exists an LD-refutation of $P \cup \{ \leftarrow \mathbf{B}'\psi \}$ with c.a.s. ϕ. Therefore there exists an LD-refutation of $P \cup \{ \leftarrow A \}$ with c.a.s. θ.

Since $\langle k, \mathbf{B} \rangle < \langle k, \mathbf{A} \rangle$, by the induction hypothesis also there exists an LD-refutation of $P \cup \{ \leftarrow \mathbf{B}\theta \}$ with c.a.s. σ.

This means that there exists an LD-refutation of $P \cup \{ \leftarrow \mathbf{A} \}$ with c.a.s. η. □

Corollary 2.37

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

Proof. By Definition 2.8 and Theorems 2.22 and 2.36. □

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

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

In this section we show that the Θ-semantics is a natural extension to Prolog programs of the S-semantics defined in Falaschi et al. [FLMP89] 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 is isomorphic 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. [FLMP89]).

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

or, equivalently,

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

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

Definition 2.38 Let I be a Θ-interpretation. I is called

- *upward-closed* if $\forall\langle A, \eta\rangle \in I$, $\forall\theta$ if $\sigma = mgu(A\theta, A\eta)$, then $\langle A\theta, \sigma \mid A\theta\rangle \in I$.
- *downward-closed* if $\forall A, \theta, \sigma.\ \langle A\theta, \sigma\rangle \in I \ \Rightarrow\ \exists\eta.\ \langle A, \eta\rangle \in I$ and $\sigma = mgu(A\theta, A\eta) \mid A\theta$.

□

Proposition 2.39 *Let P be a pure Prolog program. Then N_P is upward- and downward-closed.*

Proof. By using the characterization of N_P expressed by Corollary 2.37 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 2.40 *Let P be a pure prolog program.*

(i) If the goal $\leftarrow Q$ has an LD-refutation with c.a.s. η, then, for all θ if $\sigma = mgu(Q\theta, Q\eta)$, then the goal $\leftarrow Q\theta$ has an LD-refutation with c.a.s. σ' which is the restriction of σ to $Q\theta$.

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

Proof. This lemma can be proved by using the correctness and completeness theorems for the S-semantics (cf. Falaschi et al. [FLMP89]), but we prefer to give here an independent, purely operational proof.

We prove (i) by induction on the length of the refutation.

$n = 1$) Immediate since in this case $Q = true$ and $\eta = \epsilon$.

$n > 1$) Consider an LD-refutation ξ for $\leftarrow Q$ with c.a.s. η. Let θ, σ be substitutions such that $\sigma = mgu(Q\theta, Q\eta)$. Assume $Q = A_1, \ldots, A_n$. Let $\leftarrow R$ be the direct descendent of $\leftarrow Q$ in ξ, and let $A \leftarrow \mathbf{B}$ be the clause in $Ren(P)$ used to derive $\leftarrow R$ from $\leftarrow Q$. Assume, without loss of generality, that $A \leftarrow \mathbf{B}$ has no variables in common with Q or θ. Let $\mu = mgu(A, A_1)$. Then

$$R = (\mathbf{B}, A_2, \ldots, A_n)\mu$$

and $\leftarrow R$ has an LD-refutation ξ' with c.a.s. ϕ such that

$$Q\eta = Q\mu\phi. \qquad (8)$$

From $Q\theta\sigma = Q\eta\sigma$ and (8) it follows

$$Q\theta \leq Q\theta\sigma = Q\eta\sigma = Q\mu\phi\sigma \tag{9}$$

from which we get

$$A_1\theta \leq A_1\mu\phi\sigma. \tag{10}$$

Moreover we have

$$A \leq A\mu\phi\sigma = A_1\mu\phi\sigma. \tag{11}$$

By assumption A and $A_1\theta$ don't have variables in common, so from (10) and (11) it follows that they are unifiable. Consider $\gamma = mgu(A, A_1\theta)$. Since $\mu = mgu(A, A_1)$, and Q,θ and $A \leftarrow \mathbf{B}$ have no variables in common, we have $A = A\theta$, so $A\theta\gamma = A_1\theta\gamma$, thus $\mu \leq \theta\gamma$. Also $B = B\theta$, so $[\![\mu]\!] \leq B\theta\gamma = B\gamma$. Let ψ the most general substitution such that

$$Q\mu\psi = Q\theta\gamma \tag{12}$$

and

$$\mathbf{B}\mu\psi = \mathbf{B}\gamma. \tag{13}$$

Observe now that by (10), (11), and since the variables of Q, θ and $A \leftarrow \mathbf{B}$ are disjoint, we have

$$Q\theta\gamma \leq Q\mu\phi\sigma \tag{14}$$

and

$$\mathbf{B}\gamma \leq \mathbf{B}\mu\phi\sigma. \tag{15}$$

By (12) and (14) we have

$$Q\mu\psi \leq Q\mu\phi\sigma. \tag{16}$$

Analogously, by (13) and (15) we have

$$\mathbf{B}\mu\psi \leq \mathbf{B}\mu\phi\sigma. \tag{17}$$

Consider now $R\psi = (\mathbf{B}, A_2, \ldots, A_n)\mu\psi =$ (by (12) and (13)) $(\mathbf{B}, (A_2, \ldots, A_n)\theta)\gamma$. By (16) and (17) we have $R\psi \leq (\mathbf{B}, A_2, \ldots, A_n)\mu\phi\sigma = R\phi\sigma$. Therefore $R\psi$ and $R\phi$ are unifiable. Let τ be the most general substitution such that

$$R\psi\tau = R\phi\sigma. \tag{18}$$

By the induction hypothesis, the goal $\leftarrow R\psi$ has an LD-refutation with c.a.s. τ. Consequently, the goal $\leftarrow Q\theta$ has a LD-refutation with c.a.s. $\gamma\tau$ restricted to the variables of $Q\theta$. Finally, observe that by (12) and (18) we have

$$A_1\theta\gamma\tau = A_1\mu\psi\tau = A\mu\psi\tau = A\mu\phi\sigma = A_1\mu\phi\sigma$$

and

$$(A_2, \ldots, A_n)\theta\gamma\tau = (A_2, \ldots, A_n)\mu\psi\tau = (A_2, \ldots, A_n)\mu\phi\sigma$$

from which we devive $Q\theta\gamma\tau = Q\mu\phi\sigma$ and therefore, by (9), $Q\theta\gamma\tau = \theta\sigma$, i.e., on the variables of $Q\theta$ the substitutions $\gamma\tau$ and σ coincide.

