Oracle Semantics for Prolog

Roberto Barbuti
Dipartimento di Informatica, Università di Pisa
Corso Italia 40, I-56125 Pisa, Italy
barbuti@di.unipi.it

Michael Codish
Department of Mathematics & Computer Science
Ben-Gurion University of the Negev
P.O. Box 653, Beer-Sheba, Israel
codish@bengus.bgu.ac.il

Roberto Giacobazzi
Dipartimento di Informatica, Università di Pisa
Corso Italia 40, I-56125 Pisa, Italy
giaco@di.unipi.it

and

Michael J. Maher
IBM T.J. Watson Research Center
P.O. Box 704, Yorktown Heights
NY 10598
mjm@watson.ibm.com

Abstract

This paper proposes to specify semantic definitions for logic programming languages such as Prolog in terms of an oracle which specifies the control strategy and identifies which clauses are to be applied to resolve a given goal. The approach is quite general. It is applicable to Prolog to specify both operational and declarative semantics as well as other logic programming languages.

Previous semantic definitions for Prolog typically encode the sequential depth-first search of the language into various mathematical frameworks. Such semantics mimic a Prolog interpreter in the sense that following the "leftmost" infinite path in the computation tree excludes computation to the right of this path from being considered by the semantics. The basic idea in this paper is to abstract away from the sequential control of Prolog and to provide a declarative characterization of the clauses to apply to a given goal. The decision whether or not to apply a clause is viewed as a query to an oracle which is specified from within the semantics and reasoned about from outside. This approach results in simple and concise semantic definitions which are more useful for arguing the correctness of program transformations and providing the basis for abstract interpretations than previous proposals.

Contents

1 Introduction

Logic programming is attractive for its non-algorithmic style and for its clean logical and fixpoint semantics. This leads to a high-level approach to programming, provides for the application of semantic-based techniques, such as abstract interpretation, and eases the justification of program transformations. On the other hand, real logic languages, such as Prolog, sacrifice logical purity to achieve acceptable efficiency. As a consequence the semantic simplicity and its advantages are often lost. Due to this dichotomy, logic programming faces two different paradigms: *logic* programming, in which the accent is placed on declarative specification of problems, and logic *programming*, in which the central issue is to write efficient programs to solve problems.

In this paper we weaken this dichotomy by applying the *logic* programming style to define a semantics for the logic *programming* language Prolog. The resulting semantics is more declarative than previous proposals. The central idea is perhaps best paraphrased as applying the equation $Prolog = Logic + Control$ at the semantic level. Our semantics is partitioned into two components. The 'logic' component which is similar to what is found in classic semantics of logic programs; and the 'control' component which specifies when a given clause may be applied to solve a query or sub-query. In many cases both components have a clean specification such as those given for logic programs. As a consequence many of the advantages of *logic* programming are regained.

Declarative semantics for logic programs specify the set of ground atoms which are implied by the program, often by means of a fixpoint construction. This construction abstracts away from operational aspects of the interpreter, for example answer substitutions, goal dependence and search strategy. Although declarative semantics can be specified in different frameworks, they are typically defined in terms of an *immediate consequence operator* such as the classic T_P operator [van Emden and Kowalski 76]. While this approach captures the logical meaning of a program it does not adequately model one of the fundamental aspects of a logic program, namely its ability to compute substitutions. The work in [Falaschi, Levi, Martelli and Palamidessi 89] and [Falaschi, Levi, Martelli and Palamidessi 93] fills this gap by proposing a declarative semantics for logic programs that makes explicit the correspondence between computed answer substitutions and declarative meaning (model-theoretic and fixpoint) of the programs. The idea is to enhance the semantic domain by introducing variables to the Herbrand Universe. The declarative semantics is defined in terms of an immediate consequence operator as in the ground case. Moreover, the answer substitutions for a query can be obtained by "solving" the goal into the non-ground model of the program. However, the semantics is still not equipped to cope with the control features present in real logic languages such as left-to-right selection rule, depth-first search strategy, cut, etc...

In contrast, operational semantics are much more closely linked to the interpreter. Such semantics generally follow a top-down and goal-dependent approach. Because in these semantics the control is explicit, they are suitable to model real logic languages such as Prolog, in which the set of answers to goals computed by a program strongly depends on the control. For this reason the semantic definition of Prolog is usually given by encoding operational semantics in frameworks such as dynamic algebra [Börger 90], deterministic transition systems [de Bruin and de Vink 89], and denotational semantics [Jones and Mycroft 84], [Arbab and Berry 87], and [Debray and Mishra 87].

In [Barbuti, Codish, Giacobazzi and Levi 92] a new approach to defining the semantics of real logic programs is considered. Prolog control features are modeled in a constraint-based language called τ belonging to the *ask/tell* paradigm of Saraswat [Saraswat 90]. A transformation from Prolog to τ is defined to describe the meaning of a Prolog program P in terms of the declarative semantics of the corresponding τ-program. Ask constraints are used to specify the conditions for a clause to be chosen for reduction. These constraints "ask" for the termination of the computation with the preceding clauses. The meaning of a Prolog program is then constructed by composing the meaning of the appropriate τ program (the "logic component") with a corresponding *termination theory* (the "control component") which is not specified from within the semantics.

In contrast, this paper adopts a direct and self-contained approach. Instead of 'asking' about the control, a standard semantics of logic programs is enhanced by an *oracle* which at each step specifies from within the semantics the set of applicable clauses. Different notions of control are specified by different choices of oracles. In particular we focus on an oracle for pure Prolog. A pure Prolog program is a logic program in which an order is fixed for both clauses and goals in the body of clauses [Sterling and Shapiro 86]. Its meaning can be obtained by combining leftmost selection rule and sequential search for unifiable clauses. While the selection rule can easily be specified within the semantics, the specification of sequential search is done by an oracle. This oracle is obtained by viewing the Prolog program as a logic program and by reasoning about its meaning. This can be done both in a top-down manner, by means of a transition system, and in a bottom-up manner, using "T_P-like" methods. Furthermore, the oracle can be introduced into both transition system and T_P-based semantics. We believe that the approach is general and can be applied to extend a wide range of semantic definitions for logic programs to deal with various notions of control.

The oracle approach to the semantics provides the conditions for a generic oracle to ensure that desirable semantic properties of logic programs, such as the existence of a fixpoint semantics and the equivalence between top-down and bottom-up semantics, are satisfied for any oracle-driven computations. This is particularly suitable for the definition of semantics as a basis for reasoning about program behavior and constructing abstract interpretations. Previous semantic definitions for Prolog have not been sufficiently useful for these purposes, although they have existed for some time. Instead, work on abstract interpretation, for example, has concentrated on abstracting semantics which ignore the depth-first nature of Prolog computation. Clearly the results of such analyses are inherently less accurate than the results of similar analyses based on truer semantics. We believe that this is largely due to the complexity of previous definitions in which information about control is mimicked but not explicitly specified. In contrast the semantic-based analysis of a program in our approach can be partitioned into the analysis of the logic component and the analysis of the control component. If no information about the control (which amounts to termination) is obtained then the result is equivalent to current commonly used analyses which ignore control. Otherwise, we can only gain in precision.

The following example illustrates the advantage of using a Prolog semantics instead of a logic programming semantics for the analysis of Prolog programs.

EXAMPLE **1.1**. *Consider a logic program P in Table 1 which is intended to compute the sum of a list of numbers and check that it is positive (non-zero). There is a fundamental difference between the logic program P and the Prolog program P due to the depth-first strategy of the later. The Prolog program diverges when queried with* `positive_sum_list(X,Y)` *while the same goal has a refutation for the corresponding logic program.*

```
positive_sum_list(X,Y) ← list_of_numbers(X), sum(X,Y), positive(Y).  % (1)

list_of_numbers([ ]).                                                % (2)
list_of_numbers([X|Xs]) ← number(X), list_of_numbers(Xs).            % (3)

number(0).                                                           % (4)
number(s(X)) ← number(X).                                            % (5)

sum([ ],0).                                                          % (6)
sum([X|Xs],Z) ← sum(Xs,Y), add(X,Y,Z).                               % (7)

add(0,X,X).                                                          % (8)
add(s(X),Y,s(Z)) ← add(X,Y,Z).                                       % (9)

positive(s(X)).                                                      % (0)
```

Table 1: A logic program.

Now, consider a simple semantic-based type analysis of the Prolog program P and query `positive_sum_list(X,Y)`*, like those proposed in [Barbuti and Giacobazzi 92], [Pyo and Reddy 89], and [Kanomori and Horiuchi 88], which ignore Prolog's search strategy. Current implementations, such as those proposed in [Bruynooghe 91] and [Le Charlier and Van Hentenryck 94], easily detect that any answer for the call* `list_of_numbers(X)` *binds* `X` *to a list of numbers. However, a more accurate analysis based on a Prolog semantics could detect that clause (5) in the Prolog program is never applied. Hence* `X` *is bound to a list of zeroes. Consequently, for any call pattern of* `list_of_numbers`*, the predicate* `number` *in the Prolog program P is deterministic and hence the introduction of cuts is justified:*

```
number(0) ← !.                 % (4′)
number(s(X)) ← !, number(X).   % (5′)
```

Moreover, given such an analysis we can determine that the goal `positive_sum_list(X,Y)` *will never succeed thus justifying the introduction of cuts in the predicate* `list_of_numbers`*:*

```
list_of_numbers([ ]) ← !.                                     % (2′)
list_of_numbers([X|Xs]) ← !, number(X), list_of_numbers(Xs).  % (3′)
```

The transformed Prolog program is equivalent to P in the sense that it gives exactly the same answers for any call to `positive_sum_list`*. However, the transformed program has a preferable behaviour as it does not diverge. Moreover it is deterministic and can be compiled more efficiently as described in [Debray and Warren 89]. The transformation described in this example can be viewed as transforming infinite failure into finite failure. Observe that similar transformations can be applied to introduce cuts to the predicates defining* `sum` *and* `add`*. However these can be justified without considering Prolog's depth-first control strategy because they are called with their first arguments ground.*

The rest of this paper is structured as follows: Section 2 presents some preliminary definitions and notations. Section 3 illustrates how a Prolog oracle is operationally specified and introduced into a transition system semantics for logic programs. This is fairly straightforward and the resulting semantics is concise and resembles standard transition system semantics for logic programs. However as a basis for program analysis we wish to obtain a T_P-like semantics, namely a fixpoint characterization of the program meaning. This is done in Section 4. In particular we define an unfolding-based oracle semantics which is proved to be equivalent to a corresponding bottom-up fixpoint oracle semantics. These semantics are shown to be sound and complete with respect to the operational one. In Section 5 we introduce a suitable oracle for sequential logic programs and for pure Prolog programs. While the first corresponds to a straightforward oracle, the second, once combined with the oracle-based semantics, provides a concise specification for fixpoint semantics of Prolog. We also discuss some approximations for the Prolog oracle showing how the oracle semantics can be abstracted for semantics-based program analysis. We close with a short conclusion. For continuity and ease of readability, the proofs together with auxiliary lemmata, have been moved to the appendix. This paper is an extended and revised version of [Barbuti, Codish, Giacobazzi and Maher 92]: in particular the ideas of [Barbuti, Codish, Giacobazzi and Maher 92] are reported in Sections 3 and 4.1.

2 Preliminaries

In the following we assume familiarity with the standard definitions and notation for logic programs. The standard reference works by Apt [Apt 90] and Lloyd [Lloyd 87] provide the necessary background material.

Given a possibly infinite set X, we denote by X^* the set of finite sequences of symbols in X. The concatenation of sequences π_1 and π_2 is denoted $\pi_1 :: \pi_2$. The sequence of symbols with first element s followed by the sequence S is denoted $s \mid S$. The empty sequence is denoted Λ. The *length* of a sequence π is denoted $|\pi|$. We extend this notation to denote set-cardinality. If $R \subseteq X \times X$ is a relation on X then R^* denotes the reflexive and transitive closure of R. For any $a, b \in X$ we denote $a\ R^k\ b$, for $k \geq 2$, the relation $\exists c \in X :\ a\ R\ c\ \wedge\ c\ R^{k-1}\ b$, where $R^1 = R$. If $\sim$ is an equivalence relation on X, we denote by $[x]_\sim$ the equivalence class of $x \in X$. When clear from the context we abbreviate $[x]_\sim$ by $[x]$ and often abuse notation by letting the elements of a set denote their corresponding equivalence classes. In the following, Nat will denote the set of natural numbers. Moreover, we denote the power set of a set X by $\wp(X)$. Finally, to specify function parameters, we often make use of Church's lambda notation.

Let $L(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ be a non empty complete lattice with partial ordering $\sqsubseteq$, least upper bound $\sqcup$, greatest lower bound $\sqcap$, top element $\top$ and bottom element $\bot$. Let $f : L \to L$ be a function. We say that f is idempotent if $f(f(x)) = f(x)$, reductive if $f(x) \sqsubseteq x$, and additive if $f(x \sqcup y) = f(x) \sqcup f(y)$, for every x and y. Functions are ordered as follows: $f \sqsubseteq g$ iff $f(x) \sqsubseteq g(x)$ for every x. The ordinal power of f is defined as usual: $f{\uparrow}0(x) = x$, $f{\uparrow}\alpha(x) = f(f{\uparrow}(\alpha - 1)(x))$ for every successor ordinal α and $f{\uparrow}\alpha(x) = \bigsqcup_{\delta<\alpha} f{\uparrow}\delta(x)$ for every limit ordinal α. ω denotes the first limit ordinal. The fundamental theorem of Tarski states that the set of fixpoints of a monotonic function f is a complete lattice. The least fixpoint of a function f is denoted $lfp(f)$. If f is a continuous function then $lfp(f) = f{\uparrow}\omega(\bot)$ and is often denoted $f{\uparrow}\omega$.

