ELSEVIER Science of Computer Programming 32 (1998) 145–176

Science of Computer Programming

Towards a logical semantics for pure Prolog[1]

Roberto Barbuti[a], Nicoletta De Francesco[b], Paolo Mancarella[a,*], Antonella Santone[b]

[a] *Dipartimento di Informatica, Università di Pisa, I-56125 Pisa, Italy*
[b] *Dipartimento di Ingegneria dell'Informazione, Università di Pisa, I-56125 Pisa, Italy*

Abstract

The coincidence of the declarative and procedural interpretations of logic programs does not apply to Prolog programs, due to the depth-first left-to-right evaluation strategy of Prolog interpreters. We propose a semantics for Prolog programs based on a four-valued logic. The semantics is based on a new concept of completion analogous to Clark's and it enjoys the nice properties of the declarative semantics of logic programming: existence of the least Herbrand model, equivalence of the model-theoretic and operational semantics.

Keywords: Logic programming; Prolog; Semantics; Multi-valued logics

1. Introduction

One of the most attractive features of the logic programming paradigm is the equivalence between its declarative and procedural reading. When looked at as a first-order theory, a collection of Horn clauses can be characterized by its least Herbrand model; when looked at as a set of procedure definitions, a collection of Horn clauses can be characterized by its success set, which coincides with the least Herbrand model. Unfortunately, it is well known that this equivalence is lost when moving from logic programming to Prolog programming, the reason being that Prolog interpreters use, for efficiency reasons, a depth-first left-to-right computation strategy. As a consequence, the declarative semantics of logic programming cannot be adopted as the abstract logical semantics for Prolog programs. For this reason, usually the semantics of Prolog is defined using non-logical frameworks [4–6, 8–10, 16].

When dealing with computational issues, one has to abandon classical two-valued logic and has to move to multiple-valued logic. A first attempt is to adopt a three-valued logic where the third truth value (*undefined*) is introduced to model non-terminating

* Corresponding author. E-mail: paolo@di.unipi.it.

[1] Work partially supported by the EEC Project KIT011-LPKRR.

0167-6423/98/$19.00
PII: S0167-6423(97)00033-6

computations (see, e.g. [1, 2, 14, 17, 20]). However, these three-valued based semantics do not allow to model the computational behaviour of Prolog.

In this paper we propose a logical semantics for pure Prolog (without extra-logical features and negation) based on a four-valued logic. Roughly speaking, the fourth truth value is intended to model a computation in which a success is "followed by" a non-termination as in the computation of the goal $\leftarrow p$ with respect to the Prolog program $\{p,\ p \leftarrow p\}$.

The semantics is based on the notion of *sequential completion* of a Prolog program, which differs from Clark's completion in that the standard connectives $\wedge$ and $\vee$ are interpreted as *sequential conjunction* and *sequential disjunction*. These connectives are suitably defined on our four-valued logic and their logical meaning reflect the computational behaviour of Prolog: $\wedge$ models the left-to-right computation rule of Prolog, while $\vee$ models the search strategy, i.e. the sequential use of the clauses in a program.

The semantics we propose for Prolog enjoys the nice properties of the declarative semantics of logic programming (existence of the least Herbrand model, equivalence of the model-theoretic and operational semantics).

It is worth mentioning that our semantics is *truly* logical for propositional Prolog, whereas it looses part of its logical flavour when moving to the non-propositional case. This is due to the evaluation of existentially quantified goals which is based on a suitable ordering on ground instances of goals, which is obtained by exploiting the fixpoint approach of [9]. However, the truth value of an existentially quantified goal is still obtained logically as the sequential disjunction of its ground instances, according to the ordering just mentioned.

For the sake of clarity, we first explore our approach for propositional Prolog, and then we extend it to full, pure Prolog.

2. Preliminaries

In this section, we will provide the basic notions of multiple-valued logic and logic programming.

2.1. Multiple valued logics

There are different ways to present multiple-valued logics (see, e.g. [13, 23, 21]): in this paper we basically follow the approach of [21] based on *valuation systems*.

We consider a predicate language $\mathscr{L} = (\mathscr{P}, \mathscr{F}, \mathscr{V}, \mathscr{O}, \mathscr{Q})$, where $\mathscr{P}$ is a set of predicate symbols, $\mathscr{F}$ is a set of function symbols, $\mathscr{V}$ is a set of variables, $\mathscr{O}$ is a set of operators (connectives), and $\mathscr{Q}$ is a set of quantifiers. With each function symbol $f \in \mathscr{F}$, predicate symbol $p \in \mathscr{P}$, and operator o in $\mathscr{O}$ is associated a unique natural number called its *arity*. We assume that the language contains at least one constant symbol (constant symbols are function symbols of arity 0). The ground term algebra over $\mathscr{F}$ is denoted by $Tm(\mathscr{F})$. The non-ground term algebra over $\mathscr{F}$ and $\mathscr{V}$ is denoted by $Tm(\mathscr{F}, \mathscr{V})$.

The set of atoms constructed from predicate symbols in $\mathscr{P}$ and terms from $Tm(\mathscr{F},\mathscr{V})$ is denoted $Am(\mathscr{P},\mathscr{F},\mathscr{V})$ or Am for short.

A *formula* is (i) an atom, or (ii) $o(\phi_1,\ldots,\phi_n)$, where each ϕ_i is a formula and o has arity n, or (iii) $q\ x.\phi$, where q is a quantifier in $\mathscr{Q}$, x is a variable in $\mathscr{V}$ and ϕ is a formula.

Given a formula ϕ, the notion of *free* and *bound* occurrence of a variable is defined as usual. We denote by $\phi[x:=t]$ the formula obtained from ϕ by replacing the free occurrences of the variable x by the term t. As usual, bound variables in ϕ may be renamed in order to avoid conflict with the variables in t.

A *valuation system* $\mathscr{V}$ for a predicate language $\mathscr{L}=(\mathscr{P},\mathscr{F},\mathscr{V},\mathscr{O},\mathscr{Q})$ is a tuple $\langle\mathscr{T},\mathscr{D},\mathscr{R},\mathscr{G}\rangle$ where

- $\mathscr{T}$ is the set of truth values, with at least two elements;
- $\mathscr{D}$ is the set of *designated* truth values, a non-empty proper subset of $\mathscr{T}$;
- $\mathscr{R}$ is a set of functions. Each function $r_o\in\mathscr{R}$ corresponds to one operator in $\mathscr{O}$, and it is such that $r_o:\mathscr{T}^{n_o}\to\mathscr{T}$ (where n_o is the arity of the operator o). We say that r_o interprets o;
- $\mathscr{G}$ is a set of functions from $\wp(\mathscr{T})$ to $\mathscr{T}$. Each function g_q corresponds to one quantifier q and it maps possibly infinite subsets of $\mathscr{T}$ onto an element of $\mathscr{T}$.

A *basic assignment* α relative to a valuation system $\langle\mathscr{T},\mathscr{D},\mathscr{R},\mathscr{G}\rangle$ for a predicate language $\mathscr{L}=(\mathscr{P},\mathscr{F},\mathscr{V},\mathscr{O},\mathscr{Q})$ is a pair $\alpha=\langle\rho,I\rangle$, where I is a non-empty set of *individuals* (also called *universe of discourse*) and ρ is a mapping such that

- $\rho(t)\in I$ for each $t\in Tm(\mathscr{F},\mathscr{V})$;
- $\rho(p):I^n\to\mathscr{T}$ for each predicate $p\in\mathscr{P}$ with arity n.

Each basic assignment α induces an *interpretation* (or *valuation*) v_α of a sentence in the language, inductively defined as follows:

- $v_\alpha(p(t_1,\ldots,t_n))=\rho(p)(\rho(t_1),\ldots,\rho(t_n))$, where $p\in\mathscr{P}$ with arity n, and $t_1,\ldots,t_n\in Tm(\mathscr{F},\mathscr{V})$;
- $v_\alpha(o(\phi_1,\ldots,\phi_n))=r_o(v_\alpha(\phi_1),\ldots,v_\alpha(\phi_n))$, where $o\in\mathscr{O}$ with arity n, and r_o interprets o;
- $v_\alpha(q\ x.\ \phi)=g_q(\{v_\alpha(\phi[x:=t])\mid t\in Tm(\mathscr{F})\})$.

An interpretation v_α is a *model* for a sentence ϕ iff $v_\alpha(\phi)\in\mathscr{D}$. Given two formulae ϕ,ϕ' we say that ϕ' is a *logical consequence* of ϕ, denoted by $\phi\models\phi'$, iff $v_\alpha(\phi')\in\mathscr{D}$, for all models v_α of ϕ.

As an example, consider the language $\mathscr{L}=(\mathscr{P}_c,\mathscr{F}_c,\mathscr{V}_c,\mathscr{O}_c,\mathscr{Q}_c)$ of *classical* first-order logic, where $\mathscr{O}_c=\{\wedge,\vee,\neg,\bot\}$ and $\mathscr{Q}=\{\forall,\exists\}$. In the corresponding valuation system, we have $\mathscr{T}=\{\mathbf{t},\mathbf{f}\}$, $\mathscr{D}=\{\mathbf{t}\}$, $f_\wedge,f_\vee,f_\neg$ are the classical interpretations of the connectives $\wedge,\vee$ and $\neg$, while $f_\bot$ is a constant function which returns the truth value $\mathbf{f}$. Finally, the functions $g_\forall$ and $g_\exists$ are defined as follows:

$$g_\forall(X)=\begin{cases}\mathbf{f} & \text{if } \mathbf{f}\in X,\\ \mathbf{t} & \text{otherwise,}\end{cases}\qquad g_\exists(X)=\begin{cases}\mathbf{t} & \text{if } \mathbf{t}\in X,\\ \mathbf{f} & \text{otherwise.}\end{cases}$$

It is worth noting that this definition of $g_\forall$ (resp. $g_\exists$) corresponds to the usual interpretation of the universal (resp. existential) quantifier as a possibly infinite conjunction

(resp. disjunction). Later we will use an ordered set for X in the above for sequential quantification.

2.2. Logic programming

We assume that the reader is familiar with logic programming, and so we recall only some basic definitions. For the concepts which are not reported here, the reader can refer to [18, 3]. Logic programming is based on a predicate language $\mathscr{L}_{lp} = (\mathscr{P}_{lp}, \mathscr{F}_{lp}, \mathscr{V}_{lp}, \mathscr{O}_{lp}, \mathscr{Q}_{lp})$, where $\mathscr{O}_{lp} = \{\wedge, \leftarrow, \mathit{false}\}$ and $\mathscr{Q}_{lp} = \{\forall, \exists\}$. A *definite clause* is a formula of the form $A \leftarrow \overline{B}$ where A is an atom and $\overline{B}$ is a conjunction $B_1 \wedge \cdots \wedge B_n$ of atoms. A is called the *head* of the clause, and $\overline{B}$ is called the *body* of the clause. All the variables occurring in a clause are implicitly universally quantified. Hence, $A \leftarrow \overline{B}$ stands for $\forall x_1 \cdots \forall x_k.\ A \leftarrow \overline{B}$, where $x_1, \ldots, x_k$ are all the variables (possibly none) occurring in the clause.

A *logic program* is a (finite) set of definite clauses P. A *goal* is a clause with an empty head, denoted by $\leftarrow \overline{B}$.

The declarative semantics of logic programs is given by classical two-valued logic and *Herbrand interpretations*. The interpretation of the operators in $\mathscr{O}_{lp}$ is the classical one, namely $\wedge$ is interpreted as conjunction, $\leftarrow$ as consequence and *false* as the truth value **f**.

The ground term algebra $Tm(\mathscr{F})$ is referred to as the *Herbrand Universe*. A *Herbrand interpretation* is a valuation v_α corresponding to a basic assignment $\langle \rho, Tm(\mathscr{F}) \rangle$, in which the domain of individuals is the Herbrand Universe. The standard semantics of a logic program P based on a first-order language $\mathscr{L}$ is given by its *least* Herbrand model.

