A Multiple-Valued Logical Semantics for Prolog

Roberto Barbuti and Paolo Mancarella *

Dipartimento di Informatica, Università di Pisa, 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 new semantics for Prolog programs based on a new 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 minimal Herbrand model, equivalence of the model-theoretic and operational semantics.

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 minimal 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 minimal 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, 5, 6, 7, 8, 9, 17].

When dealing with computational issues, one has to abandon classical 2-valued logic and has to move to multiple-valued logic. A first attempt is to adopt a 3-valued logic where the third truth value (*undefined*) is introduced to model non-terminating computations (see, e.g. [1, 2, 14, 16, 19]). It is worth noting that the 3-valued approach has been successfully used to give a semantics also to logic programming with negation. However, these 3-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 $\{p,\ p \leftarrow p\}$.

* Work partially supported by the EEC *Keep in Touch* activity KIT011 - LPKRR.

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 replaced by two new connectives $\wedge$ (sequential conjunction) and $\vee$ (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-most 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 minimal Herbrand model, equivalence of the model-theoretic and operational semantics).

For the sake of clarity, in this paper we first explore our approach for propositional Prolog, and then we extend it to full, pure Prolog. The main difficulty in moving from propositional to pure Prolog stands in the semantics of the existential quantification. In this respect, we introduce the concept of *sequential* existential quantification and we give its meaning in our four-valued logic.

2 Preliminaries

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

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

We refer to a first-order language $\mathcal{L}$ with predicate symbols $\mathcal{P}$, variables $\mathcal{V}$ and function symbols $\mathcal{F}$. The ground term algebra over $\mathcal{F}$ is denoted by $T(\mathcal{F})$. The non-ground term algebra over $\mathcal{F}$ and $\mathcal{V}$ is denoted by $T(\mathcal{F}, \mathcal{V})$. The set of atoms constructed from predicate symbols in $\mathcal{P}$ and terms from $T(\mathcal{F}, \mathcal{V})$ is denoted $Am(\mathcal{P}, \mathcal{F}, \mathcal{V})$ or Am for short.

A *valuation system* for a language $\mathcal{L}$ is tuple $\langle \mathcal{T}, \mathcal{D}, \mathcal{R}, \mathcal{G} \rangle$ where: i) $\mathcal{T}$ is the set of truth values, with at least two elements; ii) $\mathcal{D}$ is the set of *designated* truth values, a non-empty proper subset of $\mathcal{T}$; iii) $\mathcal{R}$ is a set of functions interpreting the connectives of the logic; iv) $\mathcal{G}$ is a set of functions interpreting the quantifiers of the logic.

Note that using valuation systems, we can assign different meanings to connectives and quantifiers.

An *assignment* α relative to a valuation system $\langle \mathcal{T}, \mathcal{D}, \mathcal{R}, \mathcal{G} \rangle$ is a pair $\alpha = \langle \rho, I \rangle$, where I is a non-empty set of *individuals* and ρ is a function which maps variables and ground terms to elements of I, and each predicate symbol p to a function $I^n \longrightarrow \mathcal{T}$, where n is the arity of p.

Each assignment α induces an *interpretation* (or *valuation*) v_α of a sentence in the language, inductively defined on the structure of the sentence and based on $\mathcal{R}$ and $\mathcal{G}$ as far as the interpretation of connectives and quantifiers is concerned. An interpretation v_α is a *model* for a sentence ϕ iff $v_\alpha(\phi) \in \mathcal{D}$. Given two formulae ϕ, ϕ' we say that ϕ' is a *logical consequence* of ϕ, denoted by $\phi \models \phi'$, iff $v_\alpha(\phi') \in \mathcal{D}$, for all models v_α of ϕ.

As an example, classical first order logic is a multiple valued logic in which $\mathcal{T} = \{\mathbf{t}, \mathbf{f}\}$, $\mathcal{D} = \{\mathbf{t}\}$, and the connectives and quantifiers are interpreted as usual.

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].

Given a first order language $\mathcal{L}$, a logic program over $\mathcal{L}$ is a set of definite clauses of the form $A \leftarrow \mathbf{B}$ where A is an atom and $\mathbf{B}$ is a conjunction $B_1 \wedge \ldots \wedge B_n$ of atoms. A is called the *head* of the clause, and $\mathbf{B}$ is called the *body* of the clause. All the variables occurring in a clause are implicitly universally quantified. A *goal* is a clause with an empty head, denoted by $\leftarrow \mathbf{B}$.