The proof of (ii) is analogous. □

□

Note the analogy between Lemma 2.40(ii) and the Lifting Lemma. Actually, Lemma 2.40(ii) (which can be easily 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 only know that $Q\eta \leq Q\theta\sigma$). A property analogous to Lemma 2.40 (extended to partial c.a.s. and restricted to elementary atomic goals) has also been proved in **??**.

If P contains built-in relations, then N_P may fail to be upward-closed or downward-closed.

Example 2.41

- Consider the program

```
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 LD-refutation. Thus, due to Corollary 2.37, N_P is not upward-closed.

- Consider the program

```
p(X) ← nonvar(X).
```

  The goal $\leftarrow p(a)$ has an LD-refutation, but the goal $\leftarrow p(x)$ has no LD-refutation. Thus, due to Corollary 2.37, N_P is not downward-closed. □

We now show 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 an abstraction operator α from S-interpretations to Θ-interpretations, and an abstraction operator β from Θ-interpretations to S-interpretations such that for every program P

$$N_P = \alpha(S_P),$$

and

$$S_P = \beta(N_P).$$

Here α, β are abstraction operators in the sense that they do not depend upon P. (Thus 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 $\alpha(N_{P_1}) = \alpha(N_{P_2})$.)

In what follows we identify S-interpretations with Θ-interpretations whose very element is of the form $\langle A, \eta\rangle$ where A is an elementary atom.

Definition 2.42 The mappings *Up* and *Kernel* from Θ-interpretations to Θ-interpretations are defined as follows.

$$Up(I) = \{\langle A\theta, \sigma \mid A\theta\rangle \mid \exists\eta.\ \langle A, \eta\rangle \in I \text{ and } \sigma = mgu(A\theta, A\eta)\},$$

$$Kernel(I) = I \cap \{\langle A\eta\rangle \mid A \text{ is an elementary atom}\}.$$

□

Note that the definitions of *Up* and *Kernel* do not depend upon P. The following simple lemma relates the notions of upward- and downward-closedness with with these mappings.

Lemma 2.43 *(i) I is upward-closed iff $Up(I) \subseteq I$,*

(ii) I is downward-closed iff $I \subseteq Up(I)$,

(iii) if I is downward-closed then $Up(I) = Up(Kernel(I))$.

□

We now prove that *Up* restricted to S-interpretations and *Kernel* are the intended abstraction operators α and β which satisfy the property described above.

Proposition 2.44 *If P is a pure Prolog program, then*

(i) $N_P = Up(S_P)$,

(ii) $S_P = Kernel(N_P)$.

Proof. (ii) follows immediately from the definitions of *Kernel*, Corollary 2.37 and (7). Thus to prove (i) it sufficies to show that $N_P = Up(Kernel(N_P))$. But this is a direct consequence of Proposition 2.39. □

2.5 Full abstraction of the Θ-semantics

In the previous sections we saw that N_P coincides with the set of computational pairs $\{\langle A, \eta\rangle \mid$ there exists an LD-refutation of$P \;\cup\; \{\leftarrow A\}$with c.a.s. $\eta\}$ 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 from the truth value of the atoms.

We would like to argue that a declarative semantics should provide such an and-compositional interpretation of conjunctions of atoms sharing variables, because they might occur as bodies of program clauses. More generally, a declarative semantics should provide a compositional interpretation for conjunctions 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 clause can be transformed into a semantically and computationally equivalent clause whose body is in this format.

We 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 by encoding less information concerning the computational behavior of goals. We might for instance be interested in *observing* only the *non-ground success set* of a program P, defined by:

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

(which in fact corresponds to the least C-model when P is a pure Prolog program (cf. Falaschi et al. [FLMP91])). This notion can be viewed as the most interesting one, since (as we already argued in the introduction) the notion of *ground* success set doesn't make much sense in case of programs containing built-in relations. So the question is:

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

If we whish 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)–see Example 2.47 below. We have therefore to be content with a declarative semantics from which it is possible to derive $NGSS_P$, but which in general contains more information than $NGSS_P$ (information which is 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 essentially means that N_P is the simplest declarative semantics for Prolog programs.

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

Definition 2.45 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:

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

□

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. $NGSS_P$). More precisely we have the following obvious observation.

Lemma 2.46 $NGSS_P[\![Q]\!] = \{\theta\eta \mid \langle\theta, \eta\rangle \in N_P[\![Q]\!]\}$. □

On the other hand, it is not possible to retrieve the information encoded in N_P from the one encoded in $NGSS_P$. In fact, the mapping N_P is and-compositional, in the sense that $N_P[\![Q, R]\!]$ can be derived from $N_P[\![Q]\!]$ and $N_P[\![R]\!]$. Indeed, by the definition of LD-resolution

$$\langle\theta, \eta\rangle \in N_P[\![Q, R]\!] \quad \text{iff} \quad \begin{array}{l} \exists\sigma_{Q\theta}.\ \langle\theta, \sigma\rangle \in N_P[\![Q]\!], \\ \langle\theta\sigma, \eta\rangle \in N_P[\![R]\!], \text{ and} \\ R\sigma_{Q\theta} \text{ is defined.} \end{array}$$

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

Example 2.47 Consider the program

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

$$P \;=\; \{\;\; \begin{array}{l} p(x) \leftarrow x = a. \\ q(x) \leftarrow var(x), x = a. \quad \}. \end{array}$$

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 that N_P is the most abstract and-compositional semantics which is correct w.r.t. $NGSS_P$.

Theorem 2.48 *If $N_P[\![Q]\!] \neq N_P[\![R]\!]$ then there exists a conjunction $\mathbf{A}$ such that $NGSS_P[\![\mathbf{A}, Q]\!] \neq NGSS_P[\![\mathbf{A}, R]\!]$.*