Throughout, Σ, Π and Var will respectively denote a set of function symbols, a set of predicate symbols and a denumerable set of variables. With each function symbol $f \in \Sigma$ and predicate symbol $p \in \Pi$ is associated a unique natural number called its *arity*; a (predicate or function) symbol f with arity n is written f/n. The non-ground term algebra over Σ and Var is denoted $Term(\Sigma, Var)$ or $Term$ for short. The corresponding Herbrand universe is $Term(\Sigma, \emptyset)$. In the following we will always implicitly consider non trivial Herbrand universes (i.e. containing more than one element). The set of atoms constructed from predicate symbols in Π and terms from $Term$ is denoted $Atom(\Pi, \Sigma, Var)$ or $Atom$ for short. A goal is a sequence of atoms and we define $Goal = Atom^*$. The empty atom sequence is denoted by $true$, and sometimes is omitted. A clause is an object of the form $h \leftarrow B$ where h is an atom, called the *head*, and B is a goal, called the *body*. The set of clauses constructed from elements of $Atom(\Pi, \Sigma, Var)$ is denoted $Clause(\Pi, \Sigma, Var)$ or $Clause$ for short. Syntactic equivalence for any syntactic object (e.g. terms, atoms, clauses, *etc.*) is denoted $\equiv$. The set of variables that occur in a syntactic object t is denoted $vars(t)$. A logic program is a (finite) set of clauses P. We sometimes associate with a program an ordering on the clauses of the program. In this case we let P_ℓ denote a program P and an injection $\ell : P \to Nat$. The injection ℓ is intended to reflect the textual ordering of the clauses in P as assumed in Prolog.

A *substitution* ϑ is a mapping from Var to $Term$ which acts as the identity almost everywhere, i.e. such that its domain $dom(\vartheta) = \{x \in Var \mid \vartheta(x) \neq x\}$ is finite. It extends to apply to any syntactic object in the usual way. The set $range(\vartheta)$ is defined as $range(\vartheta) = \cup_{x \in dom(\vartheta)} vars(\vartheta(x))$. We will often make use of the set theoretic notation $\{x \mapsto t \mid x \in dom(\vartheta),\ \vartheta(x) = t\}$ to represent ϑ. The empty substitution is denoted ε. For a syntactic object s, we denote by $\vartheta_{|s}$ the restriction of ϑ to the variables of s. The application of a substitution ϑ to a syntactic object s is denoted by $s\vartheta$. If Θ is a set of substitutions, then $s\Theta = \{s\vartheta \mid \vartheta \in \Theta\}$. Composition of substitutions ρ and σ is defined as usual and denoted $\rho\sigma$. A substitution σ is *idempotent* if $\sigma\sigma = \sigma$ (or $dom(\sigma) \cap range(\sigma) = \emptyset$). The set of idempotent substitutions is denoted Sub. A *variable renaming* is a (generally not idempotent) substitution which is a bijection on Var. Syntactic objects t_1 and t_2 are *equivalent up to renaming*, denoted $t_1 \sim t_2$, if, for some variable renaming ρ, $t_1\rho = t_2$. We write $t_1 \leq t_2$ iff there exists a substitution ϑ such that $t_1 = t_2\vartheta$. These notions can be extended on idempotent substitutions in the obvious way. Given an equivalence class (induced by renaming) $\hat{t}$ of syntactic objects and a finite set of variables $V \subseteq Var$, it is always possible to find a representative t of $\hat{t}$ (i.e., an object t such that $[t] = \hat{t}$) which contains no variables from V. Let I be a set of (equivalence classes of) syntactic objects and let s be a syntactic object. Then, $A \ll_s I$ denotes that A is a renaming (representative) of an element of I which does not share variables with s. Furthermore, we extend this to specify tuples of renamed apart syntactic objects:

$$\langle A_1, \ldots, A_n \rangle \ll_s I \Leftrightarrow \bigwedge_{i=1}^{n} (A_i \ll_s I) \ \wedge \ \bigwedge_{i \neq j} (vars(A_i) \cap vars(A_j) = \emptyset)$$

The notion of unification can be given in terms of equations and equation solving. This is because the difference between unification and equation solving is only a matter of representation [Lassez, Maher, and Marriott 88]. An equation has the form $t = u$ where t, u are syntactic objects and $=$ is interpreted as syntactic equality. Given a set E of equations, E is unifiable iff there exists ϑ such that for each $(t = u) \in E$, $(t\vartheta) \equiv (u\vartheta)$. We denote by $Unif(E)$ the (possibly empty) set of unifiers of E. We fix a partial function mgu which maps a pair of syntactic objects (or a set of equations) into an idempotent most general unifier of the objects, if such exists. For sequences of syntactic objects (e.g. terms, atoms) $\langle t_1, ..., t_n \rangle$ and $\langle u_1, ..., u_n \rangle$, $n \geq 1$,

$mgu(\langle t_1, ..., t_n \rangle, \langle u_1, ..., u_n \rangle)$ is an equivalent notation for $mgu(\{t_1 = u_1, ..., t_n = u_n\})$. For a set of equations E: $mgu(E)$ is an idempotent element in the set $\{\vartheta \in Unif(E) \mid \forall \vartheta' \in Unif(E) : \vartheta \leq \vartheta'\}$. $mgu(E) = fail$ means that E is not unifiable. It is well known that the mgu of a set of equations is unique up to renaming (see [Eder 85] and [Lassez, Maher, and Marriott 88]). Let $\vartheta = \{x_1 \mapsto t_1, ..., x_n \mapsto t_n\}$ be a substitution. We denote $\mathcal{E}(\vartheta) = \{x_1 = t_1, ..., x_n = t_n\}$. For non-trivial Herbrand universes, $\vartheta = mgu(E)$ iff E has the same solutions as $\mathcal{E}(\vartheta)$ (denoted $\mathcal{E}(\vartheta) \approx E$) [Lassez, Maher, and Marriott 88].

3 Operational oracle semantics

This section presents an operational semantics for logic programs defined in terms of transition systems. The semantics is standard except for the fact that an *oracle* is introduced providing a uniform approach to capture different control strategies. In particular, we describe an oracle to capture the depth-first control strategy of Prolog.

3.1 Transition systems

DEFINITION **3.1** (transition system).
A transition system is a pair (X, R) consisting of a set X of states and a transition relation $R \subseteq X \times X$ on states.

In general, transition systems are applied to describe program meaning in terms of the transitive closure of the transition relation which specifies a basic computation step.

DEFINITION **3.2** (derived states).
Let $\Psi = (X, R)$ be a transition system on states X. If $s\ R^*\ s'$ then we say that s' is derived from s. If $s\ R^k\ s'$ we say that s' is derived from s in k steps.

A transition system can be viewed as associating a *computation tree* with (initial) states.

DEFINITION **3.3** (computation tree).
Let $\Psi = (X, R)$ be a transition system and $x \in X$ a state. The computation tree associated with x is the tree $T(\Psi, x)$ with root x and inductively defined so that every state y derived from x is a vertex in the tree and has a child z iff $y\ R\ z$.

The notion of *universal termination* is formalized by the following:

DEFINITION **3.4** (universal termination).
Let $\Psi = (X, R)$ be a transition system and $x \in X$ a state. We say that x terminates universally for Ψ iff $T(\Psi, x)$ is of finite height.

3.2 Transition systems for logic programs

The transition system approach is well suited and commonly applied to define semantics of logic languages. States are goals and correspond to resolvents, the transition relation specifies resolution and the transition system captures the computation tree for initial goals.

DEFINITION **3.5** (transition system, Ψ_P^{lp}).
Let P be a logic program. The transition system Ψ_P^{lp} is $(Goal, \xrightarrow{lp}_P)$ where $\xrightarrow{lp}_P \subseteq Goal \times Goal$ is the smallest relation such that $G \xrightarrow{lp}_P G'$ iff

1. $G = \langle A_1, \ldots, A_i, \ldots, A_k \rangle$, $1 \leq i \leq k$;
2. $H \leftarrow B_1, \ldots, B_n \ll_G P$ *such that* $\varphi = mgu(A_i, H)$; *and*
3. $G' = \langle A_1, \ldots, B_1, \ldots, B_n, \ldots, A_k \rangle \varphi$.

The substitution φ is referred to as the associated substitution *of the transition $G \xrightarrow{lp}_P G'$.*

The standard notion of a computation tree for an initial goal G and a program P is captured by $T(\Psi_P^{lp}, G)$. Definition 3.5 is often refined (e.g. [Lloyd 87]) by introducing a *selection rule* on the atoms in a state. In particular, Prolog assumes a left-to-right selection rule. In addition logic programming languages typically introduce a control strategy to specify which clauses should be applied to the selected atom. Prolog employs a depth-first search strategy, applying the rules in textual order. In the following we introduce the notion of an oracle transition system in which an oracle is used to capture specific control features.

DEFINITION **3.6** (oracle).
An oracle is a function $f : \wp(Clause) \times Goal \rightarrow \wp(Clause)$ such that for any logic program P and goal G: $f(P, G) \subseteq P$ (i.e. f is reductive).

The following transition system employs a left-to-right selection rule. Moreover it enhances Definition 3.5, by introducing a generic *oracle*. A Prolog semantics is then derived as an instance of this definition by specifying an appropriate oracle. By Prolog, we mean pure Prolog in which evaluation is assumed to proceed until all answers are generated and that unification is sound and terminating (i.e. the occur check is performed).

DEFINITION **3.7** (oracle transition system, Ψ_P^f).
Let P be a Prolog program and f be an oracle. The transition system Ψ_P^f is $(Goal, \xrightarrow{f}_P)$ where $\xrightarrow{f}_P \subseteq Goal \times Goal$ is the smallest relation such that $G \xrightarrow{f}_P G'$ iff

1. $G = \langle A_1, \ldots, A_k \rangle$;
2. $H \leftarrow B_1, \ldots, B_n \ll_G f(P, G)$ *such that* $\varphi = mgu(A_1, H)$; *and*
3. $G' = \langle B_1, \ldots, B_n, A_2, \ldots, A_k \rangle \varphi$.

The substitution φ is referred to as the associated substitution *of the transition $G \xrightarrow{f}_P G'$. The substitution associated with a sequence of transitions is the composition of the substitutions associated with the transitions in the sequence.*

The transition system Ψ_P^{slp} is defined by taking the oracle $slp = \lambda P.\lambda G.P$ in Definition 3.7. This specifies what we call *sequential logic programs* which are logic programs executed with a left-to-right selection rule.

Intuitively, the computation tree for a Prolog program P and goal G corresponds to a pruning of $T(\Psi_P^{slp}, G)$. The children of a node are assumed to be ordered to reflect the textual ordering of

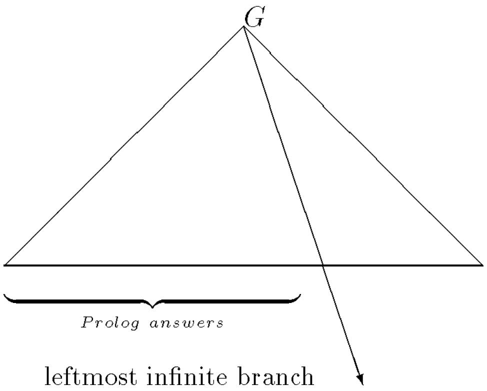

Figure 1: Prolog computation tree.

clauses in P and everything to the right of the left-most infinite path is excluded by the pruning (see Figure 1). Prolog evaluation corresponds to a depth-first, left-to-right traversal of the Prolog computation tree, and is assumed to find all of the alternative answers. This intuition can be applied to define a transition system for Prolog by imposing a condition on Ψ_P^{slp}, excluding goals which are to the right of an infinite branch. Consider an oracle *prolog* which, for a given program P, associates each goal with exactly those clauses in P which are eventually chosen by Prolog's search strategy. In this case $\xrightarrow{prolog}_P$ specifies (somewhat tautologously) the transition relation for *Prolog*. A similar argument can be applied to specify transition relations for Prolog with cut, and meta-logical predicates such as $var(x)$ or negation.
A less tautologous description of the Prolog oracle is provided as follows. Let P_ℓ be a Prolog program (where ℓ is the textual ordering) and let G be a goal. Then the clause $C \in P$ is chosen by the oracle iff each path in $T(\Psi_P^{slp}, G)$ starting with a transition which applies a clause C' such that $\ell(C') < \ell(C)$ is finite. It is straightforward to show that the Prolog computation tree for a goal G and program P_ℓ terminates universally iff $T(\Psi_P^{slp}, G)$ terminates universally [Vasak and Potter 86]. Furthermore, by Konig's Lemma, $T(\Psi_P^{slp}, G)$ terminates universally iff it is finite because $\xrightarrow{slp}_P$ has finite branching. Hence,

Definition **3.8** (Prolog oracle).
Let P be a logic program ordered textually by ℓ and G a goal. The Prolog oracle is given by

$$prolog(P, G) = \left\{ C \in P \;\middle|\; \begin{array}{l} Q = \{C' \mid \ell(C') < \ell(C)\}, \\ G \xrightarrow{slp}_Q G' \;\Rightarrow\; T(\Psi_P^{slp}, G') \text{ is finite} \end{array} \right\}.$$

Note that in contrast to $\xrightarrow{lp}_P$ and $\xrightarrow{slp}_P$, the relation $\xrightarrow{prolog}_P$ is not decidable (it is semi-decidable). However, the vertices in $T(\Psi_P^{prolog}, G)$ are recursively enumerable just as the vertices of $T(\Psi_P^{lp}, G)$ and $T(\Psi_P^{slp}, G)$. Moreover, approximations of Prolog semantics are derived by determining that a given clause is not chosen (for a given goal) by the Prolog oracle. This boils down to determining that a corresponding computation tree is infinite. This is similar to the proposal described in [Lever 91] to reason about SLS resolution [Przymusinski 89] applying loop-checking techniques. Some applications in dataflow analysis will be discussed in Section 5.