On the other hand, the operational semantics of logic programs is given in terms of *SLD-resolution* and the *SLD-refutation procedure*. Given a logic program P and a goal G, an *SLD-tree* for P and G is a tree satisfying the following: (i) each node of the tree is a (possible empty) goal, and (ii) the root node is G, and (iii) let $B_1, B_2, \ldots, B_s, \ldots, B_m$ $(m \geq 1)$ be a node in the tree and B_s be the selected atom for this node via a *computation rule*. Then, for each clause $A \leftarrow \overline{B}$ such that $mgu(A, B_s) = \vartheta \neq \mathit{fail}$, the node has child $(B_1, B_2, \ldots, \overline{B}, \ldots, B_m)\vartheta$, where $mgu(A, B)$ denotes the most general unifier of A and B, which is *fail* if A and B do not unify.

A *search rule* is a strategy for searching SLD-trees. An *SLD-refutation procedure* is specified by a computation rule together with a search rule. *Success branches* in an SLD-tree are the ones ending in the empty goal, while *failure branches* are the ones ending in a non-empty node without children.

The operational semantics of Prolog corresponds to a particular way of constructing and visiting SLD-trees, which can be formalized as follows. A *Prolog-tree* is an SLD-tree such that

- the computation rule is the left-to-right one (i.e., $s = 1$);
- the children of a non-leaf node are obtained (from left to right) by considering the clauses in the textual order they appear in the program.

Finally, the operational semantics of Prolog corresponds to the list of substitutions at success nodes encountered in the left-to-right depth-first traversal of the Prolog-tree for a given goal.

3. Related works

The idea of giving the semantics to logic programs by means of a multiple-valued logic is not new. A third value, the *undefined* value **u**, was introduced to model infinite computations. Examples of semantic definitions based on Kleene's three-valued logic are the ones in [14, 17, 20]. These logical semantics are defined for pure logic programming, i.e., they model an operational behaviour based on SLD-trees built by a fair computation rule and visited by a breadth-first strategy. Recall that a *fair* computation rule is such that any (instance of) atom occurring in a goal is eventually selected.

In [2], a semantics for Prolog in a logical style is presented. This logical semantics is proved correct with respect to an operational one, which essentially mimics the left-to-right depth-first visit of a Prolog tree. The left-to-right depth-first search rule is taken into account by using the completion of programs and by giving a sequential interpretation to the disjunction in the right part of each predicate-completed definition. Of course, in this case, the order of the arguments of the disjunction is essential. This order must respect exactly the order of clauses. Moreover, the left-to-right computation rule is modelled by the sequential interpretation of the conjunction.

To define the semantics with respect to the completion of a program, [2] generalizes the standard notion of *goal*, in the style of [19]. We refer to a similar notion of goal, which reflects the type of formulae which arise when Clark's completion is taken into account. To simplify the notation, in the following we denote simply by x and t a sequence of variables and terms, respectively. Moreover, we denote simply by $s = t$ a conjunction of equations $s_1 = t_1 \wedge \cdots \wedge s_k = t_k$ where s_i, t_i are terms. So, for instance, $x = t$ may denote a conjunction of equations of the form $x_1 = t_1 \wedge \cdots \wedge x_n = t_n$.

Definition 1. A *goal* is an element of the syntactic category `Disj_Goal` defined as follows:

```
Disj_Goal ::= Ex_Goal ∨ Disj_Goal | Ex_Goal
Ex_Goal ::= Conj_Goal | ∃x. s = t ∧ Conj_Goal
Conj_Goal ::= Atom | Atom ∧ Conj_Goal
Atom ::= false | true | p(t)
```

In the rest of the paper we will refer to this definition of goal.

As an example, consider the Prolog program

$$p(t) \leftarrow q(t'). \qquad p(s) \leftarrow r(s').$$

where t, t', s, s' are sequences of terms. Its completion is given by

$$p(x) \leftrightarrow \exists y.(x = t \wedge q(t')) \vee \exists z.(x = s \wedge r(s')),$$

where y, z are the sequences of variables occurring in the two clauses, respectively.

The semantics of Prolog is obtained by giving an order to the evaluation of $\vee$, so that, when evaluating the definition of $p(x)$, the part corresponding to the first clause is examined first, and the second part is evaluated only if the truth result of the first one is **f**. The same evaluation order is given to $\wedge$. The evaluation order of $\vee$ models the sequential use of the clauses, while the one of $\wedge$ models the left-to-right computation rule.

More formally, [2] gives the semantics in terms of the three truth values, $\{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$ [2] with the following interpretation of $\vee$ and $\wedge$. The function v is a *valuation function* mapping ground formulae onto truth values:

$$v(B \vee C) = \begin{cases} v(C) & \text{if } v(B) = \mathbf{f}, \\ v(B) & \text{otherwise}, \end{cases} \qquad v(B \wedge C) = \begin{cases} v(C) & \text{if } v(B) = \mathbf{t}, \\ v(B) & \text{otherwise}. \end{cases}$$

Unfortunately, this interpretation of the connectives does not allow a complete logical specification of Prolog semantics. Consider the following propositional Prolog program (in completed form):

$$p \leftrightarrow q \vee loop. \qquad loop \leftrightarrow loop. \qquad q \leftrightarrow true. \qquad r \leftrightarrow false.$$

where the computation of the predicate *loop* is infinite. In the three-valued logic this behaviour is modelled by assigning the truth value **u** to *loop*.

Consider the two goals p and r. It is easy to see that, by using the valuation function v and the predicate definitions, the first one has truth value **t**, while the second one has value **f**. Consider now the goal $p \wedge r$ which is equivalent, by using the definition of p, to $(q \vee loop) \wedge r$. By using the values of q, *loop* and r and the valuation function v we obtain the result **f**, but this is not the result we get by a Prolog interpreter. In fact, due to the backtracking, the goal $p \wedge r$ would run indefinitely, thus its truth value should be **u**.

If we expand the definition of p and we apply the distributivity of $\vee$ on $\wedge$ we obtain, from $p \wedge r$, the goal $(q \wedge r) \vee (loop \wedge r)$, which has the right value **u**.

Intuitively, the problem comes from the fact that the valuation function v cannot model backtracking on different alternatives in a predicate definition. It works well only on goals in which the alternatives are "compiled", by applying distributivity, in a disjunction. For this reason, [2] gives semantics to Prolog programs in two steps. In the first one a goal is transformed into an **O**-formula, that is a formula in which each disjunction is an immediate subformula of either a negation or other disjunctions.

[2] Actually, one more value, **n**, is used to model floundering of negation. We do not consider it, since in this paper we do not take negation into account.

Then, to these formulae, the valuation function is applied to get the truth value. In the above example, $(q \wedge r) \vee (loop \wedge r)$ is an **O**-formula, while $(q \vee loop) \wedge r$ is not.

Although [2] represents a step towards the definition of a logical semantics of Prolog, the approach is not completely satisfactory. It is not a truly logical semantics because it is not compositional on all the possible goals. In the previous example, the true value of $p \wedge r$ is not given simply by the *and* of the values of p and r. To get the correct value we have to apply a transformation, which has the sound of making some computation steps, to get a formula in the **O**-form.

In the following section, we will present a four-valued logic which can be used to give a compositional truly logical semantics of Prolog. The fourth truth value, which is denoted by $\mathbf{t}_u$, models the computational behaviour of a goal which has at least one solution, but whose computation is infinite.

A first intuition on the use of this fourth truth value can be found in [20], in which a value corresponding to $\mathbf{t}_u$ was used to model a goal which results in **t** under some evaluation strategies and **u** in others. However, this is not suitable for the semantics of Prolog, because a solution for a goal is given only if the success branch, corresponding to this solution, has no infinite branches on the left.

4. A logical semantics for propositional Prolog

Our aim is to give a compositional logical semantics to Prolog by using a four-valued logic. For the sake of clarity, we first define the semantics of *propositional Prolog*. In the next section this semantics is extended to full Prolog.

4.1. Propositional multiple-valued logics

Throughout this section we refer to a *propositional* language. Propositional languages can be viewed as special cases of predicate languages, where predicate symbols have all arity 0, and neither variables, nor function symbols nor quantifiers are considered. Hence, a propositional language $\mathscr{L}$ is simply a pair $\langle \mathscr{P}, \mathscr{O} \rangle$, where $\mathscr{P}$ is a collection of predicate symbols and $\mathscr{O}$ is a set of connectives. The set of *sentences* of a propositional language $\mathscr{L}$ is the smallest set containing $\mathscr{P}$ and closed under the operators $\mathscr{O}$.

The notion of valuation system for a propositional language is a simplified version of the one given in Section 2.1. Here, a valuation system $\mathscr{V}$ is simply a triple $\langle \mathscr{T}, \mathscr{D}, \mathscr{R} \rangle$ where

- $\mathscr{T}$ is the set of truth values, with at least two elements;
- $\mathscr{D}$ is the set of *designated* truth values, a non-empty proper subset of $\mathscr{T}$;
- $\mathscr{R}$ is a set of functions. Each function $r_o \in \mathscr{R}$ corresponds to one operator in $\mathscr{O}$, and it is such that $r_o : \mathscr{T}^{n_o} \to \mathscr{T}$ (where n_o is the arity of the operator o).

Finally, a *basic assignment* ρ relative to a valuation system $\langle \mathscr{T}, \mathscr{D}, \mathscr{R} \rangle$ for a propositional language $\langle \mathscr{P}, \mathscr{O} \rangle$ is simply a mapping $\rho : \mathscr{P} \to \mathscr{T}$. Each basic assignment induces

an *interpretation* (or *valuation*) v_ρ of a sentence in the language, inductively defined by

- $v_\rho(p) = \rho(p)$, for $p \in \mathscr{P}$.
- $v_\rho(o(\phi_1, \ldots, \phi_n)) = r_o(v_\rho(\phi_1), \ldots, v_\rho(\phi_n))$, where n is the arity of o and r_o interprets o.

4.2. Propositional logic programming

A propositional logic program is a collection of definite clauses from a propositional language $\mathscr{L}_{lp} = \langle \mathscr{P}_{lp}, \mathscr{O}_{lp} \rangle$, where $\mathscr{P}_{lp}$ is a set of propositional symbols and $\mathscr{O}_{lp} = \{\wedge, \leftarrow, \mathit{false}\}$ (see Section 2.2).

Our semantics is based on the notion of *completion* of a logic program which is similar to Clark's completion [11]. The latter was originally introduced by Clark in order to provide a declarative semantics to negation as finite failure. However, (variants of) Clark's completion have been adopted to capture the logical meaning of Prolog, and this is the case also for our approach. Let us briefly recall the definition of Clark's completion for propositional logic programs.

Given a propositional language $\mathscr{L}_{lp}$ as before, let $\mathscr{L}_{comp} = \langle \mathscr{P}_{comp}, \mathscr{O}_{comp} \rangle$ be the language where $\mathscr{P}_{comp} = \mathscr{P}_{lp}$ and $\mathscr{O}_{comp} = \{\vee, \wedge, \leftrightarrow, \mathit{false}, \mathit{true}\}$.

Definition 2. Let P be a logic program on the language $\mathscr{L}_{lp}$ and let

$$C_{p_i} : p \leftarrow \overline{B}_i \quad 0 \leqslant i \leqslant m$$

be the sequence of m clauses of P with head p, where each $\overline{B}_i$ is a (possibly empty) conjunction of atoms. The *completed definition* of p is the following formula in the language $\mathscr{L}_{comp}$:

$$p \leftrightarrow \overline{E}_1 \vee \cdots \vee \overline{E}_m,$$

where $\overline{E}_i$ coincides with $\overline{B}_i$ if $\overline{B}_i$ is a non-empty conjunction, and $\overline{E}_i$ is true otherwise. If an atom $p \in \mathscr{P}_{lp}$ never occurs in the head of a clause in P, its *completed definition* is the formula $p \leftrightarrow \mathit{false}$ in the language $\mathscr{L}_{comp}$. The *completion* of a propositional Prolog program P, denoted by $comp(P)$ is the collection of the completed definitions of the predicates in $p \in \mathscr{P}_{comp}$.