The declarative semantics of logic programs is given by classical two-valued logic and *Herbrand interpretations*. A *Herbrand interpretation* is a valuation v_α corresponding to an assignment $\langle \rho, T(\mathcal{F}) \rangle$, in which the domain of individuals is the ground term algebra (Herbrand Universe). The standard semantics of a logic program P is given by its *minimal* Herbrand model, with respect to the pointwise ordering on interpretations induced by the ordering $\mathbf{f} < \mathbf{t}$ on truth values.

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 via a *computation rule*. Then, for each clause $A \leftarrow \mathbf{B}$ such that $mgu(A, B_m) = \vartheta \neq \mathit{fail}$, the node has child $(B_1, B_2, \ldots, \mathbf{B}, \ldots, B_m)\vartheta$, 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 a 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: i) the computation rule is the left-most one; ii) 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 left-most depth-first visit of the Prolog-tree for a given goal.

Our semantics is based on a notion of *completion* of a logic program which is similar to Clark's completion [11]. The latter has been 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.

Let P be a logic program. Each clause $p(t_1, t_2, \ldots, t_k) \leftarrow \mathbf{B}$ is first transformed into

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

where $x_1, \ldots, x_k$ are new variables and $\overline{y}$ is the sequence of variables occurring in the original clause. Let then $p(\overline{x}) \leftarrow \mathbf{E}_i$, $0 \leq i \leq m$ be the sequence of the transformed clauses m clauses of P defining p. The *completed definition* of p is the formula $p(\overline{x}) \leftrightarrow \mathbf{E}_1 \vee \ldots \vee \mathbf{E}_m$ where $\mathbf{E}_i$ coincides with $\mathbf{B}_i$ if $\mathbf{B}_i$ is a non empty conjunction, and E_i is *true* otherwise. If a predicate p never occurs in the head of a clause in P, its *completed definition* is the formula $p(\overline{x}) \leftrightarrow$ *false*. The *completion* of a Prolog program P, denoted by $comp(P)$ is the collection of the completed definitions of the predicates in $p \in \mathcal{P}$, equipped with the axioms of Clark's equality theory which force $=$ to be interpreted as syntactic identity.

3 Related works

There are several papers dealing with a logical description of the Prolog semantics.

In [10], a translation from a propositional Prolog program P to a linear logic theory $\mathbf{LT}_P$ is given, and the soundness and completeness of the translation is proved: the goal A succeeds under the Prolog evaluation iff $\mathbf{LT}_P \vdash_{lin} A$, and A fails iff $\mathbf{LT}_P \vdash_{lin} A^{\perp}$.

A similar approach, using first order logic, is developed in [24]. A new completion of programs, ℓ–completion is defined. The ℓ–completion is a theory in a language extended by new predicates which expresses the success, failure and termination of goals. If the Prolog evaluation of a goal succeeds, fails or terminates, then the corresponding formulas, in the extended language, are provable from the ℓ–completion. Conversely, if it is provable, in the ℓ–completion, that a goal succeeds and terminates, then the goal has a successful Prolog evaluation.

These approaches, although able to logically describe the successes and failures of Prolog computation, do not capture an important aspect of computations, that is non-termination. To model infinite computations a new value is used, namely the *undefined* value **u**. The introduction of this new value leads to the use of a multiple-valued logic as a tool for describing the operational behaviour of logic programs.

Examples of semantic definitions based on three-valued logic are the ones of [14, 16, 19]. These logical semantics are defined for pure logic programming, that is they model an operational behaviour based on fair SLD-trees visited by a breadth-first strategy.

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-most depth-first visit of a Prolog tree. The left-most 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 [20]. A *goal* is a formula defined as follows: (i) a truth value **f** or **t**, or (ii) an atom, or (iii) $s = t$, where s, t are possibly non ground terms, or (iv) $G \wedge G'$, $G \vee G'$, $\exists x.G$, where G and G' are goals.

As an example consider the Prolog program $p(\overline{t}) \leftarrow q(\overline{t'})$. $p(\overline{s}) \leftarrow r(\overline{s'})$. where $\overline{t}, \overline{t'}, \overline{s}, \overline{s'}$ are sequences of terms. Its completion is given by

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

where $\overline{y}, \overline{z}$ are the sequences of variables occurring in the two clauses respectively, and $\overline{x = t}$ is an abuse of notation to indicate all the equations having in the left part a variable in $\overline{x}$ and in the right part the corresponding term in $\overline{t}$.