EXAMPLE **3.1**.
Consider the program P from Example 1.1 and the goal `positive_list_sum(X,Y)`. *The Prolog computation tree is illustrated in Figure 2. To see why Prolog does not apply the clause (5) to the goal G (indicated in Figure 2 by a double arrow) it is sufficient to observe that the tree which first applies clause (4) to G is infinite. In this case, clause (5) is not in $prolog(P, G)$.*

The notion of the *answers* and *partial answers* for a goal G are defined as usual in transition system semantics by considering the substitutions associated with the corresponding sequences of transitions. We do not distinguish partial answers and the corresponding instances of the initial goal as computation proceeds.

DEFINITION **3.9** (answers and partial answers).
*Let P be a logic program, G a goal and $L \in \{lp, slp, prolog\}$. We say that G' is an L-*partial answer *for G (at depth k) iff $G' \sim G\vartheta$ where ϑ is the associated substitution of a sequence $G \xrightarrow{L}{}^{k}_{P} G''$ of transitions. If $G'' = true$ then G' is a L-answer for G at depth k.*

While it is well known ([Apt and van Emden 82]) that the answers for a goal are independent of the selection rule, this property does not extend to partial answers. For example, consider the program:

```
p(x) ← q(x), r(x).
r(s(x)) ← r(x).
```

The lp-partial answers for $p(x)$ are $\{p(s^n(x)) \mid n > 0\}$, whereas the slp-partial answers are $\{p(x)\}$ (up to variable renaming). Furthermore, observe that the tree $T(\Psi_P^{lp}, p(x))$ has an infinite branch while $T(\Psi_P^{slp}, p(x))$ is finite.

We will need the following

DEFINITION **3.10** (derivation paths).
Let $\Psi_P^f = (Goal, \xrightarrow{f}_P)$ be an oracle transition system for program P ordered textually by ℓ and oracle f. A derivation path for Ψ_P^f and goal G is an element $\pi \in Nat^*$ such that either $\pi = \Lambda$ or $\pi = \langle n \rangle :: \pi'$, where $G \xrightarrow{f}_P G'$ by applying the clause $C \in P$, $\ell(C) = n$ and π' is a derivation path for G'. A derivation from G to G' with path π is denoted $G \xrightarrow[\pi]{f}{}^{*}_{P} G'$.

Derivation paths (paths for short), namely sequences of symbols in Nat^*, are denotations which represent the sequence of clauses applied in the derivation. We extend the notion of slp- and lp-partial answers to include paths in the obvious way. The link between paths and the depth of a computation is a consequence of the following straightforward proposition.

PROPOSITION **3.1**.
Let $L \in \{lp, slp\}$. G' is a L-(partial) answer at depth d for a goal G and program P iff it is a L-(partial) answer with path π for G and P and $d = |\pi|$.

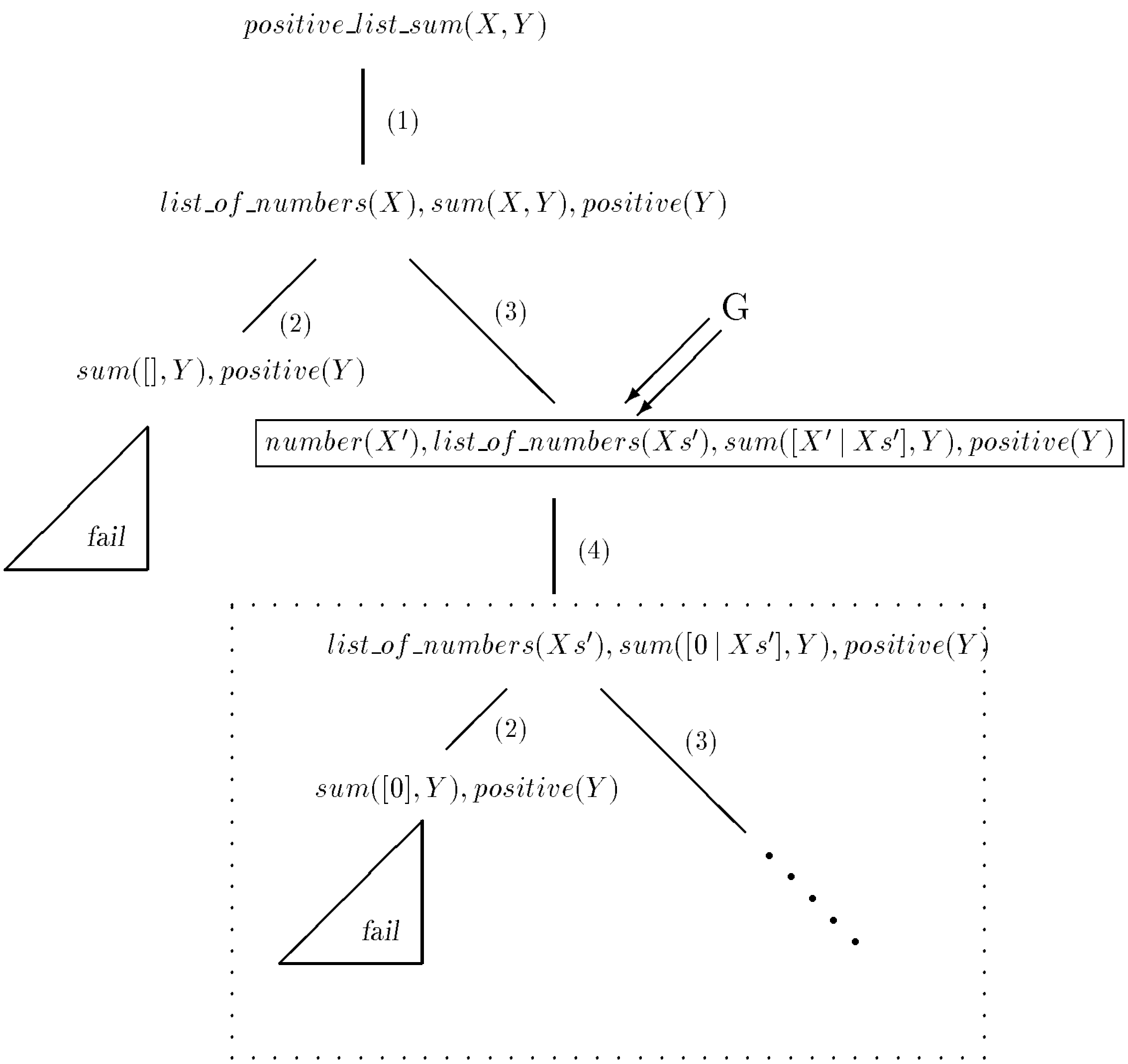

Figure 2: An example Prolog computation tree.

We adopt the convention that branches in a computation tree are labeled by the number of the clause applied. A finite subtree which does not contain a "true" leaf is indicated as `fail`.

4 Oracle declarative semantics

The objective of this section is to provide a semantic basis for the analysis of Prolog programs which maintains the simplicity of the T_P-based semantics for logic programs. We are motivated by the relative simplicity in constructing program analyses as abstractions of T_P semantics, as proposed in [Barbuti, Giacobazzi and Levi 93] and [Codish, Dams and Yardeni 94]. We aim to provide a semantics basis for more precise program analyses by introducing specific control issues into the semantics, following the approach illustrated above for transition systems. The reader is reminded that proofs have been deferred to the appendix.

The semantics we develop provides a success set from which Prolog answers can be determined. In order to specify the oracle from within the semantics we enhance the semantic objects to capture also partial computations and derivation paths. Our approach is based on the observation that both top-down and bottom-up semantics of logic programs can be expressed in terms of program *unfolding* [Levi 88]. Section 4.1 introduces a top-down unfolding semantics for logic programs with oracles. Later, in Section 4.3, a bottom-up unfolding semantics is introduced by allowing the same oracle to be applied in the process of unfolding, in a "reversed way". In Sections 5.1 and 5.2 we consider two instances of the oracle semantics by specifying the declarative oracles *slp* and *prolog*.

The semantics we consider is that of partial computations, which includes the computed answer semantics as a special case. This facilitates the specification of both logic and control in a similar approach. As partial computations are dependent on the selection rule, we assume a left-to-right selection rule in what follows. Declarative semantics with selection rules for logic programs are independently proposed also in [Gabbrielli and Meo 92], by considering the more general family of *local selection rules*. We follow the approach of [Bossi, Gabbrielli, Levi and Meo 94] which defines a bottom-up semantics in which interpretations consist of clauses instead of atoms. Each clause in an interpretation represents a partial computation from its head to its body. We further enhance interpretations so that each clause is associated with a path indicating its position in a corresponding computation tree. The length of the paths in an interpretation provides a handle on the finiteness of computations, as needed for the oracle specification in the case of Prolog. Paths are ordered by the lexicographical ordering $\preceq$. The *lexicographic ordering* $\preceq$ on paths is defined by $\pi \preceq \pi'$ iff $\pi = p_1...p_n$, $\pi' = p'_1...p'_m$ for $p_i, p'_i \in Nat$ and either there exists an integer j such that $p_j \leq p'_j$ and for all $i < j$, $p_i = p'_j$, or $n \leq m$ and $p_i = p'_i$ for $1 \leq i \leq n$. This enables us to reason about such concepts as the "leftmost" infinite path in a computation tree.

Definition **4.1** (interpretations).
Let $\mathcal{H} = Clause(\Pi, \Sigma, Var)/_{\sim} \times Nat^*$. *The domain of interpretations is* $Int = (\wp(\mathcal{H}), \subseteq)$.

In order to view programs as interpretations, we let P_ℓ denote the interpretation associated with the program P ordered textually by ℓ, namely the set of clauses $\left\{ (C; \ell(C)) \,\middle|\, C \in P_\ell \right\}$. In what follows we will implicitly assume the existence of ℓ and, when clear from the context, we denote the interpretation P_ℓ by P.

The following definition extends the notion of an oracle (Definition 3.6) for interpretations. This provides suitable oracles for the declarative semantics. Their meaning is different from those defined in Definition 3.6. We associate an oracle with each program, specifying which clauses in an interpretation are applicable to a goal.

DEFINITION **4.2** (declarative oracle).
Let P be a program. An oracle for P is a function: $f_P : Int \times Goal \rightarrow Int$ which is idempotent, additive and reductive in its first argument. When clear from the context the subscript P is omitted.

Idempotence specifies that the reduction to obtain the set of "applicable clauses" is all at once. Additivity specifies that a clause may be "applicable" independently from the other clauses in the current interpretation. As we will see later, the applicability of a clause in a Prolog computation depends only on its textual order in the program.

A difference between oracles according to Definition 3.6 (we will call them operational oracles) and declarative oracles is that the latter are defined over interpretations incorporating paths, whereas the former are not. For simplicity, in the following we refer to declarative oracles only. However, it is straightforward to extend the *operational oracle semantics* and related notions in Section 3 with declarative oracles, by interpreting programs P as corresponding interpretations P_ℓ. In particular, if f is a declarative oracle, we denote by $G \overset{f}{\underset{\pi}{\longrightarrow}}{}^{*}_{P} G'$ the sequence of transitions $G \overset{f'}{\underset{\pi}{\longrightarrow}}{}^{*}_{P} G'$, where f' is the operational oracle defined by $f'(P, G) = f(P_\ell, G)$.

In general an oracle $f(I, G)$ for a program P specifies the set of clauses in an interpretation I for P which are applicable to the goal G. In the following we will focus on the class of *downward closed* and *sound* oracles which include in particular the Prolog oracle.

Downward closure is specified on both of the oracle arguments. It specifies the reduction implemented by the oracle in terms of the instantiation of its arguments. In particular we require that the applicability of a clause will not depend on its degree of instantiation, as in Prolog, where the applicability condition for a clause depends upon the preceding clauses in the textual order. In the following, for an oracle f, the applicability of a clause C to a goal G is expressed by $f(\{C\}, G) \neq \emptyset$. Note that the applicability condition for a clause C to a goal G does not imply that C and G unify.

DEFINITION **4.3** (downward closed oracles).
An oracle f is downward closed *iff for each interpretation I, goals G and G', clause C and substitution ϑ:*

- $f(I, G) \subseteq f(I, G\vartheta)$;
- $f(I, G :: G') \subseteq f(I, G)$;
- $f(\{C\}, G) \neq \emptyset \quad \Leftrightarrow \quad f(\{C\vartheta\}, G) \neq \emptyset$.