For instance, the completion of the logic program

$$p \leftarrow q \wedge r$$
$$q \leftarrow$$

on a language such that $\mathscr{P}_{lp} = \{p, q, r\}$ is

$$p \leftrightarrow q \wedge r$$
$$q \leftrightarrow \mathit{true}$$
$$r \leftrightarrow \mathit{false}.$$

4.3. A four-valued logic

In this section we introduce the valuation system $\mathscr{V}_4$ defining the four-valued logic we will use to provide propositional Prolog with a logical semantics.

As mentioned in the previous section, the idea is to extend the usual three-valued logic by a fourth truth value, $\mathbf{t}_u$, which is intended to model an infinite Prolog tree which has at least a successful branch to the left of the first infinite one. Moreover, the connectives $\wedge$ and $\vee$ are interpreted in a sequential manner, in order to reflect the operational behaviour of Prolog. Since the semantics of a Prolog program P is given in terms of (a variant of) its Clark's completion, we directly define our valuation system with respect to the language $\mathscr{L}_{comp} = \langle \mathscr{P}_{comp}, \mathscr{O}_{comp} \rangle$ introduced in Section 4.2.

Definition 3. The valuation system $\mathscr{V}_4 = \langle \mathscr{T}_4, \mathscr{D}_4, \mathscr{R}_4 \rangle$ is defined as follows:
- $\mathscr{T}_4 = \{\mathbf{f}, \mathbf{u}, \mathbf{t}_u, \mathbf{t}\}$;
- $\mathscr{D}_4 = \{\mathbf{t}\}$;
- The 0-ary functions f_{false} and f_{true} are defined as $f_{false} = \mathbf{f}$ and $f_{true} = \mathbf{t}$, the functions $f_\wedge, f_\vee$ are defined according to the following truth tables:

$x \backslash y$	$\mathbf{t}$	$\mathbf{t}_u$	$\mathbf{u}$	$\mathbf{f}$
$\mathbf{t}$	$\mathbf{t}$	$\mathbf{t}_u$	$\mathbf{u}$	$\mathbf{f}$
$\mathbf{t}_u$	$\mathbf{t}_u$	$\mathbf{t}_u$	$\mathbf{u}$	$\mathbf{u}$
$\mathbf{u}$	$\mathbf{u}$	$\mathbf{u}$	$\mathbf{u}$	$\mathbf{u}$
$\mathbf{f}$	$\mathbf{f}$	$\mathbf{f}$	$\mathbf{f}$	$\mathbf{f}$

$f_\wedge(x, y)$

$x \backslash y$	$\mathbf{t}$	$\mathbf{t}_u$	$\mathbf{u}$	$\mathbf{f}$
$\mathbf{t}$	$\mathbf{t}$	$\mathbf{t}_u$	$\mathbf{t}_u$	$\mathbf{t}$
$\mathbf{t}_u$	$\mathbf{t}_u$	$\mathbf{t}_u$	$\mathbf{t}_u$	$\mathbf{t}_u$
$\mathbf{u}$	$\mathbf{u}$	$\mathbf{u}$	$\mathbf{u}$	$\mathbf{u}$
$\mathbf{f}$	$\mathbf{t}$	$\mathbf{t}_u$	$\mathbf{u}$	$\mathbf{f}$

$f_\vee(x, y)$

and the function $f_{\leftrightarrow}$ is defined in the usual way:

$$f_{\leftrightarrow}(x, y) = \begin{cases} \mathbf{t} & \text{if } x = y, \\ \mathbf{f} & \text{otherwise.} \end{cases}$$

Let us explain intuitively the above definitions. The interpretation of $\wedge$ mimics the computation corresponding to a conjunction of goals. Since $\mathbf{t}$ is intended to model a finite success, if it is the first argument of a sequential conjunction the result is equivalent to the second argument. An argument equal to $\mathbf{t}_u$ models a computation in which there is at least a success and then it is infinite; if it is the first argument of a sequential conjunction, the resulting computation is still infinite, but the existence of a success, in the whole computation, depends on the value of the second argument. Finally, a value $\mathbf{f}$ or $\mathbf{u}$ for the first argument is the result of the whole computation.

On the other hand, the interpretation of $\vee$ mimics the result of exploring different alternatives in a computation of a Prolog goal. The first argument equal to $\mathbf{t}$ models the fact that we have got a finite success in the computation of the first alternative; of course, if the second argument corresponds to an infinite computation the result must reflect it. If the first argument is $\mathbf{t}_u$ we have at least a success and an infinite

computation independently from the behaviour of the second alternative. Obviously, when the first alternative has a finite computation without successes, the result of the whole computation is the one of the second alternative, and, finally, when the first alternative has an infinite computation without successes, we cannot pass to examine the second one.

Finally, the $\leftrightarrow$ connective is defined in the expected way also in our four-valued logic, that is it has **t** if and only if the two arguments have the same truth value. Otherwise its value is **f**.

4.4. Semantics of propositional Prolog

The semantics of a propositional Prolog program P is given in terms of its *sequential completion*, denoted by $s\_comp(P)$, which is similar to Clark's completion (see Definition 2). The only difference is that, when constructing the sequential completion of a program P, the textual order in which atoms occur in the body of a clause as well as the textual order in which clauses defining a predicate p occur in P determine exactly the syntactic form of the completed definition of p. Due to its similarity with the definition of Clark's completion, we omit the definition of $s\_comp(P)$ and we illustrate by means of a simple example. Consider the following clauses defining a predicate p:

$$p \leftarrow q \wedge r$$
$$p \leftarrow s$$

and assume that they appear in this order within a Prolog program P. Then, the completed definition of p in $s\_comp(P)$ is exactly the formula

$$p \leftrightarrow (q \wedge r) \vee s.$$

On the other hand, since Clark's completion is usually interpreted in a classical two-valued or three-valued logic, the formula

$$p \leftrightarrow s \vee (r \wedge q)$$

could equally be taken as the completed definition of p in Clark's completion.

It is important to notice that, for propositional Prolog, the notion of goal as in Definition 1 collapses down to the following definition.

Definition 4. A goal is an element of the syntactic category `Disj_Goal` defined as follows:

```
Disj_Goal  ::= Conj_Goal ∨ Disj_Goal | Conj_Goal
Conj_Goal  ::= Atom | Atom ∧ Conj_Goal
Atom       ::= false | true | Prop_Letter
Prop_Letter ::= p | q | ...
```

The first important observation is that in our four-valued logic we can logically model backtracking, as stated by the following proposition. Recall that a model of a formula, in a valuation system $\mathscr{V}=\langle\mathscr{T},\mathscr{D},\mathscr{F}\rangle$, is an interpretation v_ρ, where ρ is a basic assignment, which assigns to the formula a truth value in $\mathscr{D}$ (the truth value **t** in the case of our valuation system $\mathscr{V}_4$).

Proposition 5. *Given three goals, G, G' and G'', in propositional Prolog, every interpretation is a model of the formula*

$$((G \vee G') \wedge G'') \leftrightarrow ((G \wedge G'') \vee (G' \wedge G'')).$$

Stated otherwise, the formulae $(G \vee G') \wedge G''$ and $(G \wedge G'') \vee (G' \wedge G'')$ are equivalent, i.e. they have the same truth value in every interpretation.

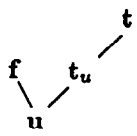
Let us consider the sequential completion of the example of Section 3.

$$p \leftrightarrow q \vee loop$$
$$loop \leftrightarrow loop$$
$$q \leftrightarrow true$$
$$r \leftrightarrow false$$

Assigning the truth value **u** to *loop*, the value of p is now $\mathbf{t}_u$ and, hence, the value of the goal $p \wedge r$ is **u**.

It is important to remark that the classical properties of $\vee$ and $\wedge$ are not preserved in our valuation system. For example, the formulae $G \wedge (G' \vee G'')$ and $(G \wedge G') \vee (G \wedge G'')$ are not equivalent.[3] However, we are interested in maintaining the properties which model the evolution of Prolog computations. In this respect, notice that $(G \wedge G') \vee (G \wedge G'')$ does not model the evolution of the computation of the goal $G \wedge (G' \vee G'')$.

Now, we can define the model-theoretic semantics of propositional Prolog as the *least* model of $s\_comp(P)$ with respect to a suitable ordering between basic assignments. This ordering is the pointwise ordering obtained from an ordering between the four truth values based on the following Hasse diagram:



Following [15], we will refer to this ordering as the *knowledge ordering*, denoted by $\leqslant_K$.

[3] Consider the values $\mathbf{t}_u$, **f** and **t** for the goals G, G' and G'', respectively. The goal $G \wedge (G' \vee G'')$ has truth value $\mathbf{t}_u$, while $(G \wedge G') \vee (G \wedge G'')$ has value **u**.

Definition 6. Given a propositional language $\mathscr{L} = \langle \mathscr{P}, \mathscr{O} \rangle$, let $\rho, \rho' : \mathscr{P} \to \mathscr{D}_4$ be two basic assignments relative to the valuation system $\mathscr{V}_4$. We say that ρ is less than or equal than ρ', denoted by $\rho \leqslant_K \rho'$, iff for all atoms $p \in \mathscr{P}$ we have $\rho(p) \leqslant_K \rho'(p)$.

It is clear that the set of basic assignments relative to $\mathscr{V}_4$ is a complete partial order with respect to $\leqslant_K$.

Proposition 7. *Let $\mathscr{L}$ be a propositional language, $\mathscr{V}_4$ be the corresponding valuation system, and $\mathscr{A}$ be the set of basic assignments relative to $\mathscr{V}_4$. Then the poset $(\mathscr{A}, \leqslant_K)$ is a complete partial order.*

Proof. Straightforward. □

The next proposition states the existence of the least model of *s_comp*(P).

Proposition 8. *Given a propositional Prolog program P, the set of all models of s_comp(P) has a least element with respect to $\leqslant_K$.*

The existence of the least model is based on the definition of a suitable bottom-up operator $\mathscr{T}_P$ associated with any program P, which is the analogue of the Fitting operator Φ_P for the three-valued case.

Definition 9. Let P be a propositional Prolog program and $\mathscr{L} = \langle \mathscr{P}, \mathscr{O} \rangle$ be the language of its sequential completion. The operator $\mathscr{T}_P$ mapping basic assignments in $\mathscr{A}$ to basic assignments in $\mathscr{A}$ is defined as follows. For each predicate symbol $p \in \mathscr{P}$

$$\mathscr{T}_P(\rho)(p) = v_\rho(\phi),$$

where $p \leftrightarrow \phi$ is the sequential completed definition of p in *s_comp*(P), and v_ρ is the valuation induced by ρ.

Lemma 10. *Let P be a propositional Prolog program, $\mathscr{L} = \langle \mathscr{P}, \mathscr{O} \rangle$ be the language of its sequential completion, and $\mathscr{T}_P$ be the operator associated with P. Then the following facts hold:*
(i) *$\mathscr{T}_P$ is continuous with respect to $<_k$,*
(ii) *a valuation v_ρ is a model of s_comp(P) iff ρ is a fixpoint of $\mathscr{T}_P$.*

Proof. (i) based on the continuity of $\wedge$ and $\vee$ with respect to $<_k$;
(ii) let $p \in \mathscr{P}$ and $p \leftrightarrow \phi$ be the sequential completed definition of p in *s_comp*(P). Assume that the basic assignment ρ is a fixpoint of $\mathscr{T}_P$. Then,

$$v_\rho(p) = \rho(p) = \mathscr{T}_P(\rho)(p) = v_\rho(\phi).$$

Hence, v_ρ is a model of $s\_comp(P)$. On the other hand, let v_ρ be a model of $s\_comp(P)$. Then

$$\rho(p) = v_\rho(p) = v_\rho(\phi) = \mathscr{T}_P(\rho)(p),$$

hence, ρ is a fixpoint of $\mathscr{T}_P$. □

As a consequence, we have that the interpretation $v_{\min}$ induced by the least fixpoint of the $\mathscr{T}_P$ operator is the least model of $s\_comp(P)$.