The semantics of Prolog is obtained by giving an order to the evaluation of $\vee$, so that, when evaluating the definition of $p(\overline{x})$, the part corresponding to the first clause is examined first, and the second part is evaluated only if the evaluation of the first one yields **f**. The evaluation order of $\wedge$ is defined analogously. Notice that the evaluation order of $\vee$ models the sequential use of the clauses, while the one of $\wedge$ models the left-most computation rule.

More formally [2] gives the semantics in terms of the three truth values, $\{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$[2] with the interpretation of $\vee$ and $\wedge$ given by the following *valuation functionv* mapping ground formulas on 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 fails to provide a complete logical specification of Prolog semantics.

Consider the following propositional Prolog program (in completed form)

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

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 **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 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

[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.

works properly 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 a O-formula, that is a formula in which all disjunctions are the immediate subformulas of either a negation or other disjunctions. Then, the valuation function is applied to these formulas to get the correct truth value. In the above example, $(q \wedge r) \vee (loop \wedge r)$ is a 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 true 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 true 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 [19], in which a value corresponding to $\mathbf{t}_u$ was used to model an SLD-tree with both infinite and success branches. 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.

A propositional logic program is a collection of definite clauses from a language $\mathcal{L}$ in which $\mathcal{F} = \emptyset$.

4.1 A four-valued logic

In this section we introduce the valuation system $\mathcal{V}_4$ 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 behavior 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 of Clark's completion.

Definition 1. The valuation system $\mathcal{V}_4 = \langle \mathcal{T}_4, \mathcal{D}_4, \mathcal{R}_4 \rangle$ is defined as follows:

(*i*) $\mathcal{T}_4 = \{\mathbf{f}, \mathbf{u}, \mathbf{t}_u, \mathbf{t}\}$

(*ii*) $\mathcal{D}_4 = \{\mathbf{t}\}$

(*iii*) The constants *false* and *true* have meaning **f** and **t** respectively, while the meaning of the connectives $\wedge, \vee$ and $\leftrightarrow$ is given by the following functions belonging to $\mathcal{R}_4$.

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

$f_\wedge(x, y)$

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

$f_\vee(x, y)$

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

Let us explain intuitively the above definitions. The interpretation of $\wedge$ mimics the computation corresponding to a conjunction of goals. Since **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 **f** or **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 **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**.

4.2 Semantics of Propositional Prolog

The semantics of a propositional Prolog program P is given in terms of its *sequential completion* $s\_comp(P)$, which is similar to Clark's completion. The only difference is that, when constructing the sequential completion of a program P, the textual order in which atoms occurs 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 order in which their transformations occur in the disjunction of conjunctions which form the completed definition of p. Due to its similarity with the definition of Clark's completion, we omit the definition of $s\_comp(P)$.

Of course, we have to give also a new definition of *goal*.

Definition 2. A *goal* is a formula defined as follows: (i) the constant *false* or the constant *true*, or (ii) an atom, or (iii) $G \wedge G'$, $G \vee G'$, where G and G' are goals.

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 $\mathcal{V} = \langle \mathcal{T}, \mathcal{D}, \mathcal{F} \rangle$, is an interpretation v_ρ, where ρ is an assignment, which assigns to the formula a truth value in $\mathcal{D}$ (the truth value $\mathbf{t}$ in the case of our valuation system $\mathcal{V}_4$).

Proposition 3. *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 formulas $(G \vee G') \wedge G''$ and $(G \wedge G'') \vee (G' \wedge G'')$ are equivalent, i.e. they have the same truth value in every interpretation.

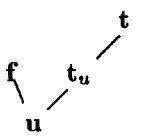
Let us consider the sequential completion of the example of Section 3.

$p \leftrightarrow q \vee \mathit{loop}.$ $\quad \mathit{loop} \leftrightarrow \mathit{loop}.$ $\quad q \leftrightarrow \mathit{true}.$ $\quad r \leftrightarrow \mathit{false}.$

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

It is important to remark that the classical properties of $\vee$ and $\wedge$ are not preserved in our valuation system. For example, the formulas $G \wedge (G' \vee G'')$ 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 *minimal* model of *s_comp*(P) with respect to a suitable ordering between 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 refer to this ordering as the *knowledge ordering*, $\leq_k$.