Note that the second condition is related with the left-to-right selection rule of Prolog. Downward closure of the oracle implies the following propositions. The first shows that variable names are not significant.

PROPOSITION **4.1** ($\sim$-congruence).
Let f be a downward closed oracle for a program. Then, for any interpretation I and goals G, G': $G \sim G' \quad \Rightarrow \quad f(I, G) = f(I, G')$.

The following property is similar to properties of stability [Gaifman, Maher and Shapiro 91] and monotonicity [Saraswat, Weinbaum, Kahn and Shapiro 88] in relation to concurrent languages.

PROPOSITION **4.2**.
Let P be a program and f be a downward closed oracle for P. If $G \xrightarrow[\pi]{f}{}^{*}_{P} G'$ with associated substitution ϑ, then $G\vartheta \xrightarrow[\pi]{f}{}^{*}_{P} G''$ and $G'' \sim G'$.

The converse (i.e. Lifting Lemma 8.2 in [Lloyd 87]) may not hold (e.g. for the Prolog oracle), as shown in the following example:

EXAMPLE **4.1**.
Consider the following Prolog program P:

```
p(s(x)) ← p(x).
p(x).
```

While the goal $p(0)$ succeeds in a Prolog transition system semantics, $p(x)$ does not.

We now generalize the notion of partial answers for declarative oracles and interpretations.

DEFINITION **4.4** (f-answers and f-partial answers).
Let $f : Int \times Goal \rightarrow Int$ be an oracle for a program P, $I \in Int$ an interpretation and $G = \langle a_1, \ldots, a_n \rangle$ a goal. We say that G' is an f-partial answer for G with path π in I *iff there exists $k \in \{1, ..., n\}$ such that for each $1 \le i \le k$: $(h_i \leftarrow B_i, \pi_i) \ll_{G\vartheta_{i-1}} f(I, (a_i, ..., a_n)\vartheta_{i-1})$ where $\vartheta_0 = \varepsilon$, $B_i = true$ for $i < k$, $\vartheta_i = \vartheta_{i-1} mgu(a_i\vartheta_{i-1}, h_i)$, $G\vartheta_k \sim G'$ and $\pi = \pi_1 :: \cdots :: \pi_k$.* The corresponding resolvent *in I with path π is $(B_k :: \langle a_{k+1}, ..., a_n \rangle)\vartheta_k$. If $k = n$ and $B_k = true$ then G' is a f-answer for G in I with path π. The set of f-partial answers with paths for G in I is denoted*

$$\mathcal{P}^f(I, G) = \left\{ (G', \pi) \;\middle|\; \begin{array}{l} G' \textit{is an } f\textit{-partial answer} \\ \textit{of } G \textit{ with path } \pi \textit{ in } I \end{array} \right\}.$$

In the following we denote by Φ the interpretation $\left\{ ([p(\bar{x}) \leftarrow p(\bar{x})]_{\sim}; \Lambda) \;\middle|\; p \in \Pi \right\}$ where each clause is a tautology and each path is empty. This interpretation associates any goal G with the corresponding empty partial computations. As in [Bossi, Gabbrielli, Levi and Meo 94], it is used to simplify the presentation of the semantics of partial computations given below.

4.1 Top-down unfolding-based oracle semantics

The unfolding semantics mimics the operational semantics by a top-down construction in which a clause $(h \leftarrow B; \pi)$ represents a partial computation from h to B and π indicates the position of the partial computation in the proof-tree. This results in a natural extension of the oracle-based operational semantics to the unfolding semantics construction.

DEFINITION **4.5** (unfolding oracle semantics).
Let P be a logic program, f a declarative oracle for P and I an interpretation.

$$Unf^f_P(I) = \left\{ ([(h \leftarrow B)\sigma]_{\sim}; \pi :: \langle l \rangle) \;\middle|\; \begin{array}{l} C \equiv (h \leftarrow b_1, \ldots, b_n; \pi) \in I, \\ (h' \leftarrow B'; l) \ll_C f(P, (b_1, \ldots, b_n)), \\ \sigma = mgu(b_1, h'), \\ B = B' :: \langle b_2, \ldots, b_n \rangle \end{array} \right\}$$

The following proposition is an immediate consequence of the definition..

Proposition **4.3**.
$\lambda I.Unf_P^f(I)$ is additive.

The unfolding process is intended to capture all partial computations for arbitrary initial goals. In the case of logic programs it is sufficient to consider answers (or partial answers) for goals of the form $p(x_1, ..., x_n)$. However as illustrated by Example 4.1 this is not the case for our oracle-based semantics due to the lack of the lifting lemma. Therefore, we initiate the unfolding process with any instance of a tautological clause i.e. an element of $lwr(\Phi)$, where

$$lwr(I) = \left\{ (h \leftarrow B; \pi)\vartheta \;\middle|\; \begin{array}{l} (h \leftarrow B; \pi) \in I \\ dom(\vartheta) \subseteq vars(h) \end{array} \right\}$$

for any interpretation I.

Definition **4.6** (unfolding-based fixpoint semantics).
Let P be a program, f be a corresponding oracle and $\mathcal{U}_P^f = \lambda I.Unf_P^f(I \cup lwr(\Phi))$. The unfolding semantics *of P is $lfp(\mathcal{U}_P^f)$.*

Note that the unfolding of a clause in Φ with any clause C is the clause C itself; thus in the first step $\mathcal{U}_P^f(\emptyset)$ collects all the clauses in $lwr(P)$ which are applicable to an atomic goal:

$$\mathcal{U}_P^f(\emptyset) = \left\{ (h \leftarrow B; l)\sigma \;\middle|\; \begin{array}{l} (h \leftarrow B; l) \in f(P, h') \\ \sigma = mgu(h, h'),\ h' \in Atom \end{array} \right\}.$$

More generally we can state the following lemma which is a straightforward consequence of the definition and of the downward closure of the oracle.

Lemma **4.4**.
Let P be a program and f be an oracle. If $[C]_\sim \in lfp(\mathcal{U}_P^f)$ then $lwr(\{[C]_\sim\}) \subseteq lfp(\mathcal{U}_P^f)$.

By Unf_P^f additivity, $lfp(\mathcal{U}_P^f)$ can be obtained as the limit of the sequence:

$\mathcal{U}_P^f{\uparrow}1 = Unf_P^f(lwr(\Phi))$,

$\mathcal{U}_P^f{\uparrow}n = Unf_P^f(\mathcal{U}_P^f{\uparrow}(n-1)) \cup Unf_P^f(lwr(\Phi))$ for $n > 1$.

According to the structure of this sequence, by a straightforward induction on n we can prove the following

Lemma **4.5**.
Let P be a program, f be an oracle and $(c; \pi) \in \mathcal{U}_P^f{\uparrow}\omega$. Then $|\pi| = n$ iff $(c; \pi) \in Unf_P^f{\uparrow}n(lwr(\Phi))$.

Thus, by definition: $(c; \pi) \in \mathcal{U}_P^f{\uparrow}\omega$, $|\pi| \leq n$ iff $(c; \pi) \in \mathcal{U}_P^f{\uparrow}n$.

The following result specifies how the oracle based semantics preserves the ordering among arbitrary oracles.

Theorem **4.6**.
Let P be a program and f, g be oracles for P. If for each interpretation I and goal G: $f(I, G) \subseteq g(I, G)$ then $lfp(\mathcal{U}_P^f) \subseteq lfp(\mathcal{U}_P^g)$.

Among the set of oracles, we can distinguish the identity oracle, $slp = \lambda I \lambda G.I$ (see Section 5.1 for a semantic interpretation of slp) and the empty oracle, $\perp = \lambda I \lambda G.\emptyset$. They are, respectively, the greatest and least oracles. Consequently, for any oracle f, $\emptyset = lfp(\mathcal{U}_P^\perp) \subseteq lfp(\mathcal{U}_P^f) \subseteq lfp(\mathcal{U}_P^{slp})$.

4.2 Relation between fixpoint and operational oracle semantics

The relation between unfolding-based oracle semantics and the transition-system semantics with oracles is provided by soundness and completeness results. We need to restrict the class of oracles to obtain these desirable results.

DEFINITION **4.7** (oracle soundness).
Let P be a program. An oracle f for P is sound *with respect to unfolding iff for each goal $g \mid G$,*

$$g \mid G \xrightarrow[\pi]{f}{}^{*}_{P} (B :: G)\vartheta \text{ with substitution } \vartheta \quad \Rightarrow \quad f(\{(g \leftarrow B; \pi)\vartheta\}, g \mid G) \neq \emptyset.$$

By the soundness condition, the sequential applicability of clauses to a goal implies the applicability to the same goal of a modified clause corresponding to their sequential application. This property allows us to merge the sequential applicability of different clauses into a single applicable clause, and plays a central role in proving soundness and completeness of the declarative oracle semantics.

EXAMPLE **4.2**.
Consider the program P in Example 1.1. The soundness condition is clearly verified by the typical Prolog's search strategy. For instance (see Figure 2), if clause (1) is applicable to the goal `positive_list_sum(X,Y)` *and clause (3) is applicable to the resulting goal:*

$$\texttt{list\_of\_numbers(X)}, \texttt{sum(X, Y)}, \texttt{positive(Y)},$$

resulting in the state $\texttt{B} = \texttt{number(X}'\texttt{)}, \texttt{list\_of\_numbers(Xs}'\texttt{)}, \texttt{sum([X}' \mid \texttt{Xs}'\texttt{], Y)}, \texttt{positive(Y)}$, *then the clause obtained by combining the application of (1) and (3):*

$$(\texttt{positive\_list\_sum(X, Y)} \leftarrow \texttt{B}; \langle 1\rangle :: \langle 3\rangle)$$

is applicable to the initial goal `positive_list_sum(X,Y)`. *The soundness of the* prolog *oracle will be formally discussed in Section 5.*

EXAMPLE **4.3**.
Consider the program P

```
p ← q.   % (1)
q ← r.   % (2)
```

Define an oracle f for P as follows. For singleton interpretations we define

$$\begin{aligned} f(\{(p \leftarrow q; \pi)\}, G) &= \{(p \leftarrow q; \pi)\} \\ f(\{(q \leftarrow r; \pi)\}, G) &= \{(q \leftarrow r; \pi)\} \\ f(\{(c; \pi)\}, G) &= \emptyset \end{aligned}$$

for every goal G and path π, and every clause c except the clauses in P. For other interpretations we define $f(\cup_j I_j, G) = \cup_j f(I_j, G)$ for every goal G.

f is a downward closed oracle for P (we leave verification of this fact to the reader) but it is not sound with respect to unfolding. Clearly $p \xrightarrow[\langle 1\rangle :: \langle 2\rangle]{f}{}^{*}_{P} r$. However $f(\{(p \leftarrow r; \langle 1\rangle :: \langle 2\rangle)\}, p) = \emptyset$.

In the following we restrict our attention to downward closed and sound oracles only.

The following theorem states the soundness of the unfolding-based fixpoint semantics with respect to the operational one. In the following we will implicitly assume partial answers with non empty paths. It is worth noticing that *empty partial answers* can be obtained from any goal by applying clauses in Φ.

THEOREM **4.7** (soundness).
Let P be a program, f an oracle for P, G a goal and ϑ the substitution associated with $G \xrightarrow[\pi]{f}{}^{*}_{P} G'$. Then, $G\vartheta$ is an f-partial answer of G with path π in the interpretation $lfp(\mathcal{U}_P^f)$ and $G' \sim G''$, where G'' is the corresponding resolvent with path π in $lfp(\mathcal{U}_P^f)$.

To prove a completeness result for the fixpoint semantics with respect to the operational one, we need an additional property for the oracles. For this purpose we introduce the notion of *$\mathcal{U}$-derivable clauses*. This allows us to consider only clauses which may be part of a computation for the program.

DEFINITION **4.8** ($\mathcal{U}$-derivable clauses).
A clause $(c;\pi)$ is $\mathcal{U}$-derivable (in a program P) iff $(c;\pi) \in lfp(\mathcal{U}_P^{slp})$.

Because interpretations and programs have the same structure, in the following we abuse by letting I be a generic interpretation in Unf_I^f.

DEFINITION **4.9** (oracle completeness).
Let P be a program. An oracle f for P is complete *with respect to unfolding iff for each goal $g \mid G$ and $\mathcal{U}$-derivable clauses $(h \leftarrow B;\pi)$ and $(c;\pi')$ sharing no variables with $g \mid G$ and with each other, if $f(Unf^{slp}_{\{(c;\pi')\}}(\{(h \leftarrow B;\pi)\}), g \mid G) \neq \emptyset$ and $mgu(h,g) \neq fail$ then*

- *$f(\{(h \leftarrow B;\pi)\}, g \mid G) \neq \emptyset$, and*
- *$f(\{(c;\pi')\}, (B :: G)mgu(h,g)) \neq \emptyset$.*

By completeness we can decompose the applicability condition for the unfolding of clauses in terms of the sequential applicability of each clause involved in the process. Therefore, if a $\mathcal{U}$-derivable clause is applicable to a goal, each clause in the derivation is sequentially applicable to the same goal. This corresponds precisely to the reverse case of the soundness condition for $\mathcal{U}$-derivable clauses.