Proof. To prove this result we use the built-in relation $\backslash ==$ defined by

$$[\![\backslash ==]\!] = \{\langle s \backslash == t\rangle \mid \; s \text{ and } t \text{ are different terms}\}.$$

Assume, without loss of generality, that there exist $\langle \theta, \eta\rangle \in N_P[\![Q]\!] \setminus N_P[\![R]\!]$. Let

$$\theta = \{x_1/t_1, \ldots, x_m/t_m\},$$

$$Var(Q\theta) = \{y_1, \ldots, y_n\}.$$

Define now:

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

$$\mathbf{A}_2 = var(y_1), \ldots, var(y_n),$$

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

where $\{\{k_1, l_1\}, \ldots, \{k_r, l_r\}\}$ are all possible two elements subsets of the set $\{1, \ldots, n\}$. Finally, define

$$\mathbf{A} = \mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3.$$

By the definition of $\mathbf{A}$ we derive immediately that $\theta\eta \in NGSS_P[\![\mathbf{A}, Q]\!]$. We now show that $\theta\eta \notin NGSS_P[\![\mathbf{A}, R]\!]$. Assume, by contradiction, that there exists $\sigma \in NGSS_P[\![\mathbf{A}, R]\!]$ such that

$$\sigma = \theta\eta. \tag{19}$$

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

$$\phi\psi = \sigma. \tag{20}$$

Thus for some γ, τ we have

$$\leftarrow \mathbf{A}_1\phi \text{ has an LD-refutation with c.a.s. } \gamma, \tag{21}$$

$$\leftarrow \mathbf{A}_2\phi\gamma \text{ has an LD-refutation with c.a.s. } \epsilon, \tag{22}$$

$$\leftarrow \mathbf{A}_3\phi\gamma \text{ has an LD-refutation with c.a.s. } \epsilon, \tag{23}$$

$$\leftarrow R\phi\gamma \text{ has an LD-refutation with c.a.s. } \tau, \tag{24}$$

and

$$\gamma\tau = \psi. \tag{25}$$

Since θ is an mgu for $\{x_1 = t_1, \ldots, x_n = t_n\}$, by (21) we have $\mathbf{A}_1\theta \leq \mathbf{A}_1\phi\gamma$. Moreover, by (22) and (23), 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)

$$\phi\gamma = \theta. \tag{26}$$

By (24), we have that $\leftarrow R\theta$ has an LD-refutation with c.a.s. τ. Furthermore, by (26), (25), (20), and (19),

$$R\theta\tau = R\phi\gamma\tau = R\phi\psi = R\sigma = R\theta\eta,$$

so, since both the domains of τ and η are restricted to $Q\theta$,

$$\tau = \eta. \tag{27}$$

Now (24) in view of (26) and (27) simply means that $\leftarrow R\theta$ has an LD-refutation with c.a.s. η which contradicts the assumption that $\langle\theta, \eta\rangle \notin N_P[\![R]\!]$. □

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 [?], Cavedon [?], Apt and Pedreschi [AP91]). The reason why restriction to ground atoms was sufficient can be best explained by the following observation.

Let $ground(P)$ denotes the set of all ground instances of clauses of the program P and $ground(C)$ denotes the set of all ground instances of the clause C.

Theorem 3.2 *Let P be a pure Prolog program and G a goal. Then all the LD-derivations of $P \cup \{G\}$ have finite length iff there exists an integer n such that all the LD-derivations of $ground(P) \cup \{G\}$ have length less than n.*

The following definition and lemmas will be useful in the proof.

Definition 3.3 Let d be a derivation of $P \cup \{G\}$. Then the set D_d associated to d is defined as follows:

let $d = G_1, \ldots, G_n, \ldots$ with input clauses $c_1, \ldots, c_n, \ldots$. Then $d' = G'_1, \ldots, G'_n, \ldots$ is in D_d iff

(i) $G'_1 = G_1$,

(ii) If G'_{i+1} is defined, then it is the resolvent of G'_i with input clause $c'_i \in ground(c_i)$.

□

Lemma 3.4 *Let d and D_d as in Definition 3.3. Then any derivation in D_d has length less or equal than the length of d.*

Proof. By construction of D_d. □

Lemma 3.5 *Let G_1 and G'_1 goals such that G'_1 is an instance of G_1. Let c be a clause such that $Var(c) \cap Var(G_1) = \emptyset$, and $c' \in ground(c)$. Let G_2 and G'_2 respectively the resolvent of G_1 and G'_1 with input clause c and c'. If both G_2 and G'_2 are different from the empty clause, then G'_2 is an instance of G_2.*

Proof. Let $G_1 = A, \mathbf{B}$, $G'_1 = A', \mathbf{B}'$ such that

$$G'_1 = G_1\eta \tag{28}$$

Let $c = A_1 \leftarrow B_1, \ldots, B_m$, $c' = A'_1 \leftarrow B'_1, \ldots, B'_m$ such that

$$c' = c\psi \tag{29}$$

Let $\theta_1 = mgu(A, A_1)$ and $\theta'_1 = mgu(A', A'_1)$. Then

$$G_2 = (B_1, \ldots, B_m, \mathbf{B})\theta_1$$

and

$$G'_2 = B'_1, \ldots, B'_m, \mathbf{B}'\theta'_1.$$

By $c' \in ground(c)$ it follows

$$A'\theta'_1 = A'_1 \tag{30}$$

By (28) it follows

$$A\eta = A' \tag{31}$$

By (30) and (31) it follows

$$A\eta\theta'_1 = A'_1 \tag{32}$$

By (29) it follows

$$A_1\psi = A'_1 \tag{33}$$

It is not restrictive to assume

$$Var(c) \cap Var(G'_1) = \emptyset \tag{34}$$

since by applying a variant of the Variant Lemma ([**?**]) we can reduce to this case.

Moreover by hypothesis

$$Var(c) \cap Var(G_1) = \emptyset \tag{35}$$

holds.

By (32), (33), (34), (35) it follows

$$A_1\eta\theta'_1\psi = A_1\psi(= A'_1) \tag{36}$$

and

$$A\eta\theta_1'\psi = A\eta\theta_1'(= A_1') \tag{37}$$

By $\theta_1 = mgu(A, A_1)$, (36), (37) it follows

$$\theta_1 \le \eta\theta_1'\psi \tag{38}$$

Then

$$\mathbf{B}\theta_1 \le \mathbf{B}\eta\theta_1' \tag{39}$$

and

$$(B_1, \ldots, B_m)\theta_1 \le (B_1, \ldots, B_m)\eta\theta_1'\psi \tag{40}$$

Now from (34) and (35) it follows

$$(B_1, \ldots, B_m)\eta\theta_1'\psi = (B_1, \ldots, B_m)\psi \tag{41}$$

and from (29) it follows

$$(B_1, \ldots, B_m)\psi = (B_1', \ldots, B_m') \tag{42}$$

Then from (39),(40),(41), (42) we obtain

$$G_2 \le G_2'$$

□

Now let Der_P denote the set of derivations of $P \cup \{G\}$ and $Der_{ground(P)}$ denote the set of derivations of $ground(P) \cup \{G\}$.

Lemma 3.6 *Let d be a derivation of $P \cup \{G\}$ and D_d as in Definition 3.3. Then*

$$D_d \subseteq Der_{ground(P)}.$$

Proof. By absurd suppose there exists $d' \in D_d$ such that $d' \neg \in Der_{ground(P)}$.

If d and d' are infinite then $d' \in Der_{ground(P)}$ holds by construction. Absurd.

If d is infinite and d' is finite then by construction d' is a finite failure derivation in $Der_{ground(P)}$. Absurd.

If d and d' are finite then we distinguish the two cases d refutation or d finite failure derivation.