Definition 4. Given a propositional language $\mathcal{L} = \langle \mathcal{P}, \mathcal{O} \rangle$, let $\rho, \rho' : \mathcal{P} \to \mathcal{D}_4$ be two assignments relative to the valuation system $\mathcal{V}_4$. We say that ρ is less than or equal than ρ', denoted by $\rho \leq_k \rho'$, iff for all atoms $p \in \mathcal{P}$ we have $\rho(p) \leq_k \rho'(p)$.

It is easy to see that the set of assignments relative to $\mathcal{V}_4$ is a complete partial order with respect to $\leq_k$.

Proposition 5. *Given a propositional Prolog program P, the set of all models of s_comp(P) has a minimal element with respect to in $\leq_k$.*

[3] Consider the values $\mathbf{t}_u$, $\mathbf{f}$ and $\mathbf{t}$ 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 $\mathbf{u}$.

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

Definition 6. Let P be a propositional Prolog program and $\mathcal{L} = \langle \mathcal{P}, \mathcal{O} \rangle$ be language of its sequential completion. The operator $\mathcal{T}_P$ mapping assignments in $\mathcal{A}$ to assignments in $\mathcal{A}$ is defined as follows. For each predicate symbol $p \in \mathcal{P}$ $\mathcal{T}_P(\rho)(p) = v_\rho(\phi)$ 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 7. *Let P be a propositional Prolog program, and $\mathcal{T}_P$ be the operator associated with P. Then the following facts hold: i) $\mathcal{T}_P$ is monotonic with respect to $<_k$, and ii) a valuation v_ρ is a model of $s\_comp(P)$ iff ρ is a fixpoint of $\mathcal{T}_P$.*

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

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

Theorem 8. Let P be a propositional Prolog program, $\mathcal{M}$ the minimal model of $s\_comp(P)$ and G a goal:

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

□

5 Pure Prolog

In this section we extend the logical semantics to pure Prolog. Without loss of generality, we consider only Prolog programs in which all the clause heads are pairwise not unifiable. Given a program in which this property does not hold, it is easy to construct a new program, by adding an extra argument to any predicate, which has the property. For example, the program $p(f(x)) \leftarrow p(x),\ p(x) \leftarrow q(x)$ would be transformed into $p(1, f(x)) \leftarrow p(y, x),\ p(2, x) \leftarrow q(y, x)$.

As a consequence of the introduction of variables and terms, we have to give a new definition of goal. In the following, for the sake of brevity, we will denote a conjunction of equations $s_1 = t_1 \wedge s_2 = t_2 \wedge \ldots \wedge s_k = t_k$ simply by $\overline{s = t}$

Definition 9. A *goal* is a formula defined as follows: (i) a constant *false* or *true*, or (ii) an atom, or (iii) a conjunction of equations $\overline{s = t}$, or (iv) $G \wedge G'$, $G \vee G'$, $\exists x.G$, where G and G' are goals.

Note that the axioms of the Clark's equality theory force $=$ to be interpreted as syntactic identity also in the four-valued logic.

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 get an intuition of this let us consider the Prolog program

$p(b) \leftarrow p(b) \qquad p(a)$

on a language in which $\mathcal{F} = \{a, b\}$ and $\mathcal{P} = \{p\}$. Its sequential completion is given by

$p(x) \leftrightarrow (x = b \wedge p(b)) \vee x = a.$

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 $\mathbf{t}$ in the minimal model of the program. Note that the program could be viewed as a propositional one, thus, in its minimal model the atom $p(b)$ would have value $\mathbf{u}$, while $p(a)$ would have value $\mathbf{t}$.

On the other hand, the meaning of $\exists x.p(x)$ can be neither given 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. Essentially, this order is 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.

We consider a denumerable set of indexed variables, $\mathcal{V}^{seq} = \{x_{\overline{n}} \mid \overline{n} \text{ is a sequence of binary digits}\}$. We will denote by $\overline{n} :: m$ the sequence composed by the sequence $\overline{n}$ followed by the digit m. We assume that $\mathcal{V} \cap \mathcal{V}^{seq} = \emptyset$, where $\mathcal{V}$ is the set of variables in the underlying language.