EXAMPLE **4.4.**
Consider the program P in Example 1.1. The completeness condition is clearly verified by the typical Prolog's search strategy. For instance (see Figure 2 and Example 4.2), if clause

$$(\texttt{positive\_list\_sum(X,Y)} \leftarrow \texttt{B}; \langle \texttt{1} \rangle :: \langle \texttt{3} \rangle)$$

where $\texttt{B} = \texttt{number(X')}, \texttt{list\_of\_numbers(Xs')}, \texttt{sum([X'} \mid \texttt{Xs'],Y)}, \texttt{positive(Y)}$, *is applicable to the goal* `positive_list_sum(X,Y)`, *then both clause (1) and (3) are applicable to the goals* `positive_list_sum(X,Y)` *and* `list_of_numbers(X), sum(X,Y), positive(Y)` *respectively. The completeness of the* prolog *oracle will be formally discussed in Section 5.*

Example **4.5**.
Consider the program P from Example 4.3, and define an oracle f for P as follows. For singleton interpretations we define

$$f(\{(p \leftarrow r;\pi)\}, G) = (p \leftarrow r;\pi)$$
$$f(\{(c;\pi)\}, G) = \emptyset$$

for every goal G and path π, and every clause c except `p` $\leftarrow$ `r`. *For other interpretations we define $f(\cup_j I_j, G) = \cup_j f(I_j, G)$ for every goal G.*

f is a downward closed oracle for P and is sound, but it is not complete with respect to unfolding. First note that $(p \leftarrow q;\langle 1\rangle)$ and $(q \leftarrow r;\langle 2\rangle)$ are $\mathcal{U}$-derivable clauses. Taking $(p \leftarrow q;\langle 1\rangle)$ for $(h \leftarrow B;\pi)$ and $(q \leftarrow r;\langle 2\rangle)$ for $(c;\pi')$ in the definition of oracle completeness, we find that $Unf^{slp}_{\{(c;\pi')\}}(\{(h \leftarrow B;\pi)\})$ contains $(p \leftarrow r;\langle 1\rangle :: \langle 2\rangle)$. Thus, considering the goal p (corresponding to $g \mid G$), $f(Unf^{slp}_{\{(c;\pi')\}}(\{(h \leftarrow B;\pi)\}), p) \neq \emptyset$. However both $f(\{(p \leftarrow q;\langle 1\rangle)\}, p)$ and $f(\{(q \leftarrow r;\langle 2\rangle)\}, q)$ are empty. Hence f is not complete.

In the following we will assume sound and complete oracles when not specified otherwise.

While for soundness we require the introduction of $lwr(\Phi)$ to the semantics, for completeness we must consider, for each path, only the most general partial answers. It is precisely because of the presence of paths that we are able to provide in this way the completeness result.
The following lemma shows that any clause in $lfp(\mathcal{U}^f_P)$ can be viewed as an instance of a clause obtained by iterating unfolding on a tautological clause in Φ.

Lemma **4.8**.
Let P be a program, f be a corresponding oracle and $(c;\pi) \in Unf^f_P{\uparrow}n(lwr(\Phi))$. Then $(c;\pi) \in lwr(\{(c';\pi)\})$ for some $(c';\pi) \in Unf^{slp}_P{\uparrow}n(\Phi)$.

The following lemma ensures that for any atomic goal h and path π there exists a *most general partial answer* with path π in $\mathcal{P}^f(lfp(\mathcal{U}^f_P), h)$.

Lemma **4.9**.
Let h be an atomic goal, π be a path such that $|\pi| = n$ and $D = \left\{ h' \,\middle|\, (h';\pi) \in \mathcal{P}^f(lfp(\mathcal{U}^f_P), h) \right\}$. If $D \neq \emptyset$ then $lub_{\leq} D \in D$ and $lub_{\leq} D = h\vartheta$ where $\vartheta = mgu(h, h')$ for $(h' \leftarrow B';\pi) \in Unf^f_P{\uparrow}n(\{h \leftarrow h;\Lambda\})$.

It is straightforward to extend the previous lemma to possibly non atomic goals, as stated in the following corollary

Corollary **4.10**.
Let G be a goal, π be a path and $D = \left\{ G' \,\middle|\, (G';\pi) \in \mathcal{P}^f(lfp(\mathcal{U}^f_P), G) \right\}$. If $D \neq \emptyset$ then $lub_{\leq} D \in D$.

Let P be a program and f be a corresponding oracle. We denote by $mgp^f(P,G)$ the set of *most general f-partial answers* (with paths) for G in $lfp(\mathcal{U}^f_P)$. Notice that $mgp^f(P,G) \subseteq \mathcal{P}^f(lfp(\mathcal{U}^f_P), G)$.

We can now prove the following completeness result for the oracle semantics.

Theorem **4.11** (completeness).
Let P be a program, f an oracle for P, and G a goal. If $G\vartheta$ is the most general f-partial answer of G with path π in $lfp(\mathcal{U}_P^f)$ and G'' is the corresponding resolvent then $G \xrightarrow[\pi]{f}{}^*_P G'$ with associated substitution ϑ', $G\vartheta \sim G\vartheta'$ and $G'' \sim G'$.

Combining this theorem with Theorem 4.7 and Corollary 4.10 we have the following characterization of answers and partial answers computed with the oracle f in terms of $\mathcal{U}_P^f$. Recall that we assume f is downward-closed, and sound and complete with respect to unfolding.

Theorem **4.12**.
Let P be a program and f an oracle for P. Let G, G', G'' be goals and ϑ, ψ be substitutions. Suppose $G \xrightarrow[\pi]{f}{}^*_P G'$ with associated substitution ϑ and suppose $G\psi$ is the most general f-partial answer of G with path π in $lfp(\mathcal{U}_P^f)$ and G'' is the corresponding resolvent. Then $G\vartheta \sim G\psi$ and $G' \sim G''$.

4.3 Bottom-up oracle semantics

Top-down constructions typically unfold clauses from an interpretation (a set of partial computations) with program clauses. In contrast, bottom-up constructions involve the unfolding of program clauses with clauses from an interpretation ([Levi 88]). The two approaches correspond to top-down and bottom-up constructions of the computation trees. A bottom-up oracle semantics is derived by switching the role of the program and the interpretation in Definition 4.5. In this case we are able to specify directly a semantics for successful computations. Hence, in the bottom-up construction each step proceeds by unfolding the atoms in the body of a program clause $h \leftarrow b_1, ..., b_n$ with a successful computation (unit clauses from an interpretation) for each b_i. The oracle, in this case, has two different functionalities: (1) to specify which program clauses should be unfolded; and (2) to specify which computations it should be unfolded with. In addition, left-to-right selection of atoms is modeled by performing the unfolding in a sequence from left-to-right.
The following definition introduces the oracle-based bottom-up immediate consequence operator. The operator T_P^f constructs the success set in a bottom-up way by applying applicable successful computations in I to the bodies of program clauses. Notice that not all the clauses can contribute to the semantics; there are clauses never chosen by the oracle, and clauses which can be chosen only under further instantiation of the variables in the head. To model this behaviour, we consider clauses in $lwr(P)$.

Definition **4.10** (bottom-up oracle semantics).
Let P be a logic program ordered textually by ℓ, let f be an oracle for P and I an interpretation. We define

$$T_P^f(I) = \left\{ [(h \leftarrow true; \bar{\pi}_n)\sigma_n]_\sim \;\middle|\; \begin{array}{l} C \equiv (h \leftarrow b_1, \ldots, b_n; l) \in f(lwr(P), h), \\ \sigma_0 = \varepsilon,\ \bar{\pi}_0 = \langle l \rangle \text{ and for each } 1 \le i \le n\text{:} \\ \quad (h_i \leftarrow true; \pi_i) \ll_{C\sigma_{i-1}} f(I, (b_i, ..., b_n)\sigma_{i-1}), \\ \quad \sigma_i = \sigma_{i-1} mgu(b_i\sigma_{i-1}, h_i),\ \bar{\pi}_i = \bar{\pi}_{i-1} :: \pi_i \end{array} \right\}.$$

Proposition **4.13**.
T_P^f is continuous.

The next result provides the equivalence of top-down and bottom-up oracle semantics for successful computations. To prove the claim we need the following lemmata.

LEMMA **4.14** (AND-compositionality).
Let P be a program, $G = b_1, ..., b_n$ be a goal and f be an oracle for P. $\langle b_1, ..., b_n\rangle \xrightarrow[\pi]{f}{}^{*}_{P} true$ is a refutation for G with substitution ϑ iff

1. *for each $1 \leq i \leq n$: ϑ_i and π_i are the substitution and path associated with the refutation $(b_i)\vartheta_1...\vartheta_{i-1} \xrightarrow[\pi_i]{f}{}^{*}_{P} true$ for the goal $(b_i)\vartheta_1...\vartheta_{i-1}$;*
2. *$\pi = \pi_1 :: \cdots :: \pi_n$;*
3. *$f(\{(b_i \leftarrow true)\vartheta_1...\vartheta_i; \pi_i)\}, (b_i, ..., b_n)\vartheta_1...\vartheta_{i-1}) \neq \emptyset$; and*
4. *$(b_1, ..., b_n)\vartheta \sim (b_1, ..., b_n)\vartheta_1...\vartheta_n$.*

The following lemma is a consequence of the oracle completeness.

LEMMA **4.15**.
Let P be a program, f be an oracle for P, $h \in Atom$ and $a \mid G$ be a goal. If $h \xrightarrow[\pi]{f}{}^{*}_{P} B$ with substitution ϑ, $f(\{(h \leftarrow B; \pi)\vartheta\}, a \mid G) \neq \emptyset$ and $mgu(h\vartheta, a) \neq fail$ then $a \mid G \xrightarrow[\pi]{f}{}^{*}_{P} (B :: G)\vartheta mgu(h\vartheta, a)$.

THEOREM **4.16**.
Let P be a program and f an oracle for P. Then,

$$lfp(T_P^f) = \left\{ (h\vartheta \leftarrow true; \pi) \;\middle|\; \begin{array}{l} h \xrightarrow[\pi]{f}{}^{*}_{P} true \\ \text{with substitution } \vartheta \end{array} \right\}.$$

Using the previous theorem and the soundness and completeness of the top-down fixpoint semantics (Theorem 4.12) we can show that the top-down and bottom-up fixpoint semantics are compatible.

COROLLARY **4.17**.
Let P be a program and f an oracle for P. Then,

$$lfp(T_P^f) = \left\{ (h \leftarrow true; \pi) \;\middle|\; (h \leftarrow true; \pi) \in lfp(\mathcal{U}_P^f) \right\}.$$

Consequently, oracle based-computations can be characterized by means of both a top-down and bottom-up semantic constructions.

COROLLARY **4.18**.
Let P be a program, f an oracle for P and G a goal. $G \xrightarrow[\pi]{f}{}^{*}_{P} true$ with associated substitution ϑ iff $G\vartheta$ is the most general f-answer of G with path π in $lfp(T_P^f)$.

5 Applications

In this section we introduce two examples of oracles: one for *slp*-programs and one for Prolog programs.

5.1 The oracle for *slp*

For *slp*-programs the order of clauses does not affect the computation. Thus any clause can be applied to a given goal provided that unification succeeds. This behaviour is captured by the oracle $slp = \lambda I.\lambda G.I$ returning the whole set of clauses. It is easy to see that *slp* is downward-closed and sound and complete with respect to unfolding.
The declarative semantics defined as the set of most general answers in $\alpha(lfp(T_P^{slp}))$ corresponds precisely to the semantics in [Falaschi, Levi, Martelli and Palamidessi 89] of computed answer substitutions. This equivalence is based on the observation that in *slp*-computations the Lifting Lemma holds [Lloyd 87]; i.e. it cannot be the case that a clause is not applicable while some of its instantiations are. Thus, applying all the instances (by lwr) of a clause and taking the most general computations corresponds to simply applying the clause itself. Similarly, the multiset semantics of [Maher and Ramakrishnan 89] computed by NSN evaluation can be extracted from $lfp(T_P^{slp})$.

5.2 The oracle for Prolog

The Prolog oracle is defined, as in the operational case, in terms of the logic programming semantics with left-to-right selection. Namely, the declarative Prolog oracle is defined in terms of the set of partial computations with paths as given by $lfp(\mathcal{U}_P^{slp})$. Each interpretation is totally ordered with respect to the *lexicographic ordering* $\preceq$ on paths. This ordering is adequate to declaratively specify a depth-first search strategy on the proof tree. In the following examples a sequence of n elements is denoted $a_1 \cdot a_2 \cdot \ldots \cdot a_n$, and a^n denotes $a \cdot a \cdot \ldots \cdot a$ with n occurrences of a.

EXAMPLE **5.1.**
Consider the Prolog program P:

```
q(x,y) ← p(x,y).  % (1)
p(a,b).           % (2)
p(x,y) ← p(x,y).  % (3)
p(x,x).           % (4)
```

$$lfp(\mathcal{U}_P^{slp}) = lwr\left(\left\{\begin{array}{l} (p(x,y) \leftarrow p(x,y); 3^{n+1}), \\ (p(a,b); 3^n \cdot 2);\ (p(x,x); 3^n \cdot 4); \\ (q(x,y) \leftarrow p(x,y); 1 \cdot 3^n), \\ (q(a,b); 1 \cdot 3^n \cdot 2); (q(x,x); 1 \cdot 3^n \cdot 4) \end{array} \;\middle|\; n \geq 0 \right\}\right).$$

The corresponding partial answers (for an arbitrary goal) are totally ordered by the lexicographical ordering on paths. Consider the successful computation for the goal $q(x,y)$: $(q(x,x); 1 \cdot 3 \cdot 4)$. The partial computations to the "left" include for instance: $(q(a,b); 1 \cdot 3^n \cdot 2)$ for $n \geq 0$. The only computation "to the the right" of $(q(x,x); 1 \cdot 3 \cdot 4)$ is $(q(x,x); 1 \cdot 4)$. By the soundness and completeness of the semantics, this behaviour is modeled by the most general partial answers in $lfp(\mathcal{U}_P^{slp})$.