If d is a refutation then c_n is a unit clause. This implies that d' is either a refutation or a finite failure derivation. In both cases $d' \in Der_{ground(P)}$. Absurd.

If d is a finite failure derivation then let

$$G_n = A, \mathbf{B}$$

and

$$G_n' = A', \mathbf{B}'$$

the last non-empty goals respectively of d and d' and let c_n and c_n' the respective input clauses.

By Lemma 3.5

$$G_1' = G_1\eta. \tag{43}$$

By construction

$$c'_n = c_n\beta(\in ground(c_n)). \tag{44}$$

Then

$$A' = A\eta \tag{45}$$

and if H and H' are the respective heads of c_n and c'_n then

$$H' = H\beta. \tag{46}$$

Moreover, since d is a finite failure derivation,

$$mgu(A, H) = \emptyset. \tag{47}$$

We show that $mgu(A', H') = \emptyset$. By absurd assume

$$mgu(A', H') = \vartheta. \tag{48}$$

Then by (45), (46), (48) it follows that

$$A\eta\vartheta = H\beta\vartheta,$$

which contradicts (47). Then d' is a finite failure derivation in $Der_{ground(P)}$. Absurd. □

The following lemma relates the set of derivations of $P \cup \{G\}$ with the set of derivations of $ground(P) \cup \{G\}$.

Lemma 3.7

$$\bigcup_{d \in Der_P} D_d = Der_{ground(P)}$$

Proof.

$\subseteq$ By Lemma 3.6.

$\supseteq$) We show that for any $d' \in Der_{ground(P)}$ there exists a derivation $d \in Der_P$ such that $d' \in D_d$. Let $d' = G'_1, \ldots, G'_n, \ldots$ with input clauses $c'_1, \ldots, c'_n, \ldots$. Take $d = G_1, \ldots, G_n, \ldots$ with input clauses $c_1, \ldots, c_n, \ldots$ such that

1. $G_1 = G'_1$
2. If G'_i is defined for some $i \geq 1$ then $c'_i \in ground(c_i)$.

To prove that $d' \in D_d$ we show that for any $i \geq 1$ if G'_i is defined then G_i is defined and different by the empty clause. We proceed by induction on i

$i = 1$ By 1).

$i > 1$ Suppose G_i defined and different by the empty clause. Then by Lemma 3.5 there exists η such that

$$G_i\eta = G'_i \tag{49}$$

Now let $c'_i = A'_1 \leftarrow B'_1, \ldots, B'_m$ and $c_i = A_1 \leftarrow B_1, \ldots, B_m$. Then by construction there exists ψ such that

$$c_i\psi = c_i'(\in ground(c_i)) \tag{50}$$

Let $G_i = A, \mathbf{B}$ and $G_i' = A', \mathbf{B}'$. Let $\theta_i' = mgu(A_1', A')$. Then from (49) and (50) it follows

$$A'\theta_i' = A_1' \tag{51}$$

$$A\eta = A' \tag{52}$$

$$A_1\psi = A_1' \tag{53}$$

¿From (51), (52) and (53) it follows

$$A\eta\theta_i' = A_1\psi$$

Thus there exists $mgu(A, A_1)$, that is G_{i+1} is defined and different from the empty clause.

□

Proof of Theorem 3.2

Assume that all derivations of $P \cup \{G\}$ are finite. Then by Lemma 3.4 and by Lemma 3.7 it follows that all derivations of $ground(P) \cup \{G\}$ are finite.

Vice versa assume that all derivations of $ground(P) \cup \{G\}$ are finite. By absurd suppose that there exists a derivation d in $P \cup \{G\}$ of infinite length. Let D_d as in Definition 3.3. By hypothesis the lengths of the derivations in D_d are boundedfrom above by n. Then there exists a maximum, say k, of their lengths.

We derive a contradiction by constructing a derivation $d' \in D_d$ of length greater than k. Let $d = G_1, \ldots, G_k, \ldots$ with input clauses $c_1, \ldots, c_k, \ldots$ and input substitutions $\theta_1, \ldots, \theta_k, \ldots$. Then d' is defined as in definition 3.3 from the $(k+2)-th$ goal and from the first till the $(k+1)-th$ goal as follows:

1. $G_1' = G_1$.
2. Let $G_1 = A, \mathbf{B}$, $c_1 = A_1 \leftarrow B_1, \ldots, B_m$. Then

   $$c_1' = c_1\theta_1 \ldots \theta_k\eta \in ground(c_1\theta_1 \ldots \theta_k)$$

   and

   G_2' is the resolvent of G_1' with input clause c_1'.

   Notice that from

   $$G_2 = (B_1, \ldots, B_m, \mathbf{B})\theta_1$$

   and

   $$G_2' = (B_1, \ldots, B_m, \mathbf{B})\theta_1 \ldots \theta_k\eta$$

   it follows that $G_2' \geq G_2\theta_2 \ldots \theta_k$.

3. Now let $i \in [2, k]$ and assume G'_i defined and different by the empty clause. Let $G_i = A, \mathbf{B}$, $G'_i = A', \mathbf{B}'$, such that

$$G'_i = G_i\theta_i \ldots \theta_k \eta \tag{54}$$

Let $c_i = A_1 \leftarrow B_1, \ldots, B_m$. Then

$$c'_i \in ground(c_i\theta_i \ldots \theta_k \eta) \tag{55}$$

and

G'_{i+1} is the resolvent of G'_i with input clause c'_i.

Notice that G'_{i+1} is defined and different from the empty clause. In fact $mgu(A', A'_1)$ does exists since from (54) it follows

$$A' = A\theta_i \ldots \theta_k \eta,$$

from $\theta_i = mgu(A, A_1)$ it follows

$$A\theta_i \ldots \theta_k \eta = A_1\theta_i \ldots \theta_k \eta$$

and from (55) it follows

$$A_1\theta_i \ldots \theta_k \eta \leq A'_1.$$

Then

$$G'_{i+1} = (B_1, \ldots, B_m)\theta_i \ldots \theta_k \eta\psi, \mathbf{B}'\psi$$

with $\psi = mgu(A', A'_1)$ and

$$G_{i+1} = (B_1, \ldots, B_m, \mathbf{B})\theta_i.$$

Thus $G'_{i+1} \geq G_{i+1}\theta_{i+1}, \ldots \theta_k$.

Then by construction $d' \in D_d$ and its length is greater than k. □

Thus to reason about termination of pure Prolog programs it is sufficient to study its ground version introducing the concept of boundness. This was done in Apt and Pedreschi [AP91] by combining the use of level mappings and models of the program.

The following simple example shows that Lemma 3.2 cannot be extended to Prolog programs with built-in's.

Example 3.8 Let $P = \{p(a) \leftarrow nonvar(x), p(a).\}$. Then the only LD-derivation of $P \cup \{\leftarrow p(a)\}$ is finite whereas the only LD-derivation of $ground(P) \cup \{\leftarrow p(a)\}$ is infinite. □

This naturally leads us to consider level mappings defined on non-ground atoms. Such level mappings were already considered in Bossi, Cocco and Fabris [?] and subsequently in Plümer [?] 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 3.9 A *level mapping* $|\ |$ is a function from atoms to a well-founded ordering such that $|A| = |B|$ if A and B are variants. □

The following auxiliary notion will be used below.

Definition 3.10 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 that are of a certain syntactic form.

Definition 3.11 A Prolog program is called *homogenous* if each of its non-unit clauses is in homogeneous form (cf. Definition 2.7). □

Our key concept in establishing termination is the following one.