Finally, we show that the least model of $s\_comp(P)$ reflects indeed the operational behaviour of Prolog.

Theorem 11. *Let P be a propositional Prolog program, $v_{\min}$ the least model of $s\_comp(P)$ and G a goal:*

$v_{\min}(G) = \mathbf{t}$ iff the Prolog tree of P and G is finite, and it contains at least one success branch
$v_{\min}(G) = \mathbf{f}$ iff the Prolog tree of P and G is finite, and it does not contain any success branch
$v_{\min}(G) = \mathbf{t}_u$ iff the Prolog tree of P and G is infinite, and it contains at least a success branch on the left of the first infinite branch
$v_{\min}(G) = \mathbf{u}$ iff the Prolog tree of P and G is infinite, and it contains no success branch on the left of the first infinite branch.

Proof. See Appendix A. □

5. Pure Prolog

In this section we extend the logical semantics to pure Prolog. The sequential completion of a Prolog program is obtained by extending the one of Section 4.4 to programs on a predicate language $\mathscr{L} = (\mathscr{P}, \mathscr{F}, \mathscr{V}, \mathscr{O}, \mathscr{Q})$ (which we consider fixed from now onwards), where $\mathscr{O}$ is the set of operators $\{\vee, \wedge, \leftrightarrow, \mathit{false}, \mathit{true}\}$ and $\mathscr{Q} = \{\exists\}$.

The sequential completion of a Prolog program is again very similar to Clark's completion. Each clause

$$p(t_1, t_2, \ldots, t_k) \leftarrow \overline{B}$$

is transformed into

$$p(x_1, x_2, \ldots, x_k) \leftrightarrow \exists y . x_1 = t_1 \wedge x_2 = t_2 \wedge \cdots \wedge x_k = t_k \wedge \overline{B},$$

where $x_1, \ldots, x_k$ are new variables and y is the sequence of variables occurring in the original clause. Then the process proceeds as in the sequential completion of propositional programs. Recall that both the order in which atoms appear in clause bodies and the order of clauses is relevant when constructing the sequential completion. In addition

to the completed definitions of predicates, the sequential completion is equipped with the axioms of Clark's equality theory [11] which are used to interpret the symbol $=$ as the syntactic identity.

Recall that we refer to the definition of goal as given in Definition 1.

To give a logical semantics to Prolog we have to give a meaning to $\exists$. It is worth noting that the meaning of a formula $\exists x.G$, cannot be simply given classically by the disjunction of $G[x := t]$ under all possible assignments for x. Actually, the disjunction must be interpreted as a form of sequential disjunction, and the assignments have to be considered following a *special* order. To have an intuition of this let us give an example.

Example 12. Consider the following Prolog program on a first-order language in which $\mathscr{F} = \{a, b\}$ and $\mathscr{P} = \{p\}$.

$$p(b) \leftarrow p(b). \quad p(a).$$

Its sequential completion is given by

$$p(x) \leftrightarrow (x = b \wedge p(b)) \vee (x = a \wedge \mathit{true}).$$

Obviously, the Prolog goal $p(x)$ will loop, and this should be reflected by the meaning of the goal $\exists x.p(x)$. The meaning of $\exists x.p(x)$ cannot be simply given by the disjunction $p(x)[x := a] \vee p(x)[x := b]$, because this disjunction has truth value **t** in the least model of the program. Note that the program could be viewed as a propositional one, thus, in its least model the atom $p(b)$ would have value **u**, while $p(a)$ would have value **t**.

On the other hand, the meaning of $\exists x.p(x)$ cannot be given either by the sequential disjunction $p(x)[x := a] \vee p(x)[x := b]$, because in this case its truth value would be $\mathbf{t}_u$.

The value for the goal, corresponding to the operational behaviour of Prolog, is given by the sequential disjunction $p(x)[x := b] \vee p(x)[x := a]$ in which the assignments for x are taken in the "right" order. This order is essentially given by the order of clauses in the Prolog program.

In the following, we define the order in which the assignments for the variables must be taken to model the computational behaviour of Prolog. In a previous version of this work [8], we defined such an order by means of a relation $\prec$ between ground substitutions, defined inductively on the structure of the program. However, the same notion can be given an easier and more intuitive characterization, by exploiting the notion of sequences used in [9] to give a bottom-up semantics for Prolog.

5.1. *The bottom-up semantics of Bossi et al.*

In [9], Bossi et al. present a fixpoint reconstruction of the semantics of Prolog which captures both the left-to-right selection rule and the depth-first search strategy

underlying the Prolog evaluation mechanism. We summarize the most relevant definitions and results below, and we refer to [9] for further details.

Roughly speaking, the idea underlying the approach of [9] is to make use of an extended notion of interpretations, rather than Herbrand interpretations, which include more complex syntactic objects than ordinary ground atoms. An interpretation is a set of sequences, composed by either ordinary (possibly non-ground) atoms or *divergent* atoms. A sequence in an interpretation is an abstraction of the ordered set of partial answers obtained by a depth-first and left-to-right traversal of a partial Prolog tree for a query in its most general form (i.e. $p(x)$ where x is a tuple of variables). A divergent atom in a sequence represents the fact that the corresponding branch of the partial Prolog tree can be further expanded.

Given a set of symbols S let us denote by S^* the set of finite sequences of symbols in S. The concatenation of two sequences s_1 and s_2 is denoted by $s_1 :: s_2$, whereas λ stands for the empty sequence.

Definition 13. Let $\mathscr{B} = Am(\mathscr{P}, \mathscr{F}, \mathscr{V})$. The extended base $\mathscr{B}_E$ is defined as

$$\mathscr{B}_E = \{\widehat{A} \mid A \in \mathscr{B}\} \cup \mathscr{B}.$$

The newly introduced atoms of the form $\widehat{A}$ are referred to as *divergent* atoms. The set $\mathscr{P}(\mathscr{B}_E^*)$ is a complete lattice under the usual inclusion ordering, with top and bottom elements defined, respectively, as $\mathscr{B}_E^*$ and $\emptyset$.

Sequences in $\mathscr{B}_E^*$ can be used to represent the frontiers of any finite cut of a Prolog tree, and such sets can be constructed bottom-up by a suitable operator. Recall that a finite cut at depth k of a tree is the tree obtained by cutting at depth k each branch longer than k. As mentioned above, the presence of a divergent atom in a sequence represents the fact that the sequence corresponds to the frontier of a cut of the Prolog tree containing nodes which can be further expanded, i.e. they are neither failure nor success nodes.

As an example, consider the following Prolog program:

$$p(a).$$

$$p(b) \leftarrow p(b).$$

The sequence

$$p(a) :: \widehat{p(b)}$$

represents any cut at level $k \geqslant 1$ of the Prolog tree corresponding to the goal $p(x)$. In fact, the leftmost branch of this tree is a success branch with answer substitution $x = a$, whereas the rightmost branch is an infinite one, which can be expanded indefinitely by using the second clause.

In [9], an interpretation μ is a set of sequences of elements in $\mathscr{B}_E$. Given an interpretation μ, a sequence $S \in \mu$ and an atom A, we define $\pi_A(S)$, the projection of A on S, as the subsequence of S which consists of the elements which unify with A. If

A is in most general form, say $A = p(x)$, then $\pi_p(S)$ will be used as a shorthand for $\pi_{p(x)}(S)$. In the next definition, clauses are assumed to be standardized apart each time they are used, in order to avoid variables clashes.

Definition 14. Let c be a clause. $\phi_c : \mathscr{B}_E^* \to \mathscr{B}_E^*$ is defined as follows:
- If c is the unit clause A, then $\phi_A(S) = A$.
- If $c = (A \leftarrow B, \overline{D})$ and $\pi_B(S) = d_1 :: \cdots :: d_k$. Then

$$\phi_c(S) = \alpha_1 :: \cdots :: \alpha_k \text{ where } \alpha_i = \begin{cases} \widehat{A\theta_i} & \text{if } d_i = \widehat{B'}, \\ \phi_{(A \leftarrow D)\theta_i}(S) & \text{if } d_i = B' \end{cases}$$

and $\theta_i = mgu(B, B')$ for $i \in [1..k]$.

Definition 15. Let P be a program and $c_1, \ldots, c_n$ the clauses of P. $\phi_P : \mathscr{B}_E^* \to \mathscr{B}_E^*$ is defined as $\phi_P(S) = \phi_{c_1}(S) :: \cdots :: \phi_{c_n}(S)$.

Definition 16. Let P be a program. Then $P^\sharp = \widehat{p_1(x_1)} :: \cdots :: \widehat{p_n(x_n)}$, where $p_1, \ldots p_n$ are the predicate symbols in P.

Definition 17. The operator $\Phi_P : \mathscr{P}(\mathscr{B}_E^*) \to \mathscr{P}(\mathscr{B}_E^*)$ is defined in terms of ϕ_P as follows:

$$\Phi_P(\mu) = \{\phi_P(S) \mid S \in \mu\} \cup \{P^\sharp\}.$$

The least fixpoint of the operator Φ_P, $\mathscr{S}_{DFL}(P)$, consists of the ordered set of the sequences over $\mathscr{B}_E$ which represent all the possible partial computations (w.r.t. the depth-first left-to-right derivation) originating from the most general atoms in the program.

Definition 18. $\mathscr{S}_{DFL}(P) = \Phi_P^\omega(\emptyset)$.

Consider the interpretations $\mu_0, \ldots, \mu_k, \ldots$ resulting from the iterative computation of the fixpoint. At the first step, $\mu_0 = \Phi_P(\emptyset) = \{P^\sharp\}$. Similarly, at the second step, $\mu_1 = \Phi_P(\mu_0) = \{P^\sharp, \phi_P(P^\sharp)\}$. In general, at step k, μ_k will consist of $k+1$ sequences $S_0, \ldots, S_k$ and these sequences can be ordered so as to ensure that, for any $j \in [1..k]$, $\phi_P(S_{j-1}) = S_j$. The fixpoint $\mathscr{S}_{DFL}(P)$ can then be viewed as the limit interpretation μ_ω consisting of the (finite or infinite) ordered set of sequences $S_0, \ldots, S_k, \ldots$

Consider the following example taken from [9]. Given the Prolog program P,

$p(b).$

$p(x) \leftarrow r(x)$

$p(c).$

$r(a) \leftarrow p(a)$

$r(b) \leftarrow q(b)$

the least fixpoint $\mathscr{S}_{DFL}(P)$ is computed at a finite iteration and it consists of the following set of ordered sequences:

$$S_0 = \widehat{p(x)} :: \widehat{r(x)} :: \widehat{q(x)}$$

$$S_1 = p(b) :: \widehat{p(x)} :: p(c) :: \widehat{r(a)} :: \widehat{r(b)}$$

$$S_2 = p(b) :: \widehat{p(a)} :: \widehat{p(b)} :: p(c) :: \widehat{r(a)}$$

$$S_3 = p(b) :: \widehat{p(a)} :: p(c) :: \widehat{r(a)}.$$

Notice that divergent atoms can evolve to ordinary atoms or they can disappear in later sequences, representing success or failure branches, respectively.

Consider now the program

$$p(0).$$

$$p(s(x)) \leftarrow p(x).$$

In this case the least fixpoint requires infinitely many iterations and it contains sequences of the form

$$S_k = p(0) :: p(s(0)) :: \cdots :: p(s^{k-1}(0)) :: \widehat{p(s^k(0))}.$$

Finally, consider the program

$$p(s(x)) \leftarrow p(x)$$

$$p(0).$$

Also in this case the least fixpoint requires infinitely many iterations and it contains sequences of the form

$$S_k = \widehat{p(s^k(0))} :: p(s^{k-1}(0)) :: \cdots :: p(s(0)) :: p(0).$$

Among others, an important result which is a direct consequence of the results reported in [9] and which we will use is the following.