Intuitively, if P is a Prolog program and $C \equiv (h \leftarrow b_1, ..., b_n; \pi)$ represents a partial computation with path π from the goal h to the resolvent $b_1, ..., b_n$; then the declarative Prolog oracle for

P should specify that the clause $C' \equiv (h' \leftarrow B'; l')$ can be applied to the goal $b_1, ..., b_n$ iff the computation which applies any of the previous clauses universally terminates. This is formalized by introducing the notion of *most general partial answers (mgp)* which characterizes the set of most general slp-partial answers for a goal G in $lfp(\mathcal{U}_P^{slp})$ that have paths to the left of a path π

$$mgp_\pi^{slp}(P,G) = \left\{ ([G']_\sim; \pi') \,\middle|\, \begin{array}{l} ([G']_\sim; \pi') \in mgp^{slp}(P,G) \\ \pi' \prec \pi \end{array} \right\}.$$

We can use slp-partial answers instead of $prolog$-partial answers because the universal termination behaviour of Prolog programs is identical to the universal termination behaviour of sequential logic programs. Thus, we can give the following

Definition **5.1** (declarative Prolog oracle).
Let P be a program, $I \in Int$ and G be a goal. Then,

$$prolog_P(I,G) = \left\{ (c,\pi) \in I \,\middle|\, mgp_\pi^{slp}(P,G) \text{ is finite } \right\}$$

As usual, when clear from the context the subscript P is omitted.

Example **5.2**.
Consider the Prolog program P in Example 5.1. The fourth clause cannot be applied to the goal $p(x,y)$ because:

$$\forall n \geq 0: \quad (p(a,b); 3^n \cdot 2), (p(x,x); 3^{n+1} \cdot 4) \in mgp_{\langle 4 \rangle}^{slp}(P, q(x,y)).$$

We now prove that $prolog$ is an oracle in its declarative meaning (i.e. an additive lower closure which is downward closed, sound and complete). It is straightforward to prove that $\lambda I.prolog(I,G)$ is idempotent, additive and reductive. Moreover

Proposition **5.1**.
The function prolog is a downward closed oracle.

Proposition **5.2**.
The oracle prolog is sound and complete with respect to unfolding.

The equivalence between the declarative and the operational definitions of the Prolog oracle is a consequence of the following theorem.

Theorem **5.3**.
Let P be a program textually ordered by ℓ, $C \in P$, $Q = \left\{ C' \,\middle|\, \ell(C') < \ell(C) \right\}$, and G be a goal. Then, $G \xrightarrow{slp}_Q G' \;\Rightarrow\; T(\Psi_P^{slp}, G')$ is finite iff $mgp_{\langle \ell(C) \rangle}^{slp}(P,G)$ is finite.

Example **5.3**.
Consider the Prolog program P in Example 5.1. The only goal G such that $T(\Psi_P^{slp}, G)$ is finite is $G = true$. Thus the evaluation of $lfp(\mathcal{U}_P^{prolog})$ proceeds as follows, for each $n \geq 0$:

$$(T_P^{prolog})^n(\emptyset) = \left\{ \begin{array}{l} (p(a,b); 3^m \cdot 2) \\ ((p(x,y) \leftarrow p(x,y))\sigma; 3^m), \\ (q(a,b); 1 \cdot 3^m \cdot 2), \\ ((q(x,y) \leftarrow p(x,y))\sigma; 1 \cdot 3^m) \end{array} \,\middle|\, \begin{array}{l} m \leq n, \\ \sigma \text{ substitution} \end{array} \right\}.$$

Since the only clause for p with an empty body in $lfp(\mathcal{U}_P^{prolog})$ is $p(a,b)$, we can argue that the (Prolog) success set for p in P is $\{p(a,b)\}$.

EXAMPLE **5.4**. *Consider the Prolog program P:*

```
p(a,b).                      % (1)
p(c,b).                      % (2)
p(x,y) ← p(y,x).             % (3)
p(x,y) ← p(x,z),p(z,y).      % (4)
```

This program is used by Lloyd [Lloyd 87] to illustrate the incompleteness of Prolog's search strategy. The claim is that: "no matter how the clauses of P are ordered and no matter what the computation rule, a depth-first search with fixed order of program clauses will never find a solution for $p(a,c)$". The Prolog semantics is:

$$lfp(\mathcal{U}_P^{prolog}) = \left\{ \begin{array}{l} (p(a,b); 3^{2n} \cdot 1), (p(c,b); 3^{2n} \cdot 2) \\ (p(b,a); 3^{2n+1} \cdot 1), (p(b,c); 3^{2n+1} \cdot 2) \end{array} \;\middle|\; n \geq 0 \right\},$$

with a corresponding success set for P: $\{p(a,b), p(c,b), p(b,a), p(b,c)\}$. Because clauses (3) and (4) have completely general heads, they match with any partial computation. Thus if (3) is before (4) in the program, the system will never consider (4) and viceversa [Lloyd 87]. This behaviour is captured in our semantics construction. In particular, for each partial computation $G = p(t_1, t_1'), ..., p(t_n, t_n')$, the clause $(p(x,y) \leftarrow p(x,z), p(z,y); 4) \notin prolog(P,G)$ because, $\forall n \geq 0$ we have $(p(x,y) \leftarrow p(x,y); 3^{2n}), (p(x,y) \leftarrow p(y,x); 3^{2n+1}) \in lfp(\mathcal{U}_P^{slp})$. Thus $\forall n \geq 0$

$$(p(t_1, t_1'), ..., p(t_n, t_n'); 3^{2n}), (p(t_1', t_1), ..., p(t_n, t_n'); 3^{2n+1}) \in mgp_{\langle 4 \rangle}^{slp}(P,G).$$

A similar behaviour can be verified by changing the order of clauses 3 and 4:

```
p(a,b).                      % (1)
p(c,b).                      % (2)
p(x,y) ← p(x,z),p(z,y).      % (3)
p(x,y) ← p(y,x).             % (4)
```

where $\forall n \geq 1$: $(p(x,y) \leftarrow p(x,z_n), ..p(z_n, z_{n-1}).., p(z_1, y)); 3^n) \in lfp(\mathcal{U}_P^{slp})$. As before, the clause $(p(x,y) \leftarrow p(y,x); 4)$ cannot be applied to any partial computation.

5.3 Approximations

In the previous section we have shown how to capture declaratively the semantics of a Prolog program P by enhancing the standard semantics for pure logic programs with an oracle. Of course it is not possible in general to provide an oracle which corresponds precisely to *prolog*. However, since we are interested to provide a semantic basis for the analysis of Prolog programs, for this purpose approximations of *prolog* both from *above* and from *below* can be useful. A similar argument has been considered in [Barbuti, Codish, Giacobazzi and Levi 92] and [Barbuti, Codish, Giacobazzi and Levi 93], which also provides a semantic basis for Prolog program analysis. By Theorem 4.6 we can distinguish between two possible approximations for the *prolog* oracle:

- from above: let f_a be an oracle such that for each goal G: $\lambda I.f_a(I,G) \supseteq \lambda I.prolog(I,G)$. In this case we have $lfp(\mathcal{U}_P^{prolog}) \subseteq lfp(\mathcal{U}_P^{f_a})$, hence providing an approximation from *above* of the (Prolog) success set for P. This may provide better approximations than can be achieved when abstracting the meaning of P as a logic program (i.e. with the oracle *slp*).

- <u>from below:</u> let f_b be an oracle such that for each goal G: $\lambda I.f_b(I,G) \subseteq \lambda I.prolog(I,G)$. In this case we have $lfp(\mathcal{U}_P^{f_b}) \subseteq lfp(\mathcal{U}_P^{prolog})$, hence providing an approximation from *below* of the (Prolog) success set for P which can be useful for example in the context of the complexity analysis described in [Debray and Lin 91]; or to estimate a lower bound to the number of solutions of P.

Figure 3 illustrates the computation tree for a logic program P and a goal G for a left-to-right selection rule in which the left most infinite branch is indicated. The solutions for G correspond to the success branches of the entire tree. These can be derived by considering the trivial oracle *slp*. The solutions for G assuming Prolog's depth-first strategy are those which are to the left of the infinite branch. These can be derived with *prolog*. Approximations of the Prolog solutions for G from above and below can be derived by any oracles f_a and f_b which, respectively, approximate *prolog* from above and from below.

Approximation from below corresponds to the use of knowledge concerning definite (universal) termination of subgoals, whereas approximation from above corresponds to the use of knowledge concerning definite non-termination of subgoals. A survey of techniques for establishing termination and non-termination is given in [De Schreye and Decorte 94].

EXAMPLE **5.5.**
In this example we consider approximations from below and above in the dataflow analysis of variable aliasing *for a simple logic program. Consider the following fragment of a Prolog program*

```
p(a,b) ← q(b).          % (1)
p(X,f(Y)) ← q(f(Y)).   % (2)
p(X,X).                 % (3)
```

with the hypothesis that goal `q(b)` *definitely terminates (terminates universally), while goal* `q(f(Y))` *may have some solutions but it is not definitely terminating. It is easy to see that an approximation from below, which collects computations from definitely terminating goals only, concludes that ground substitutions are computed for the goal* `p(X,Y)`*. By approximating from above, namely by discarding inapplicable computations, we can observe variable independence between* `X` *and* `Y`*. This because clause (3) cannot be applied to the goal. With the straightforward oracle slp instead, we lose precision by observing also a variable aliasing between* `X` *and* `Y`*, which is introduced by clause (3). The latter aliasing is not computed by a Prolog interpreter.*

6 Conclusions

We have presented an oracle approach for specifying the semantics of pure Prolog programs. The key idea is to separate the notions of logic and control in the semantic definition. This approach results in simple and concise semantic definitions which are useful in proving properties of programs and in the context of semantic based techniques for analysis and debugging. By factoring the semantics into two components, we provide a semantic foundation to combine results on termination analysis (e.g. see [Apt, Bol and Klop 89], [Apt and Pedreschi 93], [Smith, Genesereth and Ginsberg 86], [Verschaetse and De Schreye 91], and

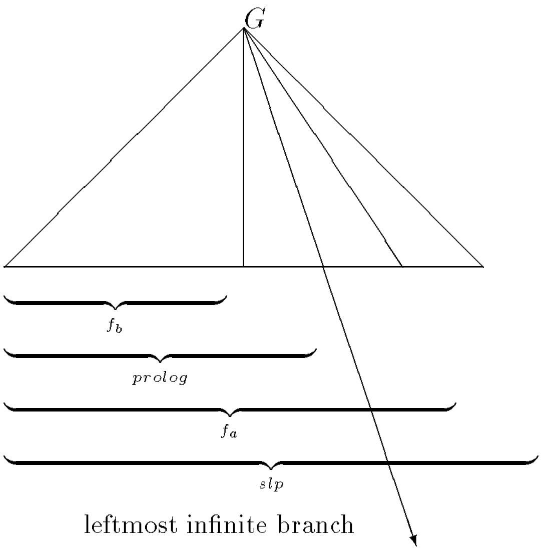

Figure 3: SLD-tree for G with P

[De Schreye and Decorte 94]) with semantics-based analysis (e.g. abstract interpretation, type inference).

We believe that the oracle approach is of general value in specifying the semantics of nondeterministic languages which are implemented in terms of a specific control strategy. In such cases, our technique can produce a semantics for the implementation in terms of the more natural semantics of the nondeterministic language. For example, the approach could be applied to depth-first implementations of such well-known models of computation as nondeterministic automata and nondeterministic Turing machines. Similarly, the technique applies to grammar formalisms, where the top-down and bottom-up semantics correspond to parsing and sentence generation respectively. Of greater practical interest are current implemented languages to which this technique applies. They include constraint logic programming languages [Jaffar and Lassez 87, Jaffar and Maher 94], set-based languages [Jayaraman 92], and equational languages [Jayaraman 89], among others. The technique also applies to a subset of the string processing languages SNOBOL [Griswold, Poage and Polonsky 71] and Icon [Griswold 93]. However it does not apply to the full languages, nor to full Prolog, because of the existence of constructs which allow the execution of one branch to affect another (non-backtrackable assignment in SNOBOL and Icon, *assert* and *retract* in full Prolog).

There remain open questions whether and/or how the oracle approach can be applied to other aspects of control in logic programming languages such as *negation-as-failure*, *once* and *cut* in Prolog; dynamic scheduling as in MU-Prolog [Naish 86] and languages based on the Andorra principle [Haridi and Janson 90]); and extra-logical predicates such as *var* and *nonvar* in various logic languages.

Finally, we note that our approach may contribute to the derivation of parallel implementations (e.g. of Prolog). To see this consider that an implementation can be viewed as a strategy for constructing a corresponding computation tree and that information about an oracle's decisions may be applied to identify subtrees which can be fruitfully constructed in parallel and subtrees which need not be constructed.