Definition 3.12 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 3.13 (Soundness I) *Let P be a homogeneous Prolog program. Suppose P is acceptable. Then P is strongly terminating.*

The following notion will be useful in the proof.

Definition 3.14

(i) By the *length* of a goal we mean the number of its atoms. For a goal G we denote its length by $l(G)$.

(ii) 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 3.13. 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 2.22 we have $I \models \langle (B_1, \ldots, B_{i-1})\theta, \eta \rangle$. By the acceptability of P

$$|A\theta| > |B_i\theta\eta|. \tag{56}$$

By the choice of G_{j+1} the input clause used to obtain H_{j+1} from G_{j+1} is non-unit, so by assumption about P it is a homogeneous clause. Thus 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

$$|B_i\theta\eta| = |B_i\theta\eta\mu|. \tag{57}$$

Thus assuming $j > 1$, we have

$$|C_1| = |C_1\theta|, \tag{58}$$

(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

$$|C_1\theta| = |A\theta|. \tag{59}$$

¿From (56), (58), and (59) 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 I Theorem 3.13.

For a strongly terminating Prolog program P 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 3.15 (LD-tree) *Let P be a strongly terminating Prolog program. 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 [Apt90] 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 3.16 (Completeness I) *Let P be a homogeneous Prolog program. Suppose P is strongly terminating. Then P is acceptable.*

Proof. Put for an atom A

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

By Lemma 3.15 (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 3.15 (ii)

$$nodes_P(\leftarrow A) > nodes_P(\leftarrow B_1, \ldots, B_n). \tag{60}$$

This conclusion was reached under the assumption that $C\theta$ is disjoint with C but Lemma 3.15 (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 2.36 there exists an LD-derivation 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 3.15 (ii)

$$nodes_P(\leftarrow B_1, \ldots, B_n) \geq nodes_P(\leftarrow (B_i, \ldots, B_n)\eta) \tag{61}$$

and by Lemma 3.15 (iii)

$$nodes_P(\leftarrow (B_i, \ldots, B_n)\eta) \geq nodes_P(\leftarrow B_i\eta). \tag{62}$$

By (60), (61), and (62) 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.

Definition 3.17 Let P be a Prolog program. Let $x_1, x_2, \ldots$ be distinct variables not occurring in P. Transform each non-unit clause

$$p(t_1, \ldots, t_k) \leftarrow B_1, \ldots, B_n$$

of P into the clause

$$p(x_1, \ldots, x_k) \leftarrow x_1 = t_1, \ldots, x_k = t_k, B_1, \ldots, B_n.$$

Here $=$ is a 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. □

Theorem 3.18 *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.*

Proof. We use the following lemma.

*******introduce "one step" move and prove the lemma below for this transformation. *********

Lemma 3.19 *A goal G has a direct descendant H using a non-unit clause $C = p(t_1, \ldots, t_k) \leftarrow \mathbf{B}$ in P iff G has a derivation $G, G_1, \ldots, G_{k+1}$ in $Hom(P)$ such that $G_{k+1} = H$ and G_1 derives from G by using the clause $Hom(C) = p(x_1, \ldots, x_k) \leftarrow x_1 = t_1, \ldots, x_k = t_k, \mathbf{B}$*

Proof.

only if) Let G be the goal $\leftarrow p(u_1, \ldots, u_k), Q$. Let $\mu = mgu(p(u_1, \ldots, u_k), p(t_1, \ldots, t_k))$. Then $H = \leftarrow (\mathbf{B}, Q)\mu$. Let $\theta = \{x_1/u_1, \ldots, x_k/u_k\}$. Then $\theta = mgu(p(u_1, \ldots, u_k), p(x_1, \ldots, x_k))$. Define

$$\theta_0 = \theta$$

and, for $i \in [1, k]$

$$\theta_i = mgu((x_i = t_i)\theta_0 \ldots \theta_{i-1}),$$

and, for $i \in [1, k+1]$

$$G_i = \leftarrow (x_i = t_i, \ldots, x_k = t_k, \mathbf{B}, Q))\theta_0 \ldots \theta_{i-1}.$$

We have that $G, G_1, \ldots, G_{k+1}$ is a derivation for G using the clause $p(x_1, \ldots, x_k) \leftarrow x_1 = t_1, \ldots, x_k = t_k, \mathbf{B}$ in the first step. Furthermore, from [Pal90] we know that

$$\forall i \in [1, k].\ \theta_0 \ldots \theta_i = mgu(x_1 = u_1, \ldots, x_i = u_i, u_1 = t_1, \ldots, u_i = t_i).$$

Therefore we derive $\theta_0 \ldots \theta_k = \theta \cup \mu$ (note that $Dom(\theta) \cap Dom(\eta) = \emptyset$), hence

$$G_{k+1} = \leftarrow (\mathbf{B}, Q)(\theta \cup \mu).$$

Finally, since $\mathbf{B}, Q$ does not contain $x_1, \ldots, x_k$, we have

$$G_{k+1} = \leftarrow (\mathbf{B}, Q)\mu.$$

if) Analogous (just reverse the reasoning). □

Proof of Theorem 3.18

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 $Hom(P)$. To this end it suffices to note that by Lemma 3.19 G has a derivation $G, G_1, \ldots, G_n(, \ldots)$ in P iff G has a generalized derivation $G, G_1, \ldots, G_n(, \ldots)$ in $Hom(P)$. □

Corollary 3.20 *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 3.21 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. □

The following is a generalization of Definition 3.12 to arbitrary Prolog programs.

Definition 3.22 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 3.23 *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 3.24 *For homogeneous programs both definitions of acceptability coincide.* □

The following theorem is a generalization of the Soundness I Theorem 3.13.

Theorem 3.25 (Soundness II) *Let P be a Prolog program. Suppose P is acceptable. Then P is strongly terminating.*

Proof. The proof is completely analogous to that of the Soundness Theorem 3.13. The only difference is that instead of (57) 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 (58). However, this weaker conclusion is still sufficient to yield the same contradiction as in the proof of Theorem 3.13. □

Ideally, we would like to prove the converse of the Soundness II Theorem 3.25, that is that strongly terminating Prolog programs are acceptable. Unfortunately this is not the case.

Theorem 3.26 *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 3.27 *Let P be a Prolog program. Then P strongly terminates iff $Hom(P)$ is acceptable.*

Proof. By the Soundness I Theorem 3.13 and Completeness I Theorem 3.16 applied to $Hom(P)$ and Corollary 3.20. □

4 Applications

In this section 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 strongly terminating relation 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:

```
(l1)  list([]).
(l2)  list([X | Xs]) ←
        nonvar(Xs), list(Xs).
```

To prove that list is strongly terminating 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 of (l_2). C is of the form $A \leftarrow B_1, B_2$. Suppose $\Theta_P \models \langle B_1, \eta\rangle$. Then $\eta = \epsilon$ and $B_1 = nonvar(xs)$ with $xs \notin Var$ by the definition of Θ_P. We now prove three claims which obviously imply that list satisfies the appropriate requirement.

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

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

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

Proof. Note that $|list([x|xs])| = |[x|xs]| > |xs| = |list(xs)|$. □

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). Then any idempotent and relevant mgu θ of $B_2\eta$ and $list([x'|xs'])$ does not bind the variables of $B_2\eta$, so $B_2\eta\theta = B_2\eta$ and consequently $|B_2\eta\theta| = |B_2\eta|$. □
□

It is useful to note that the pure Prolog conterpart of list obtained by dropping the atom $nonvar(xs)$ may diverge.

Modularity

In the proof of Theorem 3.13 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 built-in.

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 consider the program `append` formed by clauses (a_1) and (a_2). 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` and the level mapping always equal to 0. Now to show that `append` strongly terminates we prove that it is acceptable. We take the following level mapping:

$$|append(x, y, z)| = |x|,$$

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

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

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

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

Proof. Note that $|[\,]| + |ys| = |ys|$ and that $|xs| + |ys| = |zs|$ implies $|[x|xs]| + |ys| = |[x|zs]|$. This implies that I 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 We say that a relation p is *defined* in a Prolog program P if p occurs in a head of a clause from P. □

Definition 4.6 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 4.7 (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 $P_1 \cup P_2$ *is strongly terminating.*

Proof. P_2 extends P_1. Thus $P_1 \cup P_2$ is strongly terminating iff P_1 is strongly terminating and P_2 is strongly terminating 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 I Theorem 3.13 P_1 is strongly terminating. 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 2.26 P_2 is acceptable w.r.t. $N_{P_1 \cup P_2}$ and the level mapping $| \ |$. Moreover, by Corollary 2.37 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 I Theorem 3.13 P_2 is strongly terminating 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 now the program `UNIFY` (for unification without occurs check) from Sterling and Shapiro [SS86][page 150]. In this program several built-in's, namely *var*, *nonvar*, $=$, *constant*, *compound*, *functor*, $>$, $:=$, *arg*, $-$ are used. The meaning of some of them was already given in section 2. The meaning of the remaining ones is defined as follows.

$$\begin{array}{lcl}
[\![constant]\!] & = & \{\langle constant(a), \epsilon\rangle \mid a \text{ is a constant}\}, \\[2ex]
[\![compound]\!] & = & \{\langle compound(s), \epsilon\rangle \mid s \text{ is a compound term}\}, \\[2ex]
[\![functor]\!] & = & \{\langle functor(t, f, n), \eta\rangle \mid Dom\eta = \{f, n\},\ n\eta \text{ is a natural number and} \\
 & & \qquad\qquad t = (f\eta)(t_1, \ldots, t_{n\eta}) \text{ for some } t_1, \ldots, t_{n\eta}\}, \\[2ex]
[\![:=]\!] & = & \{\langle x := s, \{x/t\}\rangle \mid s \text{ is an arithmetic expression whose value is } t\}, \\[2ex]
[\![arg]\!] & = & \{\langle arg(n, s, t), \eta\rangle \mid t\eta \text{ is the } n\eta - th \text{ argument of } s\eta\}.
\end{array}$$

Additionally, the function "$-$" (minus) is used in 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 relations $>$ and $-$, written in infix notation, are defined. This domain can be incorporated into the Herbrand universe of `UNIFY` by adding to the language of `UNIFY` the constant 0 and the successor function s (for example by adding to `UNIFY` the clause `s(0) > 0` $\leftarrow$.).

In Pieramico [**?**] 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 [AP91]).

We prove here a stronger statement, namely that `UNIFY` is strongly terminating by showing that it is acceptable in the sense of Definition 3.12.

For the subsequent analysis it is important to understand how this program operates. Intuitively, the goal $\leftarrow$ `unify(s,t)` yelds 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 $f(t_1, \ldots, t_n)$, 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 defined below.

We use here the following notation. Let s be a term. Then

- s_i denotes the i-th component of s if it exists and otherwise an arbitrary constant,
- $nodes(s)$ denotes the number of nodes of s,
- $a(s)$ denotes the arity of s.

We now define the following Θ-interpretation of UNIFY.

$$
\begin{array}{rcl}
I & = & \{\langle unify(s,t),\theta\rangle \mid Var(s\theta) = Var(t\theta),\ nodes(s\theta) = nodes(t\theta), \\
& & Var(s\theta) \cup Var(t\theta) \subseteq Var(s) \cup Var(t), \\
& & Var(s\theta) \cup Var(t\theta) = Var(s) \cup Var(t) \Rightarrow (nodes(s) + nodes(t) = \\
& & nodes(s\theta) + nodes(t\theta))\} \\