Proposition 19 ([9]). *Let $\exists y.G$ be a closed goal. Then the following facts hold:*

(i) *the Prolog tree of G is finite and it contains at least one success branch iff for some k, S_k is non-empty and does not contain divergent atoms*

(ii) *the Prolog tree of G is finite and it contains no success branch iff for some k, $S_k = \lambda$.*

(iii) *the Prolog tree of G is infinite iff for each k, S_k is not empty and does contain a divergent atom.*

5.2. A logical semantics for pure Prolog

The problem with an existentially quantified goal of the form $\exists x.G$ is that the disjunction of all possible ground instances $G[x:=g]$ must be taken in the right order corresponding to the order of evaluation of Prolog. The idea is that this order is represented by the sequences in $\mathscr{S}_{DFL}(P)$. The projection of a sequence S_k in $\mathscr{S}_{DFL}(P)$ over a goal $p(x)$ can be seen as representing the frontier of the cut at level k of the Prolog tree for $p(x)$, which induces an order between ground instances. For instance, given the goal $\exists x.p(x)$, the order induced by the sequence $p(a)::p(b)::p(c)$ imposes that $[x:=a]$ must precede $[x:=b]$ which itself must precede $[x:=c]$. Hence, in the disjunction corresponding to the evaluation of $\exists x.p(x)$ with respect to a basic assignment ρ, $\rho(p(x)[x:=a])$ must occur before $\rho(p(x)[x:=b])$ and the latter must occur before $\rho(p(x)[x:=c])$. Consider now the sequence $p(a)::\widehat{p(b)}::p(c)$, in which the divergent atom $\widehat{p(b)}$ represents a potentially infinite path in the Prolog tree. This is reflected, in the evaluation of $\exists x.p(x)$, by taking the disjunction $\rho(p(x)[x:=a])\vee\mathbf{u}\vee\cdots$.

The actual truth value of an existential goal should be computed as a function of the truth value of the same goal with respect to *any* sequence in $S_{DFL}(P)$. Since a set $\mathscr{I}$ of truth values may not have a maximal value with respect to the $\leqslant_\kappa$ ordering, we define the $\max^f(\mathscr{I})$ function which in such cases ignores the **f** value.

Definition 20. Let $\mathscr{I}$ be a set of truth values in $\{\mathbf{t},\mathbf{f},\mathbf{t}_u,\mathbf{u}\}$. Then $\max^f(\mathscr{I})$ is defined as follows:

$$\max^f(\mathscr{I})=\begin{cases}\max_\kappa(\mathscr{I}\backslash\{\mathbf{f}\}) & \text{if } \mathscr{I}\cap\{\mathbf{t},\mathbf{t}_u\}\neq\emptyset,\\ \max_\kappa(\mathscr{I}) & \text{otherwise,}\end{cases}$$

where $\max_\kappa$ is the maximal value with respect to the $\leqslant_\kappa$ ordering.

Given a sequence and an assignment, we define the *sequential disjunction* of the sequence, which is based, according to the previous informal discussion, on the fact that the sequence dictates in which order the ground instances represented by the sequence should be evaluated.

Definition 21. Given a sequence $S=d_1::\cdots::d_n$ in $\mathscr{B}_E^*$ and an assignment ρ, the sequential disjunction corresponding to S in ρ, denoted by $\overrightarrow{\bigvee_\rho}(S)$ is defined by

$$\overrightarrow{\bigvee_\rho}(S)=\begin{cases}\mathbf{f} & \text{if } S=\lambda,\\ \widehat{v}_\rho(d_1)\vee\cdots\vee\widehat{v}_\rho(d_n) & \text{otherwise,}\end{cases}$$

where

$$\widehat{v}_\rho(d)=\begin{cases}\mathbf{u} & \text{if } d \text{ is a divergent atom,}\\ \max_t\{\rho(p(t)\theta\gamma)\mid\gamma \text{ ground}\} & \text{if } d=p(t)\theta,\end{cases}$$

where $\max_t$ is the maximal value with respect to the *truth* ordering $\mathbf{f}\leqslant_t\mathbf{u}\leqslant_t\mathbf{t}_u\leqslant_t\mathbf{t}$.

Notice that the above definition reflects the intuition that any divergent atom in a sequence "represents" the undefined truth value, whereas a non-divergent atom represents the logical disjunction of all possible truth values of its ground instances in the assignment ρ.

In the next definition we exploit the notion of sequential disjunction to assign a truth value to existentially quantified goals. Notice that in the definition the projection of a sequence is defined over a possibly non-atomic conjunctive goal. The trivial extension of the projection operator π over a conjunctive goal G is still denoted by π_G and it is given in Appendix A (see Definition 32).

Since $S_{DFL}(P)$, in general, can be an infinite set of sequences, we must consider the sequential disjunction of the goal with respect to any such sequence and then take the maximal non-false value obtained in this way.

Definition 22. Given a closed goal G, and an assignment ρ, the truth value of G with respect to the interpretation v_ρ, denoted by $v_\rho(G)$, is obtained as follows:
- **f** if G is *false*,
- **t** if G is *true*,
- the truth value $\rho(G)$ if G is $(t = t')$ or G is $p(t)$,
- the value of $v_\rho(G_0) \vee v_\rho(G_1)$ if G is the goal $G_0 \vee G_1$,
- the value of $v_\rho(G_0) \wedge v_\rho(G_1)$ if G is the goal $G_0 \wedge G_1$,
- $\max^f\{\overrightarrow{\bigvee_\rho}\alpha_i \mid \alpha_i = \pi_{G'}(S_i)$, for each $S_i \in S_{DFL}(P)\}$, if G is the goal $\exists x.G'$.

Now we can define the notion of model for pure Prolog.

Definition 23. Given a Prolog program P and an assignment ρ, the interpretation v_ρ induced by ρ is a model of $s\_comp(P)$ iff for every definition

$$p(x) \leftrightarrow G$$

in $s\_comp(P)$ we have that $\rho(p(x)[x := t])$ is the same truth value of $v_\rho(G[x := t])$, for each sequence t of terms in $Tm(\mathscr{F})$.

According to this notion of model, the results for propositional Prolog can be extended to pure Prolog. In particular, we can define a new $\mathscr{T}_P$ operator.

Definition 24. Let P be a Prolog program and $\mathscr{L} = \langle \mathscr{P}, \mathscr{F}, \mathscr{V}, \mathscr{O}, \{\exists\}\rangle$ be the language of its sequential completion. The operator $\mathscr{T}_P$ mapping basic assignments to basic assignments is defined as follows. For each predicate symbol $p \in \mathscr{P}$

$$\mathscr{T}_P(\rho)(p(t)) = v_\rho(\phi[x := t]),$$

where $p(x) \leftrightarrow \phi$ is the sequential completed definition of p in $s\_comp(P)$, t is a sequence of terms in $Tm(\mathscr{F})$ and v_ρ is the interpretation induced by ρ.

Proposition 25. *Let P be a Prolog program.*
(i) *An interpretation v_ρ is a model of s_comp(P) iff ρ is a fixpoint of $\mathscr{T}_P$.*
(ii) *The $\mathscr{T}_P$ operator is continuous.*

Thus, we have that the interpretation $v_{\min}$ induced by the least fixpoint of the $\mathscr{T}_P$ operator is the least model of *s_comp*(P).

Finally, as in the propositional case we show that the least model of *s_comp*(P) reflects indeed the operational behaviour of Prolog.

Theorem 26. *Let P be a Prolog program, $v_{\min}$ the least model of s_comp(P) and G be a closed goal belonging to* `Ex_Goal` *as in Definition* 1, *i.e. $G = G'$ if it does not contain variables, or $G = \exists x.G'$.*

$v_{\min}(G) = \mathbf{t}$ iff the Prolog tree of P and G' is finite, and it contains at least one success branch
$v_{\min}(G) = \mathbf{f}$ iff the Prolog tree of P and G' is finite, and it does not contain any success branch
$v_{\min}(G) = \mathbf{t}_u$ iff the Prolog tree of P and G' is infinite, and it contains at least a success branch on the left of the first infinite branch
$v_{\min}(G) = \mathbf{u}$ iff the Prolog tree of P and G' is infinite, and it contains no success branch on the left of the first infinite branch.

Proof. See Appendix A. □

Let us now give two examples.

Example 27. Consider the Prolog program

$$P: p(s(x)) \leftarrow p(x). \quad p(0).$$

Its sequential completion is given by $p(x) \leftrightarrow (\exists y.x = s(y) \wedge p(y)) \vee (x = 0 \wedge \mathit{true})$.

In the least model of the sequential completion all the atoms $p(s^n(0))$, with $n = 0, 1, 2, \ldots$, have value **t**.

Consider now the goal $G = \exists x.p(x)$. The computation of $\mathscr{S}_{DFL}(P)$ requires infinitely many iterations of $\Phi_P^\omega(\emptyset)$. At step k, μ_k contains the $k+1$ sequences $\{S_0, \ldots, S_k\}$, where for each j

$$S_j = \begin{cases} \widehat{p(x)} & j = 0, \\ \widehat{p(s^j(x))} :: p(s^{j-1}(0)) :: \cdots :: p(0) & j > 0. \end{cases}$$

The truth value of the goal is then obtained as

$$\max^f \{\mathbf{u},\ \mathbf{u} \vee \mathbf{t}, \ldots, \mathbf{u} \vee \mathbf{t} \vee \cdots \vee \mathbf{t}\}.$$

This corresponds to the value **u**, which is the value of the initial goal, $\exists x.p(x)$, as well.

This models the operational behaviour of the Prolog goal $p(x)$ with respect to the given program, which has an infinite Prolog tree where all the success branches are inaccessible since an infinite number of nodes precedes them in the leftmost depth-first traversal.

Example 28. Consider the Prolog program

$$P: \quad p(0). \quad p(s(x)) \leftarrow p(x).$$

Its sequential completion is given by $p(x) \leftrightarrow (x = 0 \wedge \mathit{true}) \vee (\exists y.x = s(y) \wedge p(y))$.

In the least model of the sequential completion all the atoms $p(s^n(0))$, with $n = 0, 1, 2, \ldots$, have value $\mathbf{t}$.

Consider now the goal $G = \exists x.p(x)$. The computation of $\mathcal{S}_{DFL}(P)$ requires infinitely many iterations of $\Phi_P^{\omega}(\emptyset)$. At step k, μ_k contains the k sequences $\{S_0, \ldots, S_k\}$, where for each j

$$S_j = \begin{cases} \widehat{p(x)} & j = 0, \\ p(0) :: \cdots :: p(s^{j-1}(0)) :: p(\widehat{s^j(x)}) & j > 0. \end{cases}$$

The truth value of the goal is then obtained as

$$\max^f \{\mathbf{u},\ \mathbf{t} \vee \mathbf{u}, \ldots, \mathbf{t} \vee \mathbf{t} \vee \cdots \vee \mathbf{u}\}.$$

This corresponds to the value $\mathbf{t}_u$.

This models the operational behaviour of the Prolog goal $p(x)$ which has an infinite Prolog tree with infinitely many success branches on the left of an infinite one.

6. Conclusion

We have shown how Prolog programming can be given a logical semantics based on a four-valued logic. Besides the usual *undefined* truth value, we have a fourth truth value $\mathbf{t}_u$ which models the computation of a goal which succeeds (at least once) and then loops. Future work will concentrate mainly on two issues. First of all, we plan to extend it to normal Prolog programs (i.e. Prolog programs with negation-as-failure), possibly adapting the approach of [2] where an extra truth value N is introduced to model *floundering*. Second, we plan to explore the possibility of further extending the approach to cope with other extra-logical features of Prolog.