Acknowledgment

This work has been partly supported by "Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo" of C.N.R. under grant n. 9100880.PF69 and by the Esprit Basic Research Action 3012 - Compulog I. The stimulating discussion with E. Börger, G. Levi and D. Rosenzweig are gratefully acknowledged. This work was done while M. Codish and M. J. Maher were visiting the Università di Pisa. They thank the Dipartimento di Informatica for its hospitality.

A Appendix: proofs

PROPOSITION **4.1.** [$\sim$-congruence]
Let f be a downward closed oracle for a program. Then, for any interpretation I and goals G, G': $G \sim G' \quad \Rightarrow \quad f(I,G) = f(I,G')$.

PROOF.
The proof is straightforward by downward closure of the oracle. ∎

PROPOSITION **4.2.**
Let P be a program and f be a downward closed oracle for P. If $G \xrightarrow[\pi]{f}{}^{*}_{P} G'$ with associated substitution ϑ, then $G\vartheta \xrightarrow[\pi]{f}{}^{*}_{P} G''$ and $G'' \sim G'$.

PROOF.
By downward closure of the oracle, the proof boils down into the standard one for logic programs (e.g. see [Apt 90]). ∎

LEMMA **4.4.**
Let P be a program and f be an oracle. If $[C]_\sim \in lfp(\mathcal{U}_P^f)$ then $lwr(\{[C]_\sim\}) \subseteq lfp(\mathcal{U}_P^f)$.

PROOF. It is a straightforward consequence of the definition and of the downward closure of the oracle. ∎

LEMMA **4.5.**
Let P be a program, f be an oracle and $(c;\pi) \in \mathcal{U}_P^f \uparrow \omega$. Then $|\pi| = n$ iff $(c;\pi) \in Unf_P^f \uparrow n(lwr(\Phi))$.

PROOF.
The proof is a straightforward induction on n. ∎

THEOREM **4.6.**
Let P be a program and f, g be oracles for P. If for each goal G: $\lambda I.f(I,G) \subseteq \lambda I.g(I,G)$ then $lfp(\mathcal{U}_P^f) \subseteq lfp(\mathcal{U}_P^g)$.

PROOF. Let G be a goal and I be an interpretation. Let also $f(I,G) \subseteq g(I,G)$. We prove that $Unf_P^f(I) \subseteq Unf_P^g(I)$. If $(k \leftarrow B''; \pi :: \langle l \rangle) \in Unf_P^f(I)$ then there exists $C = (h \leftarrow b \mid B; \pi) \in I$ and $(h' \leftarrow B'; l) \ll_C f(P, b \mid B)$ such that $k = h\sigma$, $B'' = (B' :: B)\sigma$ and $\sigma = mgu(b,h')$. By the hypothesis: $(h' \leftarrow B'; l) \ll_C g(P, b \mid B)$, from which follows the thesis. Thus, by definition: $\mathcal{U}_P^f(I) \subseteq \mathcal{U}_P^g(I)$. The rest of this proof follows by a straightforward inductive argument. ∎

THEOREM 4.7. (soundness)
Let P be a program, f an oracle for P, G a goal and ϑ the substitution associated with $G \xrightarrow[\pi]{f}{}^*_P G'$. Then, $G\vartheta$ is an f-partial answer of G with path π in the interpretation $lfp(\mathcal{U}_P^f)$ and $G' \sim G''$, where G'' is the corresponding resolvent with path π in $lfp(\mathcal{U}_P^f)$.

PROOF.
The proof is by induction on the length m of the computation. Let $G = a_1, ..., a_n$.

Base case: by definition:

$$G \xrightarrow[l]{f}_P (B :: \langle a_2, ..., a_n \rangle)\vartheta$$

where $(h \leftarrow B; l) \in f(P, G)$ and $\vartheta = mgu(h, a_1)$.

By downward closure $(h \leftarrow B; l) \in f(P, a_1)$. $(a_1 \leftarrow a_1; \Lambda) \in lwr(\Phi)$ and so, by $\mathcal{U}_P^f$ definition, $((h \leftarrow B)\vartheta; l) \in \mathcal{U}_P^f(\emptyset) \subseteq lfp(\mathcal{U}_P^f)$.

By downward closure we have: $(h \leftarrow B; l) \in f(P, G) \Rightarrow ((h \leftarrow B)\vartheta; l) \in f(P\vartheta, G)$.

By additivity (notice that $((h \leftarrow B)\vartheta; l) \in P\vartheta \cap lfp(\mathcal{U}_P^f))$ we have:

$$((h \leftarrow B)\vartheta; l) \in f(lfp(\mathcal{U}_P^f), G).$$

Inductive case: assume the inductive hypothesis for lengths $\leq m$ and let $k \in \{1, ..., n\}$ such that

$$G \xrightarrow[\pi]{f}{}^{m_1}_P (\langle a_k, a_{k+1}, ..., a_n \rangle)\psi \xrightarrow[\pi]{f}{}^{m_2}_P (b|B :: \langle a_{k+1}, ..., a_n \rangle)\psi\alpha \xrightarrow[l]{f}_P (W :: B :: \langle a_{k+1}, ..., a_n \rangle)\vartheta\sigma$$

where $m \geq 1$, $m = m_1 + m_2$, $\vartheta = \psi\alpha$, $(h \leftarrow W; l) \in f(P, (b \mid B :: \langle a_{k+1}, ..., a_n \rangle)\vartheta)$ and $\sigma = mgu(h, b\vartheta)$.

By the inductive hypothesis, for $1 \leq i \leq k-1$ there exist

$$(h_i \leftarrow true; \pi_i) \ll_{G\vartheta_{i-1}} f(lfp(\mathcal{U}_P^f), (a_i, ..., a_n)\vartheta_{i-1})$$

where $\vartheta_i = \vartheta_{i-1} mgu(a_i\vartheta_{i-1}, h_i)$, $\pi = \pi_1 :: \cdots :: \pi_k$ and

$$(h_k \leftarrow b' \mid B'; \pi_k) \ll_{G\vartheta_{k-1}} f(lfp(\mathcal{U}_P^f), (a_k, ..., a_n)\vartheta_{k-1})$$

such that $(b|B :: \langle a_{k+1}, ..., a_n \rangle)\vartheta \sim (b'|B' :: \langle a_{k+1}, ..., a_n \rangle)\vartheta_k$, $(a_k, ..., a_n)\psi \sim (a_k, ..., a_n)\vartheta_{k-1}$ and $G\vartheta \sim G\vartheta_k$.

Thus, by downward closure ($\sim$-congruence):

$$(h \leftarrow W; l) \in f(P, (b' \mid B' :: \langle a_{k+1}, ..., a_n \rangle)\vartheta_k).$$

Moreover, by Lemma 4.4 and by part 3 of downward closure:

$$(h_k \leftarrow b' \mid B'; \pi_k)\vartheta_k \ll f(lfp(\mathcal{U}_P^f), (a_k, ..., a_n)\vartheta_{k-1}).$$

Let $\sigma' = mgu(h, b'\vartheta_k)$. Thus, by $\mathcal{U}_P^f$ definition:

$$(h_k \leftarrow W :: B'; \pi_k :: \langle l \rangle)\vartheta_k\sigma' \ll_{G\vartheta_{k-1}} lfp(\mathcal{U}_P^f).$$

We have now to prove that the clause $(h_k \leftarrow W :: B'; \pi_k :: \langle l \rangle)\vartheta_k\sigma$ is also applicable to the goal $(a_k, ..., a_n)\vartheta_{k-1}$. From the last $m_2 + 1$ transitions in the given derivation and the soundness of the oracle we have

$$f(\{(a_k\psi \leftarrow W :: B; \pi_k :: \langle l \rangle)\alpha\sigma\}, (a_k, ..., a_n)\psi) \neq \emptyset.$$

By manipulation of substitutions and $\sim$-congruence of f

$$f(\{(h_k \leftarrow W :: B'; \pi_k :: \langle l \rangle)\vartheta_k\sigma'\}, (a_k, ..., a_n)\vartheta_{k-1}) \neq \emptyset.$$

Thus, by additivity:

$$(h_k \leftarrow W :: B'; \pi_k :: \langle l \rangle)\vartheta_k\sigma' \ll f(lfp(\mathcal{U}_P^f), (a_k, ..., a_n)\vartheta_{k-1}).$$

Through further manipulation of substitutions, using equivalences given by the inductive hypothesis, we can show that

$$G\vartheta_{k-1}mgu(h_k\vartheta_k\sigma', a_k) \sim G\vartheta_k\sigma' \sim G\vartheta\sigma$$

and

$$(W :: B :: \langle a_{k+1}, ..., a_n \rangle)\vartheta\sigma \sim (W :: B' :: \langle a_{k+1}, ..., a_n \rangle)\vartheta_k\sigma'$$

This concludes the proof.

■

We now introduce a property of substitutions that will be used in the following.

Lemma **A.1**. *([Palamidessi 90])*
Let $\vartheta_1, \vartheta_2 \in Sub$. Then $mgu(\mathcal{E}(\vartheta_1) \cup \mathcal{E}(\vartheta_2)) = \vartheta_1 mgu(\mathcal{E}(\vartheta_2)\vartheta_1) = \vartheta_2 mgu(\mathcal{E}(\vartheta_1)\vartheta_2)$.

Lemma **4.8.**
Let P be a program, f be a corresponding oracle and $(c; \pi) \in Unf_P^f\!\uparrow\! n(lwr(\Phi))$. Then $(c; \pi) \in lwr(\{(c'; \pi)\})$ for some $(c'; \pi) \in Unf_P^{slp}\!\uparrow\! n(\Phi)$.

Proof. The proof is by induction on the power of Unf_P^f.

Base case: is straightforward by lwr definition.

Inductive case: let $(c; \pi) \in Unf_P^f\!\uparrow\! n + 1(lwr(\Phi))$, then, by definition:

$$\begin{array}{l} (c; \pi) = (h \leftarrow B' :: B; \pi' :: \langle l \rangle)\vartheta \\ (h \leftarrow b \mid B; \pi') \in Unf_P^f\!\uparrow\! n(lwr(\Phi)) \\ (h' \leftarrow B'; l) \in f(P, b \mid B) \ \text{ and } \vartheta = mgu(h', b) \end{array}$$

Thus, by the inductive hypothesis $(h \leftarrow b \mid B; \pi') = (\tilde{h} \leftarrow \tilde{b} \mid \tilde{B}; \pi')\sigma$ for some substitution σ such that $dom(\sigma) \subseteq vars(\tilde{h})$ and $(\tilde{h} \leftarrow \tilde{b} \mid \tilde{B}; \pi') \in Unf_P^{slp}\!\uparrow\! n(\Phi)$. Because $mgu(h', b) \neq fail$ there exists $\gamma = mgu(h', \tilde{b})$ such that $(\tilde{h} \leftarrow B' :: \tilde{B}; \pi' :: \langle l \rangle)\gamma \in Unf_P^{slp}\!\uparrow\! n + 1(\Phi)$.
Moreover, by Lemma A.1 and because $\mathcal{E}(mgu(\mathcal{E}(\sigma))) \approx \mathcal{E}(\sigma)$:

$$\begin{array}{rcl} (h \leftarrow B' :: B; \pi' :: \langle l \rangle)\vartheta & = & (h \leftarrow B' :: B; \pi' :: \langle l \rangle)mgu(\tilde{b}\sigma, h') \\ & = & (\tilde{h} \leftarrow B' :: \tilde{B}; \pi' :: \langle l \rangle)\sigma mgu(\tilde{b}\sigma, h') \\ & = & (\tilde{h} \leftarrow B' :: \tilde{B}; \pi' :: \langle l \rangle)mgu(\mathcal{E}(\sigma) \cup \{\tilde{b} = h'\}) \\ & = & (\tilde{h} \leftarrow B' :: \tilde{B}; \pi' :: \langle l \rangle)mgu(\tilde{b}, h')mgu(\mathcal{E}(\sigma)mgu(\tilde{b}, h')) \\ & = & (\tilde{h} \leftarrow B' :: \tilde{B}; \pi' :: \langle l \rangle)\gamma mgu(\mathcal{E}(\sigma)\gamma) \end{array}$$

Notice that $dom(mgu(\mathcal{E}(\sigma)\gamma)) \subseteq vars(\tilde{h}\gamma)$.

$\blacksquare$

LEMMA **4.9**.
Let h be an atomic goal, π be a path and $D = \left\{ h' \,\middle|\, (h';\pi) \in \mathcal{P}^f(lfp(\mathcal{U}_P^f), h) \right\}$. If $D \neq \emptyset$ then $lub_{\leq} D \in D$; and $lub_{\leq} D = h\ mgu(h, h')$ where $(h' \leftarrow B';\pi) \in Unf_P^f{\uparrow}|\pi|(\{h \leftarrow h; \Lambda\})$.

PROOF. Let $|\pi| = n$ and $D = \left\{ h' \,\middle|\, (h';\pi) \in \mathcal{P}^f(lfp(\mathcal{U}_P^f), h) \right\}$.
By Lemma 4.5 $D = D_n = \left\{ h' \,\middle|\, (h';\pi) \in \mathcal{P}^f(Unf_P^f{\uparrow}n(lwr(\Phi)), h) \right\}$. The proof is by induction on the length n of π.