Definition 10. Given the denumerable set $\mathcal{V}^{seq}$ of variables, a *total variable assignment* (a total assignment for short) is a total function $s : \mathcal{V}^{seq} \mapsto T(\mathcal{F})$.

A total assignment is an assignment of ground terms in the language $\mathcal{L}$ to all the variables in $\mathcal{V}^{seq}$. Total assignments are used to give a meaning to existentially quantified goals. The order among total assignments is based on the following definitions.

Definition 11. Given a goal G in the first-order language $\mathcal{L} = (\mathcal{F}, \mathcal{P}, \mathcal{V} \cup \mathcal{V}^{seq})$, we say that G is *$\mathcal{V}$-closed* iff all the free variables occurring in G belong to $\mathcal{V}^{seq}$.

Definition 12. Given a *$\mathcal{V}$-closed* goal G and a total assignment s, we denote by $G[s]$ the closed goal obtained from G by replacing variables in $\mathcal{V}^{seq}$ with corresponding terms in s.

Definition 13. Given a Prolog program P, two total assignments s and s', a sequence $\overline{n}$ of binary digits, and a $\mathcal{V}$-*closed* goal G; we say that s *must occur before* s' in G with variables starting from $\overline{n}$, denoted as $s \prec_G^{\overline{n}} s'$, iff one of the following holds:

- G is an atom $p(\overline{t})$, $p(\overline{x}) \leftrightarrow G'$ is in $s\_comp(P)$ and $s \prec^{\overline{n}}_{G'[\overline{x}:=\overline{t}]} s'$.
- G is $\overline{t = t'}$ and both $CET \models (\overline{t = t'})[s]$ and $CET \not\models (\overline{t = t'})[s']$.
- G is $G_0 \wedge G_1$ and either $s \prec^{\overline{n}::0}_{G_0} s'$ or both $s \not\prec^{\overline{n}::0}_{G_0} s'$, $s' \not\prec^{\overline{n}::0}_{G_0} s$, and $s \prec^{\overline{n}::1}_{G_1} s'$ hold.
- G is $G_0 \vee G_1$ and either $s \prec^{\overline{n}::0}_{G_0} s'$ or both $s \not\prec^{\overline{n}::0}_{G_0} s'$, $s' \not\prec^{\overline{n}::0}_{G_0} s$, and $s \prec^{\overline{n}::1}_{G_1} s'$ hold.
- G is $\exists x.G_0$ and $s \prec^{\overline{n}::0}_{G_0[x:=x_{\overline{n}::0}]} s'$, $x \in \mathcal{V}$ and $x_{\overline{n}::0} \in \mathcal{V}^{seq}$.

Intuitively, a total assignment s must occur before a total assignment s' iff in the left-most inner-most visit of the definition of G a set of equations is found which is satisfied by s and not by s'. Note that in the visit, each encountered existential variable belonging to $\mathcal{V}$ is substituted by a free fresh one from $\mathcal{V}^{seq}$. Thinking operationally, s corresponds to the bindings of a branch in the Prolog-tree for the goal G which is on the left of a branch (if any) s' corresponds to. Note that we assume the clause heads pairwise not unifiable, thus the bindings corresponding to different branches are different.

Proposition 14. *Let P be a Prolog program, G a goal, $\mathcal{A}$ the set of all total assignments and λ the empty sequence of binary digits. The relation $\prec_G^{(\lambda)}$ between assignments is a non-reflexive ordering relation on $\mathcal{A}$.*

Proof. Based on the fact that no clause heads are unifiable.

In the following, we will denote the ordered set by $(\mathcal{A}, \prec_G^{(\lambda)})$. Note that, in $(\mathcal{A}, \prec_G^{(\lambda)})$ we can have infinite chains whose limits do not belong to $\mathcal{A}$. This because a limit of a chain can be an assignment which substitutes infinite terms for variables. To obtain the truth value of a Prolog goal, we need to "close" $(\mathcal{A}, \prec_G^{(\lambda)})$[4] by extending it with the limits of the chains.