Appendix A. Proofs

The proofs of the main results are performed by relating the construction of the least model $v_{\min}$ with the sequences in $S_{DFL}(P)$. The least model $v_{\min}$ is the least fixpoint

$\mathscr{T}_P^\omega$ of the $\mathscr{T}_P$ operator, where

$$\mathscr{T}_P^0 = \bot$$

$$\mathscr{T}_P^{n+1} = \mathscr{T}_P(\mathscr{T}_P^n)$$

$$\mathscr{T}_P^\omega = \bigsqcup_{n\in\omega} \mathscr{T}_P^n.$$

and $\bot$ assigns **u** to every atom.

We first give some useful definitions and technical lemmas and, for the sake of the reader, we give first the proofs for propositional Prolog and then the ones for pure Prolog.

Definition 29. Let S be a sequence in $\mathscr{B}_E^*$, $p(t), q(t'), r(s)$ be atoms, $S' = \pi_{p(t)}(S) = a_1 :: \cdots a_n$, $S'' = \pi_{q(t')}(S) = b_1 :: \cdots :: b_k$, where each a_i (resp. b_j) is either $p(t)\theta_i$ (resp. $q(t')\gamma_j$) or $\widehat{p(t)\theta_i}$ (resp. $\widehat{q(t')\gamma_j}$).

$$compose_{r(s)}(S', S'') = c_1 :: \cdots :: c_m,$$

where $c_1 :: \cdots :: c_m$ is obtained from S' and S'' as follows:
- $c_i = \widehat{r(s)\theta_i}$ if a_i is $\widehat{p(t)\theta_i}$;
- $c_i = b'_1 :: \cdots :: b'_k$, where each b'_j is λ iff $mgu(\theta_i, \gamma_j) = fail$, $b'_j = r(s)\delta_j$, $\delta_j = mgu(\theta_i, \gamma_j)$ otherwise (as usual, $mgu(\theta_i, \gamma_j)$ denotes the most general solution of the union of the set of equations corresponding to the substitution θ_i, γ_j).

Example 30. Assume S is a sequence such that $S' = \pi_{q(x)}(S) = \widehat{q(x)}[x := c] :: q(x)[x := a]$ and $S'' = \pi_{r(x,y)}(S) = r(x, y)[x := a, y := b] :: r(x, y)[x := b, y := b]$. Then

$$compose_{p(x)}(S', S'') = \widehat{p(x)}[x := c] :: p(x)[x := a, y := b].$$

Lemma 31. *Let $c : p(t) \leftarrow q_1(t_1), \ldots, q_k(t_k)$ be a clause in P, and let S be a sequence. For each $i = 1, \ldots, k$ let also α_i be the sequence for $q_i(t_i)$ given by $\pi_{q_i(t_i)}(S)$. Then*

$$\phi_c(S) = compose_{p(t)}(\alpha_1, \ldots compose_{q_{k-1}(t_{k-1})}(\alpha_{k-1}, \alpha_k)\ldots).$$

Proof. Obvious by definition of *compose* and $\phi_c(S)$. □

As an example, consider the clause $p(x) \leftarrow q(x), r(x, y)$ and the sequence S of the previous example.

The following definition is a slight extension of the projection operator π over sequences in $\mathscr{S}_{DFL}(P)$ for arbitrary goals G.

Definition 32. Let S_i be a sequence in $\mathscr{S}_{DFL}(P)$ and G be a goal.

$$\pi_G(S_i) = \begin{cases} \pi_{p(t)}(S_i) & \text{if } G = p(t) \\ \pi_{G_0}(S_i) :: \pi_{G_1}(S_i) & \text{if } G = G_0 \vee G_1 \\ \lambda & \text{if } G = \exists x.t = t' \wedge G' \text{ and } t \text{ and } t' \text{ do not unify} \\ \pi_{G'\theta}(S_i) & \text{if } G = \exists x.t = t' \wedge G' \text{ and } \theta = mgu(t,t') \\ compose_{ans_G(x)}(\alpha_1, \ldots compose_{q_{k-1}(t_{k-1})}(\alpha_{k-1}, \alpha_k)\ldots) & \\ & \text{if } G = q_1(t_1), \ldots, q_k(t_k), \text{ where } ans_G \text{ is a new} \\ & \text{predicate symbol, } x \text{ are the free variables in } G, \\ & \text{and for each } h = 1, \ldots, k,\ \alpha_h = \pi_{q_h(t_h)}(S_i). \end{cases}$$

We will make the following abuse of notation. Given a goal $G = q(t_1) \wedge \cdots \wedge q(t_m)$ and a sequence S_k in $\mathscr{S}_{DFL}(P)$ the sequence $\pi_G(S_k)$ ends up in a sequence of the form $d_1 :: \cdots :: d_n$, where each d_j is an atom of the form $ans_G(x)$ or $\widehat{ans_G(x)}$, where x are the free variables of G (see Definition 32). In this case, it may not be clear what $\rho(ans_G(x)\gamma)$ stands for, γ being a ground substitution for the variables x. However, we still use this notation with the proviso that

$$\rho(ans_G(x)\gamma) = \rho(q(t_1\gamma)) \wedge \cdots \wedge \rho(q(t_m\gamma)).$$

Lemma 33. *Let P be a logic program, $p(g)$ be a ground atom and*

$$\begin{aligned} c_1 &: p(t_1) \leftarrow G_1 \\ &\vdots \\ c_n &: p(t_n) \leftarrow G_n \end{aligned}$$

be the ordered set of clauses for p in P. Moreover, for each $j = 1, \ldots, n$, let

$$\alpha_j^k = \begin{cases} \lambda & \text{if there is no } mgu(g, t_j), \\ \pi_{G_j\theta_j}(S_k) & \text{if } \theta_j = mgu(g, t_j), \end{cases}$$

where S_k is a sequence in $\mathscr{S}_{DFL}(P)$. Then

$$\pi_{p(g)}(S_{k+1}) = \beta_1^k :: \cdots :: \beta_n^k,$$

where, for each $j = 1, \ldots, n$, $\beta_j^k = \lambda$ if $\alpha_j^k = \lambda$, and $\beta_j^k = d_1' :: \cdots :: d_m'$ if $\alpha_j^k = d_1 :: \cdots :: d_m$, where $d_i' = \widehat{p(g)}$ if d_i is a divergent atom, $d_i' = p(g)$ otherwise.

Proof. The proof is straightforward by definition of π and by definition of $\phi_c(S_k)$. □

The following proposition points out some useful properties of sequences in $\mathscr{S}_{DFL}(P)$ which are obvious consequences of the definition of the operator Φ_P in [9].

Proposition 34. *Let P be a logic program, and $\mathscr{S}_{DFL}(P) = \{S_0, S_1, \ldots, S_i, \ldots\}$. Given a goal G, the following facts hold:*

(i) *if for some* k, $\pi_G(S_k) = \lambda$ *then*
(i.1) *for each* $k' \geqslant k$, $\pi_G(S_{k'}) = \lambda$
(i.2) *for each* $k' < k$, *if* $\pi_G(S_{k'}) \neq \lambda$ *then* $\pi_G(S_{k'})$ *contains only divergent atoms.*
(ii) *if for some* k, $\pi_G(S_k) = d_1 :: \cdots :: d_n$ *and each* d_i *is not a divergent atom, then for each* $k' \geqslant k$, $\pi_G(S_{k'}) = \pi_G(S_k)$
(iii) *if for some* k, $\pi_G(S_k)$ *contains a non divergent atom* d, *then any* $\pi_G(S_{k'})$, *with* $k' \geqslant k$ *contains the same atom* d.

As we mentioned before, the proofs are based on relating the least fixpoint of the $\mathscr{T}_P$ operator to the sequences in $\mathscr{S}_{DFL}(P)$. The idea is that each S_k in $\mathscr{S}_{DFL}(P)$ can be associated with an assignment ρ^k, and that the limit assignment induced by these ρ^k coincide with $\mathscr{T}_P^{\omega}$.

Definition 35. Let S_k be a sequence in $\mathscr{S}_{DFL}(P)$. The assignment ρ^k associated with S_k is defined as follows:

$$\rho^k(p(t)) = \begin{cases} \mathbf{f} & \text{if } \pi_{p(t)}(S_k) = \lambda, \\ \mathbf{t} & \text{if } \pi_{p(t)}(S_k) \neq \lambda \text{ and does not contain divergent atoms,} \\ \mathbf{u} & \text{if } \pi_{p(t)}(S_k) = \widehat{p(x)}\theta :: S', \\ \mathbf{t}_u & \text{if } \pi_{p(t)}(S_k) = p(x)\theta_1 :: \cdots :: p(x)\theta_n :: \widehat{p(x)}\theta :: S'. \end{cases}$$

We will see that, in the propositional case, each ρ^k coincides with the $\mathscr{T}_P^k$, whereas in the non-propositional case the coincidence is obtained only at the limit assignment.

A.1. Proofs for propositional Prolog

In this section we will refer to a sequence of the form $p_0 :: \cdots :: p_k$ such that each p_i is either p or $\widehat{p}$ as a *sequence for* p. Obviously, given an arbitrary sequence S, $\pi_p(S)$ is a sequence for p.

Definition 36. Let $S = q_0 :: \cdots :: q_n$ be a sequence for q, and let p be a predicate symbol. Then

$$rename_p(S) = p_0 :: \cdots :: p_n,$$

where $p_i = p$ if $q_i = q$, and $p_i = \widehat{p}$ if $q_i = \widehat{q}$.

Definition 37. Let $S = p_0 :: \cdots :: p_n$ be a sequence for p, $S' = q_0 :: \cdots :: q_k$ be a sequence for q, and r be a predicate symbol. Then

$$compose_r(S, S') = d_1 :: \cdots :: d_n$$

where $d_i = \widehat{r}$ if $p_i = \widehat{p}$, and $d_i = rename_r(S')$ if $p_i = p$.

The following lemma is the analogous, for the propositional case, of Lemma 33.

Lemma 38. *Let $c: p \leftarrow q_1, \ldots, q_k$ be a clause for p, and let S be a sequence. For each $i = 1, \ldots, k$ let also α_i be the sequence for q_i given by $\pi_{q_i}(S)$. Then*

$$\phi_c(S) = compose_p(\alpha_1, \ldots compose_{q_{k-1}}(\alpha_{k-1}, \alpha_k)\ldots).$$

Proof. Straightforward by definition of ϕ_c. □

Lemma 39. *Let $S = p_0 :: \cdots :: p_n$ be a sequence for p, $S' = q_0 :: \cdots :: q_k$ be a sequence for q, and r be a predicate symbol. Then*
(i) $\rho^{compose_r(S,S')}(r) = \rho^S(p) \wedge \rho^{S'}(q)$,
(ii) $\rho^{rename_r(S) :: rename_r(S')}(r) = \rho^S(p) \vee \rho^{S'}(q)$.

Proof. Straightforward by case analysis on the truth value of $\rho^{compose_r(S,S')}(r)$ in case (i) and on the truth value of $\rho^{rename_r(S) :: rename_r(S')}(r)$ in case (ii). □

Lemma 40. *Let P be a propositional Prolog program, and let ρ_k be the assignment associated with the sequence S_k. Then, the assignments $\mathscr{T}_P^k$ and ρ_k coincide for each k.*

Proof. We show by induction that for each k and for each predicate symbol p $\mathscr{T}_P^k(p) = \rho_k(p)$.