& \cup & \{\langle term-unify(s,t),\theta\rangle \mid Var(s\theta) = Var(t\theta), nodes(s\theta) = nodes(t\theta), \\
& & Var(s\theta) \cup Var(t\theta) \subseteq Var(s) \cup Var(t), \\
& & Var(s\theta) \cup Var(t\theta) = Var(s) \cup Var(t) \Rightarrow (nodes(s) + nodes(t) = \\
& & nodes(s\theta) + nodes(t\theta))\} \\
& \cup & \{\langle unify-args(n,s,t),\theta\rangle \mid\ n \text{ is a natural number,} \\
& & Var(s_i\theta) = Var(t_i\theta) \text{ and } nodes(s_i\theta) = nodes(t_i\theta) \text{ for } i \in [1,n], \\
& & Var(s\theta) \cup Var(t\theta) \subseteq Var(s) \cup Var(t), \\
& & Var(s\theta) \cup Var(t\theta) = Var(s) \cup Var(t) \Rightarrow (nodes(s) + nodes(t) = \\
& & nodes(s\theta) + nodes(t\theta))\} \\
& \cup & \{\langle unify-arg(n,s,t),\theta\rangle \mid\ n \text{ is a natural number,} \\
& & Var(s_n\theta) = Var(t_n\theta),\ nodes(s_n\theta) = nodes(t_n\theta), \\
& & Var(s\theta) \cup Var(t\theta) \subseteq Var(s) \cup Var(t), \\
& & Var(s\theta) \cup Var(t\theta) = Var(s) \cup Var(t) \Rightarrow nodes(s) + nodes(t) = \\
& & nodes(s\theta) + nodes(t\theta)\}.
\end{array}
$$

Lemma 4.8 *I is a Θ-model of* UNIFY.

Proof.

Notice that if $\langle s = t, \theta\rangle \in [\![=]\!]$ then θ is a mgu of s and t. It follows that

$$Var(s\theta) \cup Var(t\theta) \subseteq Var(s) \cup Var(t)$$

and

$$Var(s\theta) \cup Var(t\theta) = Var(s) \cup Var(t) \Rightarrow nodes(s) + nodes(t) = nodes(s\theta) + nodes(t\theta)$$

Thus, from the semantics of =, from the semantics of the other built-in relations previously specified and from the fact that $[\![unify]\!] = [\![term - unify]\!]$ it follows that I is a Θ-model of (u_1) - (u_5).

Notice that

$$Var(s_n) = Var(t_n),\ nodes(s_n) = nodes(t_n)$$

and

$$Var(s_i) = Var(t_i),\ nodes(s_i) = nodes(t_i),\ i \in [1, n-1]$$

implies

$$Var(s_i) = Var(t_i),\ nodes(s_i) = nodes(t_i),\ i \in [1, n].$$

This implies that I is a Θ-model of (uar_1).

It is easy to check that I is also a Θ-model of the other clauses in UNIFY. □

Next, we 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

$$n_1 < m_1$$

$$\text{or } n_1 = m_1 \wedge n_2 < m_2$$

$$\text{or } n_1 = m_1 \wedge n_2 = m_2 \wedge n_3 < m_3.$$

We put

$$|unify(s,t)| = \langle\ card(Var(s) \cup Var(t)),\ nodes(s) + nodes(t),\ 1\ \rangle.$$

$$|term - unify(s,t)| = \langle\ card(Var(s) \cup Var(t)),\ nodes(s) + nodes(t),\ 0\ \rangle.$$

$$|unify - args(n,s,t)| = \langle\ card(Var(s) \cup Var(t)),\ f(n,s,t),\ 3\ \rangle$$

$$|unify - arg(n,s,t)| = \langle\ card(Var(s) \cup Var(t)),\ nodes(s_n) + nodes(t_n),\ 2\ \rangle.$$

$$|A| = \langle\ 0,\ 0,\ 0\ \rangle \text{ if } A \text{ built-in.}$$

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

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

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 3.12, 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 of (u_5). C is of the form

$$A \leftarrow B_1, B_2, B_3, B_4, B_5.$$

We now 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
$$|A| > \langle 0, 0, 0\rangle = |B_i| \; for \; i = 1, \ldots, 4.$$

□

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

Proof. By assumption $s\eta = s$, $t\eta = t$. So

$$\begin{aligned}&|unify(s,t)| = \langle\ card(Var(s) \cup Var(t)),\ nodes(s) + nodes(t),\ 1\ \rangle\\ &> \langle\ card(Var(s) \cup Var(t)),\ nodes(s) + nodes(t),\ 0\ \rangle = |term - unify(s,t)|.\end{aligned}$$

□

Consider an instance C of (tu_1). C is of the form

$$A \leftarrow B_1, B_2, B_3.$$

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

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

Proof. Note that
$$|A| > 0 = |B_i| \text{ for } i = 1, 2.$$

□

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
$$|term - unify(s,t)| > |unify - args(n, s, t)|.$$

□

Consider an instance C of (uar_1). C is of the form

$$A \leftarrow B_1, B_2, B_3, B_4.$$

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

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

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

□

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

Proof. By assumption $s\eta = s$, $t\eta = t$, $n > 0$. 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 assumption $n > 0$, $a(s) \geq n > 0$, $n1\eta = n - 1$, $Var(s\eta) \cup Var(t\eta) \subseteq Var(s) \cup Var(t)$.

We distinguish two cases.

$Var(s\eta) \cup Var(t\eta) \subset Var(s) \cup Var(t)$.

Then

$$card(Var(s) \cup Var(t)) > card(Var(s\eta) \cup Var(t\eta))$$

$Var(s\eta) \cup Var(t\eta) = Var(s) \cup Var(t)$.

Then by assumption $nodes(s\eta) + nodes(t\eta) = nodes(s) + nodes(t)$. Thus in this case

$$f(n, s, t) > f(n - 1, s\eta, t\eta).$$

In both cases it follows

$$|unify - args(n, s, t)| > |unify - args(n - 1, s\eta, t\eta)|.$$

$\square$

Consider an instance C of (ua_1). C is of the form

$$A \leftarrow B_1, B_2, B_3.$$

We now 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 $s\eta = s$, $t\eta = t$, $n > 0$, $a(s) \geq n > 0$, $args\eta = s_n$, $argt\eta = t_n$.

So

$$|unify - arg(n, s, t)| > |unify(s_n, t_n)|.$$

$\square$

Consider now the program `UNIFYoc` for the unification with occur check (see Sterling and Shapiro [SS86][page 152]). It is obtained from `UNIFY` by first introducing the atom `not-occurs-in(X,Y)` before the last atom in the bodies of the clauses (u_2) and (u_3). We call the resulting program `UNIFY'`. Then the following program `not-occur` defining the relation `not-occurs-in(X,Y)` is added:

```
(noc1) not-occurs-in(X,Y) ←
         var(X), 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 4.7 and the Soundness I Theorem **??** 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 $|not-occurs(s,t)| = 0$.

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

$$|not-occurs-in(x,y)| = \ nodes(y)+1.$$

$$\begin{array}{ll} |not-occurs-in(n,x,y)| = & nodes(y) - \sum_{i=n+1}^{a(y)} nodes(i-th\ arg\ y) \ \text{ if } n > 0 \\ |not-occurs-in(0,x,y)| = & 0. \end{array}$$

$$|A| = \ 0 \text{ if } A \text{ is built-in.}$$

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

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

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 3.12 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 of (noc_3). C is of the form

$$A \leftarrow B_1, B_2, B_3, B_4.$$

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

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

Proof. Note that
$$|A| > 0 = |B_i| \; for \; i = 1, \ldots, 3.$$
□

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

Proof. By assumption $s\eta = s$, $t\eta = t$ and $n = a(t)$. So
$$|not - occurs - in(s,t)| = nodes(t) + 1 > nodes(t) = |not - occurs - in(n, s, t)|.$$
□

Consider now an instance C of (no_1). C is of the form
$$A \leftarrow B_1, B_2, B_3, B_4, B_5.$$

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

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

Proof. Note 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 assumption $s\eta = s$, $t\eta = t$, $argt\eta = t_n$ and $1 \leq n \leq a(t)$. So
$$|not - occurs - in(n, s, t)| = nodes(t) - \sum_{i=n+1}^{a(t)} nodes(t_i)$$
$$> \quad nodes(t_n) + 1 = |not - occurs - in(s, t_n)|.$$
□

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

Proof. By assumption $s\eta = s$, $t\eta = t$, $argt\eta = t_n$, $n1\eta = n - 1 \geq 0$ and $1 \leq n \leq a(t)$. So
$$|not - occurs - in(n, s, t)| = nodes(y) - \sum_{i=n+1}^{a(t)} nodes(t_i)$$
$$> \quad nodes(t) - \sum_{i=n-1}^{a(t)} nodes(t_i) = |not - occurs - in(n - 1, s, t)|.$$
□

To prove that `UNIFYoc` strongly terminates 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 `UNIFYoc` *is acceptable w.r.t.* $|\;|$ *and* I.

Proof. Notice that if C is an instance of (u_2) or (u_3), then C is of the form $A \leftarrow B_1, \ldots, B_4$. If $I \models \langle B_1, B_2, B_3, \eta \rangle$ then $s\eta = s$ and $t\eta = t$, i.e. `not-occurs-in`(s, t) doesn't modify s and t.

It follows that the proof that `UNIFYoc` is acceptable w.r.t. $|\;|$ and I is analogous to the one for `UNIFY`. □

References

[AP91] J.-M. Andreoli and R. Pareschi. Linear Objects: Logical processes with built-in inheritance. *New Generation Computing*, 9:445–473, 1991. Also in the Proc. of the Seventh International Conference on Logic programming, 1990, Jerusalem. The MIT Press.

[Apt90] K. R. Apt. Logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 493–574. Elsevier, 1990. Vol. B.

[Bör89] E. Börger. A logical operational semantics of full Prolog, Part III: Built-in predicates for files, terms, arithmetic and input-output. In Y.Moschovakis, editor, *Proceedings Workshop on Logic from Computer Science*. Springer MSRI Publications, 1989.

[Cla79] K.L. Clark. Predicate logic as a computational formalism. Res. Report DOC 79/59, Imperial College, Dept. of Computing, London, 1979.

[FD87] G. Ferrand and P. Deransart. An operational formal definition of Prolog. In *Proceedings of the 4th Symposium on Logic Programming*, pages 162–172. Computer Society Press, 1987.

[FLMP89] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. Declarative modeling of the operational behaviour of logic languages. *Theoretical Computer Science*, 69(3):289–318, 1989.

[FLMP91] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. A model-theoretic reconstruction of the operational semantics of logic programs. *Information and Computation*, 1991. To appear. Also available as technical report TR 32/89 at the Department of Computer Science, Pisa.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, second edition, 1987.

[Pal90] C. Palamidessi. Algebraic properties of idempotent substitutions. In M.S. Paterson, editor, *Proc. of the 17th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 443 of *Lecture Notes in Computer Science*, pages 386–399, Warwick, England, 1990. Springer-Verlag. Full version available as Technical Report TR-33/89, Dipartimento di Informatica, Università di Pisa.

[SS86] L. Sterling and E. Shapiro. *The Art of Prolog: Advanced Programming Techniques*. Series in Logic Programming. The MIT Press, 1986.

[vEK76] M.H. van Emden and R.A. Kowalski. The semantics of predicate logic as a programming language. *Journal of ACM*, 23(4):733–742, 1976.