Definition 15. Let $(\mathcal{A}, \prec_G^{(\lambda)})$ be an ordered set of all assignments, $T^{\infty}(\mathcal{F})$ the set of infinite terms built from symbols in $\mathcal{F}$, and $\mathcal{A}^{\infty}$ the set of total assignments $s : \mathcal{V}^{seq} \mapsto T(\mathcal{F}) \cup T^{\infty}(\mathcal{F})$. We define $\overline{\mathcal{A}}$ as the set

$\mathcal{A} \cup \{s | s \in \mathcal{A}^{\infty}$ such that s is the limit of an infinite chain in $(\mathcal{A}, \prec_G^{(\lambda)})\}$

In the following we will denote an ordered *closed* set of all the assignments as $(\overline{\mathcal{A}}, \prec_G^{(\lambda)})$.

Given a goal G, we can obtain its truth value with respect to the ordered closed set of all the assignments, $(\overline{\mathcal{A}}, \prec_G^{(\lambda)})$.

[4] In the sense of applying to it a closure operation.

Definition 16. Given a $\mathcal{V}$-*closed* goal G', and a set of assignments $\mathcal{B}$ ordered by $\prec$, we denote the sequential disjunction of G' with respect to $(\mathcal{B}, \prec)$,

$$\vec{\bigvee_{s \in (\mathcal{B}, \prec)}} G'[s]$$

as a possible sequential disjunction of all $G'[s]$'s such that $G'[s]$ occurs in the disjunction before $G'[s']$ if $s \prec s'$ $(s, s' \in \mathcal{B})$.

Definition 17. Given a set of total assignments $\mathcal{B}$, we define the subset $\mathcal{B}|_{x_{\overline{n}}:=t}$ of $\mathcal{B}$, where $x_{\overline{n}} \in \mathcal{V}^{seq}$, $t \in T(\mathcal{F}) \cup T^{\infty}(\mathcal{F})$, as the following set: $\mathcal{B}|_{x_{\overline{n}}:=t} = \{s | s \in \mathcal{B},\ s(x_{\overline{n}}) = t\}$

Definition 18. Given a closed goal G with terms from $T(\mathcal{F}) \cup T^{\infty}(\mathcal{F})$, an ordered set of total assignments $(\mathcal{B}, \prec)$, a sequence of binary digits $\overline{n}$, and an interpretation I, the truth value of G, with variables starting from $\overline{n}$ with respect to $(\mathcal{B}, \prec)$, in I, denoted by $\left(G^{(\overline{n})}_{(\mathcal{B},\prec)}\right)_{(I)}$ is the truth value obtained as follows:

- **f** if G is *false*,
- **t** if G is *true*,
- **u** if G is $t = t'$ and there are infinite terms occurring in it, or the truth value of $(t = t')$ in I, if all the occurring terms are finite,
- **u** if G is $p(\bar{t})$ and there are infinite terms occurring in it, or the truth value of $p(\bar{t})$ in I, if all the occurring terms are finite,
- the value of $\left(G_0{}^{(\overline{n}::0)}_{(\mathcal{B},\prec)}\right)_{(I)} \vee \left(G_1{}^{(\overline{n}::1)}_{(\mathcal{B},\prec)}\right)_{(I)}$ if G is the goal $G_0 \vee G_1$,
- the value of $\left(G_0{}^{(\overline{n}::0)}_{(\mathcal{B},\prec)}\right)_{(I)} \wedge \left(G_1{}^{(\overline{n}::1)}_{(\mathcal{B},\prec)}\right)_{(I)}$ if G is the goal $G_0 \wedge G_1$,
  the maximal value (with respect to the truth ordering relation $<_t$) of the possible sequential disjunctions

$$\vec{\bigvee_{s \in (\mathcal{B}, \prec)}} \left(G_0{}^{(\overline{n}::0)}_{(\mathcal{B}|_e,\prec)}[x := x_{\overline{n}::0}][s]\right)_{(I)}$$

  where e is the assignment $x_{\overline{n}::0} := s(x_{\overline{n}::0})$, if G is the goal $\exists x.G_0$.

When evaluating an existential goal, we replace the existential variable by a fresh one from the set $\mathcal{V}^{seq}$, indexed by a sequence longer than $\overline{n}$. The reason for substituting a fresh variable for the quantified one is to study the truth value of the formula by means of ground instantiations of the introduced fresh variable $x_{\overline{n}::0}$ (total assignments on $\mathcal{V}^{seq}$). Note that for each assignment s, the instantiated goal is evaluated with respect to the set of assignments which agree with s about the value of $x_{\overline{n}::0}$, $\mathcal{B}|_{x_{\overline{n}::0}:=s(x_{\overline{n}::0})}$.