Base step: Obvious.
Inductive step: Let

$$c_1 : p \leftarrow q_1^1 \wedge \cdots \wedge q_{h_1}^1$$
$$\ldots$$
$$c_n : p \leftarrow q_1^n \wedge \cdots \wedge q_{h_n}^n$$

be the ordered set of clauses defining p in P. Moreover, for each q_j^i, let $\alpha_j^i = \pi_{q_j^i}(S_k)$. For $i = 1, \ldots, n$, let

$$\beta_i = compose_p(\alpha_1^i, \ldots compose_{q_{h_i-1}^i}(\alpha_{h_i-1}^i, \alpha_{h_i}^i)\ldots).$$

Then, by definition of ϕ_P and Lemma 38,

$$\pi_p(S_{k+1}) = \beta_1 :: \cdots :: \beta_n.$$

By Lemma 39(i), we have that, for each $i = 1, \ldots, n$,

$$\rho^{\beta_i}(p) = \rho^{\alpha_1^i}(q_1^i) \wedge \cdots \wedge \rho^{\alpha_{h_i}^i}(q_{h_i}^i).$$

Hence, by the inductive hypothesis (since $\rho^{\alpha_j^i}(q_j^i) = \rho_k(q_j^i)$) we have that

$$\rho^{\beta_i}(p) = \mathscr{T}_P^k(q_1^i) \wedge \cdots \wedge \mathscr{T}_P^k(q_{h_i}^i). \qquad (\dagger)$$

Let $\beta = \beta_1 :: \cdots :: \beta_n$. We have that

$$
\begin{aligned}
& \rho_{k+1}(p) \\
= & \qquad \{\text{definition of } \rho_j\} \\
& \rho^{\beta}(p) \\
= & \qquad \{\text{Lemma 39(ii)}\} \\
& \rho^{\beta_1}(p) \vee \cdots \vee \rho^{\beta_n}(p) \\
= & \qquad \{\text{statement } (\dagger)\} \\
& (\mathscr{T}_P^k(q_1^1) \wedge \cdots \wedge \mathscr{T}_P^k(q_{h_1}^1)) \\
& \qquad \vee \cdots \vee \\
& (\mathscr{T}_P^k(q_1^n) \wedge \cdots \wedge \mathscr{T}_P^k(q_{h_n}^n)) \\
= & \qquad \{\text{definition of } \mathscr{T}_P\} \\
& \mathscr{T}_P^{k+1}(p). \quad \square
\end{aligned}
$$

Corollary 41. *Let P be a propositional Prolog program. Then $\rho^{\omega} = \mathscr{T}_P^{\omega}$.*

Proof of Theorem 11. The theorem is a consequence of Proposition 19, Corollary 41.

A.2. Proofs for pure Prolog

Lemma 42. *For each k and ground atom $p(g)$,*

$$\rho^k(p(g)) \leqslant_{\kappa} \rho^{k+1}(p(g)).$$

Proof. Straightforward by definition of ρ^k and Proposition 34. $\square$

By the previous lemma we can define the limit assignment ρ^{ω} as follows:

Definition 43.

$$\rho^{\omega}(p(g)) = \bigsqcup_{k \in \omega} \rho^k(p(g)),$$

where $\sqcup$ denotes the least upper bound with respect to $\leqslant_{\kappa}$.

The following lemma points out a useful property of ρ^{ω} which will be used in the sequel.

Lemma 44. *Let G be a goal of the form $\exists x.G'$, S_k be a sequence in $\mathscr{S}_{DFL}(P)$ and $\alpha_k = \pi_G(S_k)$. Then*

$$\overrightarrow{\bigvee_{\rho^{\omega}}} \alpha_k = \mathbf{f} \Leftrightarrow \alpha_k = \lambda.$$

Proof. $\alpha_k = \lambda \Rightarrow \overrightarrow{\bigvee_{\rho^{\omega}}} \alpha_k = \mathbf{f}$ by Definition 21.

Assume now $\overrightarrow{\bigvee_{\rho^\omega}}\alpha_k = \mathbf{f}$, and assume that $\alpha_k \neq \lambda$. By Definition 21, α_k is a non-empty sequence of the form $d_1 :: \cdots :: d_n$ where each d_j is not a divergent atom. Clearly, for each $j = 1,\ldots,n$ and for each ground substitution γ, we have that $\rho^k(d_j\gamma) = \mathbf{t}$, by Definition 35, and by Lemma 42, $\rho^{k+1}(d_j\gamma) = \mathbf{t}$. Hence, for each d_j,

$$\widehat{v}_{\rho^\omega}(d_j) = \mathbf{t},$$

which implies that $\overrightarrow{\bigvee_{\rho^\omega}}\alpha_k = \mathbf{t}$, contradiction. □

Lemma 45. *Let P be a logic program. Then*

$$\mathscr{T}_P(\rho^\omega) \leqslant_\kappa \rho^\omega.$$

Proof. Let $p(g)$ be an arbitrary ground atom and let

$$p(x) \leftrightarrow (\exists y.x = t_1 \wedge G_1) \vee \cdots \vee (\exists y.x = t_n \wedge G_n)$$

be the definition of p in $s\_comp(P)$. Then

$$\begin{aligned}
&\mathscr{T}_P(\rho^\omega)(p(g)) \\
=\ & && \{\text{definition of } \mathscr{T}_P\} \\
&v_{\rho^\omega}((\exists y.g = t_1 \wedge G_1) \vee \cdots \vee (\exists y.g = t_n \wedge G_n)) \\
=\ & && \{\text{definition of } \vee\} \\
&\bigvee_{i \in [1,n]} v_{\rho^\omega}(\exists y.g = t_i \wedge G_i).
\end{aligned}$$

Let $\mathscr{J} = \{j_1,\ldots,j_h\}$ be the ordered set of elements in $[1,n]$ such that there exists $\theta_{j_i} = mgu(g,t_{j_i})$. Obviously, for each $j \in [1,n]$ such that $j \notin \mathscr{J}$ we have that $v_{\rho^\omega}(\exists y.g = t_j \wedge G_j) = \mathbf{f}$ and hence the above disjunction reduces to

$$\bigvee_{j \in \mathscr{J}} v_{\rho^\omega}(\exists y.g = t_j \wedge G_j).$$

The proof is done by case analysis on the truth value of the last disjunction.

(i) if $\bigvee_{j\in\mathscr{J}} v_{\rho^\omega}(\exists y.g = t_j \wedge G_j) = \mathbf{u}$ there is nothing to prove, since $\mathbf{u}$ is the bottom element with respect to $\leqslant_\kappa$.

(ii) Assume $\bigvee_{j\in\mathscr{J}} v_{\rho^\omega}(\exists y.g = t_j \wedge G_j) = \mathbf{f}$.

We calculate

$$\begin{aligned}
&\bigvee_{j\in\mathscr{J}} v_{\rho^\omega}(\exists y.g = t_j \wedge G_j) = \mathbf{f} \\
\equiv\ & && \{\text{definition of } \vee\} \\
&\forall j \in \mathscr{J}.v_{\rho^\omega}(\exists y.g = t_j \wedge G_j) = \mathbf{f} \\
\equiv\ & && \{\text{by hypothesis there exists } \theta_j = mgu(g,t_j), \\
& && \text{definition of } v_\rho\} \\
&\forall j \in \mathscr{J}.v_{\rho^\omega}(\exists z_j.G_j\theta_j) = \mathbf{f}
\end{aligned}$$

where we have denoted by z_j the free variables in $G_j\theta_j$. For each j, let $G'_j = G_j\theta_j$.

We have

$$
\begin{aligned}
& v_{\rho^\omega}(\exists z_j.G'_j) \\
= & \qquad \{\text{definition of } v_\rho\} \\
& \max^f \left\{ \overrightarrow{\bigvee_{\rho^\omega}} \pi_{G'_j}(S_i) \,\middle|\, S_i \in \mathscr{S}_{DFL}(P) \right\} \\
= & \qquad \{\text{previous calculation}\} \\
& \mathbf{f}
\end{aligned}
$$

For each $j \in \mathscr{J}$ we have

$$
\begin{aligned}
& \max^f \left\{ \overrightarrow{\bigvee_{\rho^\omega}} \pi_{G'_j}(S_i) \,\middle|\, S_i \in \mathscr{S}_{DFL}(P) \right\} = \mathbf{f} \\
\equiv & \qquad \{\text{definition of } \max^f\} \\
& \forall i. \overrightarrow{\bigvee_{\rho^\omega}} \pi_{G'_j}(S_i) \in \{\mathbf{u}, \mathbf{f}\}, \text{ and} \\
& \exists \ell_j. \overrightarrow{\bigvee_{\rho^\omega}} \pi_{G'_j}(S_{\ell_j}) = \mathbf{f} \\
\equiv & \qquad \{\text{Lemma 44}\} \\
& \forall i. \overrightarrow{\bigvee_{\rho^\omega}} \pi_{G'_j}(S_i) \in \{\mathbf{u}, \mathbf{f}\}, \text{ and} \\
& \exists \ell_j. \pi_{G'_j}(S_{\ell_j}) = \lambda.
\end{aligned}
$$

Hence, for each $j \in \mathscr{J}$, there exists ℓ_j such that $\pi_{G'_j}(S_{\ell_j}) = \lambda$. Let then $\ell = \max\{\ell_j \mid j \in \mathscr{J}\}$. By Lemma 33, it is clear that $\pi_{p(g)}(S_{\ell+1}) = \lambda$, and hence $\rho^{\ell+1}(p(g)) = \mathbf{f}$. By this and Lemma 42 we conclude that $\rho^\omega(p(g)) = \mathbf{f}$.

(iii) Assume $\bigvee_{j \in \mathscr{J}} v_{\rho^\omega}(\exists y.g = t_j \wedge G_j) = \mathbf{t}$. We calculate

$$
\begin{aligned}
& \bigvee_{j \in \mathscr{J}} v_{\rho^\omega}(\exists y.g = t_j \wedge G_j) = \mathbf{t} \\
\equiv & \qquad \{\text{definition of } \vee\} \\
& \forall j \in \mathscr{J}. v_{\rho^\omega}(\exists y.g = t_j \wedge G_j) \in \{\mathbf{f}, \mathbf{t}\} \text{ and} \\
& \exists j \in \mathscr{J}. v_{\rho^\omega}(\exists y.g = t_j \wedge G_j) = \mathbf{t} \\
\equiv & \qquad \{\theta_j = mgu(g, t_j), \text{ definition of } v_\rho\} \\
& \forall j \in \mathscr{J}. v_{\rho^\omega}(\exists z_j.G_j\theta_j) \in \{\mathbf{f}, \mathbf{t}\} \text{ and} \\
& \exists j \in \mathscr{J}. v_{\rho^\omega}(G_j\theta_j) = \mathbf{t}.
\end{aligned}
$$

For each j, let $G'_j = G_j\theta_j$. Moreover, for each j such that $v_{\rho^\omega}(\exists z_j.G'_j) = \mathbf{f}$ we can reason as in the previous case and conclude that there exists ℓ_j such that $\pi_{G'_j}(S_{\ell_j}) = \lambda$.

For each j such that $v_{\rho^\omega}(\exists z_j.G'_j)=\mathbf{t}$ we have

$$\max^f\left\{\overrightarrow{\bigvee_{\rho^\omega}}\pi_{G'_j}(S_i)\,\middle|\,S_i\in\mathscr{S}_{DFL}(P)\right\}=\mathbf{t}$$

$\equiv$ {definition of $\max^f$}

$$\forall i.\overrightarrow{\bigvee_{\rho^\omega}}\pi_{G'_j}(S_i)\in\{\mathbf{u},\mathbf{t}_u,\mathbf{t}\},\ \text{and}$$

$$\exists\ell_j.\overrightarrow{\bigvee_{\rho^\omega}}\pi_{G'_j}(S_{\ell_j})=\mathbf{t}$$

$\equiv$ $\left\{\text{definition of }\overrightarrow{\bigvee_{\rho}}\right\}$

$$\forall i.\overrightarrow{\bigvee_{\rho^\omega}}\pi_{G'_j}(S_i)\in\{\mathbf{u},\mathbf{t}_u,\mathbf{t}\},\ \text{and}$$

$$\exists\ell_j.\pi_{G'_j}(S_{\ell_j})\neq\lambda\ \text{with no divergent atoms.}$$

Hence, for each $j\in\mathscr{J}$ there exists ℓ_j such that

(a) either $\pi_{G'_j}(S_{\ell_j})=\lambda$,

(b) or $\pi_{G'_j}(S_{\ell_j})\neq\lambda$ and does not contain divergent atoms.

Moreover, at least one such j satisfies (b). Let then $\ell=\max\{\ell_j\mid j\in\mathscr{J}\}$. By Lemma 33, it is clear that $\pi_{p(g)}(S_{\ell+1})\neq\lambda$ and does not contain divergent atoms. Hence $\rho^{\ell+1}(p(g))=\mathbf{t}$. By this and Lemma 42 we can conclude that $\rho^\omega(p(g))=\mathbf{t}$.

(iv) Finally, assume $\bigvee_{j\in\mathscr{J}} v_{\rho^\omega}(\exists y.g=t_j\wedge G_j)=\mathbf{t}_u$. Assume now that the thesis does not hold, i.e.

(a) either $\rho^\omega(p(g))=\mathbf{f}$

(b) or, $\rho^\omega(p(g))=\mathbf{u}$.

In case (a), by definition of ρ^ω, there exists $\ell\geqslant 1$ such that

(a.1) for each $k\geqslant\ell$, $\pi_{p(g)}(S_k)=\lambda$ and $\rho^k(p(g))=\mathbf{f}$

(a.2) for each k such that $0<k<\ell$, $\pi_{p(g)}(S_k)=\widehat{p(g)}::S'$ and $\rho^k(p(g))=\mathbf{u}$.

For $j\in\mathscr{J}$ let, as in the previous cases, $G'_j=G_j\theta_j$, where $\theta_j=mgu(g,t_j)$.

For each $k\geqslant\ell$ and for each $j\in\mathscr{J}$, we have

$$\pi_{p(g)}(S_k)=\lambda$$

$\Rightarrow$ {Lemma 33}

$$\pi_{G'_j}(S_{k-1})=\lambda$$

$\Rightarrow$ $\left\{\text{definition of }\overrightarrow{\bigvee_{\rho}}\right\}$

$$\overrightarrow{\bigvee_{\rho^\omega}}\pi_{G'_j}(S_{k-1})=\mathbf{f}$$

$\Rightarrow$ {definition of $\max^f$}

$$\max^f\left\{\overrightarrow{\bigvee_{\rho^\omega}}\pi_{G'_j}(S_k)\,\middle|\,S_k\in\mathscr{S}_{DFL}(P)\right\}=\mathbf{f}$$

$\equiv$ {definition of v_ρ and definition of G'_j}

$$v_{\rho^\omega}(\exists z_j.g=t_j\wedge G_j)=\mathbf{f}.$$

Hence, we also have that

$$\bigvee_{j\in\mathscr{J}} v_{\rho^\omega}(\exists y.g = t_j \wedge G_j) \in \{\mathbf{u},\mathbf{f}\}$$

which is a contradiction with the assumption $\bigvee_{j\in\mathscr{J}} v_{\rho^\omega}(\exists y.g = t_j \wedge G_j) = \mathbf{t}_u$.

In case (b), by definition of ρ^ω, for each k we have that $\pi_{p(g)}(S_k) = \widehat{p(g)} :: S'$ and $\rho^k(p(g)) = \mathbf{u}$. For each $k \geqslant 1$, we have

$$\begin{array}{ll} & \pi_{p(g)}(S_k) = \widehat{p(g)} :: S' \\ \Rightarrow & \{\text{Lemma 33}\} \\ & \pi_{G'_{j_1}}(S_{k-1}) = \widehat{d} :: S'' \\ \Rightarrow & \{\text{definition of } \vec{\bigvee}_{\rho}\} \\ & \vec{\bigvee}_{\rho^\omega} \pi_{G'_{j_1}}(S_{k-1}) = \mathbf{u} \end{array}$$

By observing that also $\vec{\bigvee}_{\rho^\omega} \pi_{G'_{j_1}}(S_0) = \mathbf{u}$, we have

$$\begin{array}{ll} & \max^f \left\{ \vec{\bigvee}_{\rho^\omega} \pi_{G'_{j_1}}(S_k) \,\middle|\, S_k \in \mathscr{S}_{DFL}(P) \right\} = \mathbf{u} \\ \equiv & \{\text{definition of } v_\rho \text{ and definition of } G'_{j_1}\} \\ & v_{\rho^\omega}(\exists z_j.g = t_{j_1} \wedge G_{j_1}) = \mathbf{u} \\ \Rightarrow & \{\text{definition of } \vee\} \\ & \bigvee_{j\in\mathscr{J}} v_{\rho^\omega}(\exists y.g = t_j \wedge G_j) = \mathbf{u} \end{array}$$

which is a contradiction with the assumption $\bigvee_{j\in\mathscr{J}} v_{\rho^\omega}(\exists y.g = t_j \wedge G_j) = \mathbf{t}_u$. □

Lemma 46. *For each k, $\rho^k \leqslant_\kappa \mathscr{T}_P(\rho^\omega)$.*

Proof. Analogous to the proof of Lemma 45. □

Corollary 47. $\rho^\omega \leqslant_\kappa \mathscr{T}_P(\rho^\omega)$

Lemma 48. *ρ^ω is a fixpoint of $\mathscr{T}_P$.*

Proof. Immediate consequence of Lemma 45 and Corollary 47. □

Since $\mathscr{T}_P^\omega$ is the least fixpoint of $\mathscr{T}_P$ we have the obvious corollary.

Corollary 49. $\mathscr{T}_P^\omega \leqslant_\kappa \rho^\omega$.

In order to show that the converse of the last statement holds as well, we need first some technical lemmas.

Lemma 50. *Let $p(g)$ be a ground atom and $p(x) \leftrightarrow G$ the definition of p in s_comp(P). Then*

$$\rho^{k+1}(p(g)) = \overrightarrow{\bigvee_{\rho^k}} \pi_{G[x:=g]}(S_k).$$

Proof. The proof is an easy case analysis on the value of $\rho^{k+1}(p(g))$, exploiting Definition 35 and Lemma 33. □

Lemma 51. $\rho^k \leqslant_\kappa \mathscr{T}_P^\omega$.

Proof. By induction on k.

Base step: Obvious.

Inductive step: Let $p(g)$ be a ground atom and let $p(x) \leftrightarrow G$ be the definition of p in *s_comp*(P). Then, by Lemma 50, $\rho^{k+1}(p(g)) = \overrightarrow{\bigvee_{\rho^k}} \pi_{G[x:=g]}(S_k)$, which, by exploiting the inductive hypothesis, is in the relation $\leqslant_\kappa$ with $\overrightarrow{\bigvee_{\mathscr{T}_P^\omega}} \pi_{G[x:=g]}(S_k)$. Indeed, clearly this is itself in the relation $\leqslant_\kappa$ with $\max^f\{\overrightarrow{\bigvee_{\mathscr{T}_P^\omega}} \pi_{G[x:=g]}(S_i) \mid S_i \in \mathscr{S}_{DFL}(P)\}$, which is nothing but $v_{\mathscr{T}_P^\omega}(G[x := g])$. This, by definition of $\mathscr{T}_P$ and the fact that $\mathscr{T}_P^\omega$ is a fixpoint of $\mathscr{T}_P$ allows us to conclude the proof. □

Corollary 52. *ρ^ω and $\mathscr{T}_\rho^\omega$ coincide.*

Proof of Theorem 26. For ground atoms, the theorem is a consequence of Proposition 19 and Corollary 52.

Let now G' be a (possibly non-ground) conjunction of atoms and consider the closed goal $G = \exists x.G'$. It is clear that the truth value of G in $\mathscr{T}_P^\omega$ coincides with the truth value of the propositional symbol ans_G in $\mathscr{T}_{P'}^\omega$, where P' is obtained by adding the clause $ans_G \leftarrow G'$ to P. Hence the theorem. □

References

[1] J.H. Andrews, The logical structure of sequential Prolog, in: S. Debray, M. Hermenegildo (Eds.), Proc. 1990 North American Conf. on Logic Programming, MIT Press, Cambridge, MA, 1990, pp. 585–602.

[2] J.H. Andrews, A logical semantics for depth-first Prolog with ground negation, in: D. Miller (Ed.), Proc. 1993 Internat. Symp. on Logic Programming, MIT Press, Cambridge, MA, 1993, pp. 220–234.

[3] K.R. Apt, Introduction to logic programming, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics, Elsevier, Amsterdam and MIT Press, Cambridge, MA, 1990, pp. 495–574.

[4] B. Arbab, D.M. Berry, Operational and denotational semantics of Prolog, J. Logic Programming 4 (1987) 309–330.

[5] R. Barbuti, M. Codish, R. Giacobazzi, G. Levi, Modelling Prolog control, J. Logic Comput. 3 (1993) 579–603.

[6] R. Barbuti, M. Codish, R. Giacobazzi, M. Maher, Oracle semantics for Prolog, Inform. and Comput. 122 (1995) 178–200.

[7] R. Barbuti, P. Mancarella, A multiple valued logical semantics for Prolog, in: Proc. ESOP 96, April 1996.
[8] E. Börger, D. Rosenzweig, A mathematical definition of full Prolog, Science of Computer Programming 24 (3) (1995) 249–286.
[9] A. Bossi, M. Bugliesi, M. Fabris, A new fixpoint semantics for Prolog, in: Proc. of the 10th Internat. Conf. on Logic Programming, MIT Press, Cambridge, MA, 1993, pp. 374–389.
[10] A. de Bruin, E. de Vink, Continuation semantics for Prolog with cut, in: J. Diaz, F. Orejas (Eds.), Proc. CAAP 89, Lecture Notes in Computer Science, Vol. 351, Springer, Berlin, 1989, pp. 178–192.
[11] K.L. Clark, Negation as failure, in: Logic and Databases, Plenum Press, New York, 1978, pp. 293–322.
[12] S.K. Debray, P. Mishra, Denotational and operational semantics for Prolog, in: M. Wirsing (Ed.), Formal Description of Programming Concepts III, North-Holland, Amsterdam, 1987, pp. 245–269.
[13] G. Epstein, The lattice theory of Post algebras, in: D.C. Rine (Ed.), Computer Science and Multiple-valued Logic, North-Holland, Amsterdam, 1984, pp. 23–40. Reprinted from Trans. Amer. Math. Soc. 95 (2) (1960) 300–317.
[14] M. Fitting, A Kripke–Kleene semantics for logic programs, J. Logic Programming 4 (1985) 295–312.
[15] M. Fitting, Bilattices and the semantics of logic programming, J. Logic Programming 11 (1991) 91–116.
[16] N.D. Jones, A. Mycroft, Stepwise development of operational and denotational semantics for Prolog, in: Sten-Åke Tärnlund (Ed.), Proc. Second Internat. Conf. on Logic Programming, 1984, pp. 281–288.
[17] K. Kunen, Negation in logic programming, J. Logic Programming 4 (1987) 298–308.
[18] J.W. Lloyd, Foundations of Logic Programming, 2nd Ed., Springer, Berlin, 1987.
[19] D. Miller, G. Nadathur, F. Pfenning, A. Scedrow, Uniform proofs as a foundation for logic programming, Ann. Pure Appl. Logic 51 (1991) 125–157.
[20] A. Mycroft, Logic programs and multi-valued logic, in: M. Fontet, K. Mehlhorn (Eds.), Proc. STACS 84, Lecture Notes in Computer Science, Vol. 166, Springer, Berlin, 1984, pp. 274–286.
[21] M. Rayan, M. Sadler, Valuation systems and consequence relations, in: S. Abramsky, D.M. Gabbay, T.S.E. Maibaum (Eds.), Handbook of Logic in Computer Science, Vol. I, Clarendon Press, Oxford, 1992, pp. 1–78.
[22] W.R. Smith, Minimization of multivalued functions, in: D.C. Rine (Ed.), Computer Science and Multiple-valued Logic, North-Holland, Amsterdam, 1984, pp. 227–267.
[23] Special Section on Multiple-valued Logic, IEEE Trans. Comput. C-30 (1981) 617–706.