Finally we can give the truth value of a goal.

Definition 19. Given a closed goal G, the closed ordered set of all the assignments $(\overline{\mathcal{A}}, \prec_G^{(\lambda)})$, and an interpretation I, the truth value of G is value obtained as:

$$\left(G^{(\lambda)}_{(\overline{\mathcal{A}}, \prec_G^{(\lambda)})}\right)_{(I)}$$

Recall that the ordering $\prec_G^{\lambda}$ is induced by the goal G and by the Prolog program P. Now we can define the notion of model for pure Prolog.

Definition 20. Given a Prolog program P and an interpretation I. I is a model for $s\_comp(P)$ iff for every definition

$p(\overline{x}) \leftrightarrow G$

in $s\_comp(P)$, the truth value of $\left(p(\overline{x})[\overline{x} := \overline{t}]\right)_{(I)}$ is the same as the truth value of $\left(G[\overline{x} := \overline{t}]\right)_{(I)}$ for each sequence $\overline{t}$ of terms in $T(\mathcal{F})$.

According to this notion of model, the results for propositional Prolog can be extended to pure Prolog. In particular, Theorem 8 can be rephrased for pure Prolog. Let us now give two examples.

Example 1. Consider the Prolog program

$p(s(x)) \leftarrow p(x). \qquad p(0).$

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

In the minimal 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 replacement of the existential variable from a fresh one in $\mathcal{V}^{seq}$ starting from the empty sequence λ produces $p(x_0)$, where 0 stands for the sequence containing the unique value 0. We have that $\{x_0 := s^{n+1}(0), \ldots\} \prec_G^{\lambda} \{x_0 := s^n(0), \ldots\}$, for all natural numbers n, and for all total assignments in $\mathcal{A}$. Thus, there is an infinite descending chain of total assignments

$\ldots \prec_G^{\lambda} \{x_0 := s^{n+1}(0), \ldots\} \prec_G^{\lambda} \{x_0 := s^n(0), \ldots\} \prec_G^{\lambda} \ldots$

$\ldots \prec_G^{\lambda} \{x_0 := s(0), \ldots\} \prec_G^{\lambda} \{x_0 := 0, \ldots\}$

whose limit in $\overline{\mathcal{A}}$ is the total assignment $\{x_0 := s^{\infty}(0), \ldots\}$. Thus, the value of

$$\overrightarrow{\bigvee_{s \in (\overline{\mathcal{A}}, \prec_G^{(\lambda)})}} (p(x_1)[s])_{(I)}$$

is $\mathbf{u}$, 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 without success branches.

Example 2. Consider the Prolog program

$p(0). \qquad p(s(x)) \leftarrow p(x).$

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

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

Consider now the goal $G = \exists x.p(x)$. We have that $\{x_0 := s^n(0), \ldots\} \prec_G^\lambda \{x_0 := s^{n+1}(0), \ldots\}$, for all natural numbers n. Thus, there is an infinite ascending chain of total assignments,

$\{x_0 := 0, \ldots\} \prec_G^\lambda \{x_0 := s(0), \ldots\} \prec_G^\lambda \{x_0 := s^2(0), \ldots\} \prec_G^\lambda \ldots$

whose limit is $\{x_0 := s^\infty(0), \ldots\}$. The replacement of the existential variable by a fresh one in $\mathcal{V}^{seq}$ starting from λ produces $p(x_0)$. The truth value of the goal is then obtained as the infinite sequential disjunction

$$\vec{\bigvee_{s \in (\overline{\mathcal{A}}, \prec_G^{(\lambda)})}} (p(x_1)[s])_{(I)}$$

This corresponds to the infinite disjunction $\mathbf{t} \vee \mathbf{t} \vee \ldots \vee \mathbf{u}$, thus the initial goal has value $\mathbf{t}_u$. This models the operational behaviour of the Prolog goal $p(x)$ which has infinite Prolog-tree with infinitely many success branches on the left of the infinite one.

6 Conclusions

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*. On the other hand, we plan to explore the possibility of further extending the approach to cope with other extra-logical features of Prolog.

Acknowledgments. We thank Paola Quaglia for helpful discussions on the subject of this work. We also thank the anonymous referees for their comments and suggestions.

References

1. J.H. Andrews. The Logical Structure of Sequential Prolog. In S. Debray and M. Hermenegildo, editors, *Proc. 1990 North American Conf. on Logic Programming*, 585–602. The MIT Press, Cambridge, Mass., 1990.
2. J.H. Andrews. A Logical Semantics for Depth-first Prolog with Ground Negation. In D. Miller, editor, *Proc. 1993 Int'l. Symp. on Logic Programming*, 220–234. The MIT Press, Cambridge, Mass., 1993.
3. K. R. Apt. Introduction to Logic Programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, 495–574. Elsevier, Amsterdam and The MIT Press, Cambridge, 1990.
4. B. Arbab and D.M. Berry. Operational and Denotational Semantics of Prolog. *Journal of Logic Programming*, 4:309–330, 1987.
5. R. Barbuti, M. Codish, R. Giacobazzi, and G. Levi. Modelling Prolog Control. *Journal of Logic and Computation*, 3:579–603, 1993.

6. R. Barbuti, M. Codish, R. Giacobazzi, and M. Maher. Oracle Semantics for Prolog. to appear in *Information and Computation.*
7. A. Bossi, M. Bugliesi, and M. Fabris. A New Fixpoint Semantics for Prolog. In *Proc. of the Tenth Int. Conference on Logic Programming*, 374–389. MIT Press, 1993.
8. E. Börger, and D. Rosenzweig. A Mathematical Definition of Full Prolog. *Science of Computer Programming*, 1994, (to appear).
9. A. de Bruin and E. de Vink. Continuation semantics for Prolog with cut. In J. Diaz and F. Orejas, editors, *Proc. CAAP 89*, volume 351 of *Lecture Notes in Computer Science*, 178–192. Springer-Verlag, Berlin, 1989.
10. D. Cerrito. A Linear Axiomatization of Negation as Failure. *Journal of Logic Programming*, 12:1–24, 1992.
11. K.L. Clark. Negation as Failure. In *Logic and Databases*, 293–322, Plenum Press, New York, 1978.
12. S. K. Debray and P. Mishra. Denotational and Operational Semantics for Prolog. In M. Wirsing, editor, *Formal Description of Programming Concepts III*, 245–269. North-Holland, Amsterdam, 1987.
13. Epstein G.. The Lattice Theory of Post Algebras. In D.C. Rine, editor, *Computer Science and Multiple-valued Logic*, 23–40. North-Holland, Amsterdam, 1984. Reprinted from *Transaction of the American Math. Society*, 95, 2:300–317, 1960.
14. M. Fitting. A Kripke-Kleene Semantics for Logic Programs. *Journal of Logic Programming*, 4:295–312, 1985.
15. M. Fitting. Bilattices and the semantics of logic programming. *Journal of Logic Programming*, 11:91–116, 1991.
16. K. Kunen Negation in Logic Programming. *Journal of Logic Programming*, 4:298–308, 1987.
17. N.D. Jones and A. Mycroft. Stepwise Development of Operational and Denotational Semantics for Prolog. In Sten-Åke Tärnlund, editor, *Proc. Second Int'l Conf. on Logic Programming*, 281–288, 1984.
18. J.W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, 1987. Second edition.
19. A. Mycroft. Logic Programs and Multy-valued Logic. In M. Fontet and K. Mehlhorn, editors, *Proc. STACS 84*, volume 166 of *Lecture Notes in Computer Science*, 274–286. Springer-Verlag, Berlin, 1984.
20. D. Miller, G. Nadathur, F. Pfenning, and A.Scedrow. Uniform Proofs as a Foundation for Logic Programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.
21. M. Rayan and M. Sadler. Valuation Systems and Consequence Relations. In S. Abramsky, D.M. Gabbay and T.S.E. Maibaum eds.,*Handbook of Logic in Computer Science*, Vol. I, Clarendon Press, Oxford, 1–78, 1992.
22. W.R. Smith. Minimization of Multivalued Functions. In D.C. Rine, editor, *Computer Science and Multiple-valued Logic*, 227–267. North-Holland, Amsterdam, 1984.
23. Special Section on Multiple-valued Logic. *IEEE Trans. on Computers*, C-30:617–706, 1981.
24. R.F. Stärk. The Declarative semantics of the prolog Selection Rule. In *Proc. Ninth IEEE Symp. on Logic in Computer Science*, Paris, 252–261, 1994.