Base case: let us assume $D_1 \neq \emptyset$. Then there exists a program clause $(h' \leftarrow B', l) \in f(P, h)$ such that $\pi = \langle l \rangle$ and

$$D_1 = \left\{ h''' \,\middle|\, \begin{array}{l} h''' = h\ mgu(h, h'\ mgu(h', h'')) \\ h'' \in Atom \end{array} \right\}$$

Thus, by definition $h\ mgu(h, h'\ mgu(h', h)) \in D_1$. Moreover, for any substitution ϑ

$$\begin{array}{rcl} h\ mgu(h, h'\ mgu(h', h)) & = & h\ mgu(h, h\ mgu(h', h)) \\ & = & h\ mgu(h', h) \\ & \geq & h\ mgu(h'\vartheta, h) \end{array}$$

therefore $h\ mgu(h', h) \in D_1$, and for each $h''' \in D_1$: $h\ mgu(h', h) \geq h'''$.

Inductive case: assume the inductive hypothesis for D_n and consider $h''' \in D_{n+1}$ ($D_{n+1} \neq \emptyset$). Then, by definition there exist:

- $C = (h' \leftarrow b' \mid B';\pi') \ll_h Unf_P^f{\uparrow}n(lwr(\Phi))$ and
- $(k \leftarrow B; l) \ll_{h,C} f(P, b' \mid B')$ such that

$\pi = \pi' :: \langle l \rangle$, $f(\{(h' \leftarrow B :: B';\pi) mgu(b', k)\}, h) \neq \emptyset$ and $h''' = h\ mgu(h, h'\ mgu(b', k))$.

By the oracle completeness: $f(\{(h' \leftarrow b' \mid B';\pi')\}, h) \neq \emptyset$, therefore $D_n \neq \emptyset$. By combining the inductive hypothesis and Lemma 4.8, there exists:

$$(\tilde{h} \leftarrow \tilde{b} \mid \tilde{B};\pi')\vartheta \ll_h Unf_P^f{\uparrow}n(\{h \leftarrow h; \Lambda\})$$

where $(\tilde{h} \leftarrow \tilde{b} \mid \tilde{B};\pi') \in Unf_P^{slp}{\uparrow}(\Phi)$ and $dom(\vartheta) \subseteq vars(\tilde{h})$, such that $f(\{(\tilde{h} \leftarrow \tilde{b} \mid \tilde{B}';\pi')\vartheta\}, h) \neq \emptyset$ and, for any substitution σ such that $dom(\sigma) \subseteq vars(\tilde{h})$ and $(\tilde{h} \leftarrow \tilde{b} \mid \tilde{B};\pi')\sigma \ll_h Unf_P^f{\uparrow}n(lwr(\Phi))$: $h\ mgu(h, \tilde{h}\vartheta) \geq h\ mgu(h, \tilde{h}\sigma)$.

Therefore: $h' \leftarrow b' \mid B' \leq (\tilde{h} \leftarrow \tilde{b} \mid \tilde{B})\vartheta$.

In particular, (because of Lemma 4.8) there exists σ' such that $h\ mgu(h, \tilde{h}\vartheta) \geq h\ mgu(h, \tilde{h}\sigma')$ and $(\tilde{h} \leftarrow \tilde{b} \mid \tilde{B};\pi')\sigma' = (h' \leftarrow b' \mid B';\pi')$. Thus, because $mgu(\tilde{b}\sigma', k) \neq fail$ then also $mgu(\tilde{b}\vartheta, k) \neq fail$. Therefore:

$$(\tilde{h} \leftarrow B :: \tilde{B};\pi)\vartheta mgu(\tilde{b}\vartheta, k) \ll_h Unf_P^{slp}{\uparrow}n+1(\{(h \leftarrow h;\Lambda)\})$$

and, by the downward closure: $f(\{(\tilde{h} \leftarrow B :: \tilde{B};\pi)\vartheta mgu(\tilde{b}\vartheta, k)\}, h) \neq \emptyset$. By the oracle completeness: $(k \leftarrow B; l) \ll_{h,C} f(P, (\tilde{b} \mid \tilde{B})\vartheta)$ and by definition:

$$(\tilde{h} \leftarrow B :: \tilde{B};\pi)\vartheta mgu(\tilde{b}\vartheta, k) \ll_h Unf_P^f{\uparrow}n+1(\{(h \leftarrow h;\Lambda)\})$$

which is also applicable to h.

Consider $\tilde{h}''' = h\ mgu(h, \tilde{h}\vartheta mgu(\tilde{b}_1\vartheta, k))$. Then because h (k) and $\tilde{h}$ $(\tilde{b}_1)$ are renamed apart from each other, and by Lemma A.1:

$$\begin{aligned} h\ mgu(h, \tilde{h}\vartheta mgu(\tilde{b}_1\vartheta, k)) &= h\ mgu(h, \tilde{h}mgu(\{\tilde{b}_1 = k\} \cup \mathcal{E}(\vartheta))) \\ &= h\ mgu(\{\tilde{b}_1 = k\} \cup \mathcal{E}(\vartheta))mgu((h = \tilde{h})mgu(\{\tilde{b}_1 = k\} \cup \mathcal{E}(\vartheta))) \\ &= h\ mgu(\{h = \tilde{h}, \tilde{b}_1 = k\} \cup \mathcal{E}(\vartheta)). \end{aligned}$$

By the inductive hypothesis it follows that: $\tilde{h}''' \geq h\ mgu(h, \tilde{h}\sigma' mgu(\tilde{b}_1\sigma', k)) = h'''$, i.e. there exists a most general partial answer for h in $\mathcal{U}_P^f\!\uparrow\! n+1$ with path $\pi' :: \langle l \rangle$.

■

THEOREM **4.11.** (completeness)
Let P be a program, f an oracle for P and G a goal. If $G\vartheta$ is the most general f-partial answer of G with path π in $lfp(\mathcal{U}_P^f)$ and G'' is the corresponding resolvent then $G \xrightarrow[\pi]{f}{}^*_P G'$ with associated substitution ϑ' and $G\vartheta \sim G\vartheta'$ and $G'' \sim G'$.

PROOF.
The proof is by induction on the power m of $\mathcal{U}_P^f$. Assume the hypothesis and let $G = a_1, ..., a_n$.

<u>Base case:</u> let $k \in \{1, ..., n\}$ such that for $1 \leq i \leq k$:

$$\begin{array}{l} (h_i \leftarrow B_i; l_i)\sigma_i \in f(Unf_P^f(lwr(\Phi)), (a_i, ..., a_n)\vartheta_{i-1}) \\ (h_i \leftarrow B_i; l_i) \in P \ \text{ and } B_i = true \text{ for } i < k \\ \vartheta_0 = \varepsilon, \quad \vartheta_i = \vartheta_{i-1}mgu(a_i\vartheta_{i-1}, h_i\sigma_i) \\ (a_i, ..., a_n)\vartheta_i \text{ is the most general } f\text{-partial answer with path } \langle l_1 ... l_i \rangle \\ \qquad \text{for } (a_i, ..., a_n)\vartheta_{i-1} \text{ in } \mathcal{U}_P^f\!\uparrow\!1 \end{array}$$

By definition and by Lemma 4.9: $\sigma_i = mgu(a_i\vartheta_{i-1}, h_i)$.
We prove by induction on i that for any $i \geq 1$:

$$\vartheta_i = mgu(\bigcup_{j=1}^{i} \{a_i = h_i\}).$$

<u>$i = 1$</u>: because $\vartheta_0 = \varepsilon$ then $\vartheta_1 = \sigma_1 = mgu(\{a_1 = h_1\})$
<u>$i > 1$</u>: assume the hypothesis for $i - 1$. By the inductive hypothesis and by Lemma A.1:

$$\begin{aligned} \vartheta_i &= \vartheta_{i-1}mgu(a_i\vartheta_{i-1}, h_i\sigma_i) \\ &= \vartheta_{i-1}mgu(a_i\vartheta_{i-1}, h_i mgu(a_i\vartheta_{i-1}, h_i)) \\ &= \vartheta_{i-1}mgu(a_i\vartheta_{i-1}, a_i\vartheta_{i-1}mgu(a_i\vartheta_{i-1}, h_i)) \\ &= \vartheta_{i-1}mgu(a_i\vartheta_{i-1}, h_i) \\ &= mgu(\mathcal{E}(\vartheta_{i-1}) \cup \{a_i = h_i\}) \\ &= mgu(\bigcup_{j=1}^{i} \{a_i = h_i\}). \end{aligned}$$

Thus, $\vartheta_i = \vartheta_i'$ for $1 \leq i \leq k$, and by the downward closure: $(h_i \leftarrow B_i; l_i) \in f(P, (a_i, ..., a_n)\vartheta_{i-1}')$ where $\vartheta_i' = \vartheta_{i-1}'mgu(a_i\vartheta_{i-1}', h_i)$, $\vartheta_0' = \varepsilon$. Thus, for $1 \leq i \leq k$:

$$(a_i, ..., a_n)\vartheta_{i-1}' \xrightarrow[l_i]{f}{}_P (B_i :: \langle a_{i+1}, ..., a_n \rangle)\vartheta_i'$$

and, since $B_i = true$ for $i < k$, we have:

$$(a_1, ..., a_n) \xrightarrow[l_i]{f} {}^{k}_{P} (B_k :: \langle a_{k+1}, ..., a_n \rangle)\vartheta'_k$$

and $\vartheta'_k = \vartheta'_{k-1} mgu(a_k \vartheta'_{k-1}, h_k)$.
Of course, $G\vartheta'_k \sim G\vartheta_k$ and $(B_k :: \langle a_{k+1}, ..., a_n \rangle)\vartheta'_k \sim (B_k :: \langle a_{k+1}, ..., a_n \rangle)\vartheta_k$.

Inductive case: let $G\vartheta$ be the most general partial answer in $\mathcal{U}^f_P \uparrow m+1$ with path π. Thus, by definition: $\mathcal{U}^f_P \uparrow m+1 = \mathcal{U}^f_P(\mathcal{U}^f_P \uparrow m)$, and there exists $k \in \{1, ..., n\}$ such that for each $1 \leq i \leq k$:

$$\begin{array}{l} (h_i \leftarrow B_i; \pi_i) \in f(\mathcal{U}^f_P(\mathcal{U}^f_P \uparrow m), (a_i, ..., a_n)\vartheta_{i-1}) \\ B_i = true \text{ for } i < k \\ \vartheta_0 = \varepsilon,\ \vartheta_i = \vartheta_{i-1} mgu(a_i \vartheta_{i-1}, h_i) \\ \pi = \pi_1 :: \cdots :: \pi_k \\ (a_i, ..., a_n)\vartheta_i \text{ is the most general } f\text{-partial answer with path } \pi_i \\ \qquad \text{for } (a_i, ..., a_n)\vartheta_{i-1} \text{ in } \mathcal{U}^f_P \uparrow m+1. \end{array}$$

By definition, for $1 \leq i \leq k$, we can have two cases:

1. if $|\pi_i| \leq m$ then, by Lemma 4.5: $(h_i \leftarrow B_i; \pi_i) \in f(\mathcal{U}^f_P \uparrow m, (a_i, ..., a_n)\vartheta_{i-1})$ and by the inductive hypothesis:

$$(a_i, ..., a_n)\vartheta_{i-1} \xrightarrow[\pi_i]{f} {}^{*}_{P} (B_i :: \langle a_{i+1}, ..., a_n \rangle)\vartheta_i$$

2. if $|\pi_i| = m+1$ then, by definition there exists $(\tilde{h} \leftarrow \tilde{b} \mid \tilde{B}; \tilde{\pi}) \in \mathcal{U}^f_P \uparrow m$ and $(w \leftarrow Q; l) \in f(P, \tilde{b} \mid \tilde{B})$ such that $(h_i \leftarrow B_i; \pi_i) \sim (\tilde{h} \leftarrow Q \mid \tilde{B}; \tilde{\pi} :: \langle l \rangle)\sigma$, where $\sigma = mgu(\tilde{b}, w)$.
Moreover: $f(Unf^f_{\{(w \leftarrow Q; l)\}}(\{(\tilde{h} \leftarrow \tilde{b} \mid \tilde{B}; \tilde{\pi})\}), (a_i, ..., a_n)\vartheta_{i-1}) \neq \emptyset$ and since $h_i \leq \tilde{h}$, f is reductive and $mgu(h_i, a_i) \neq fail$, then by the oracle completeness:
(a) $f(\{(\tilde{h} \leftarrow \tilde{b} \mid \tilde{B}; \tilde{\pi})\}, (a_i, ..., a_n)\vartheta_{i-1}) \neq \emptyset$
(b) $f(\{(w \leftarrow Q; l)\}, (\tilde{b} \mid \tilde{B} :: \langle a_{i+1}, ..., a_n \rangle)\vartheta_{i-1}\sigma) \neq \emptyset$.
It is straightforward to extend Lemma 4.9 and prove that because $(h_i \leftarrow B_i; \pi_i)$ provides the most general partial answer for $(a_i, ..., a_n)\vartheta_{i-1}$ with path π_i, then $(\tilde{h} \leftarrow \tilde{b} \mid \tilde{B}; \tilde{\pi})$ provides the most general partial answer with path $\tilde{\pi}$ for $(a_i, ..., a_n)\vartheta_{i-1}$. Therefore, because of (a) and by the inductive hypothesis:

$$(a_i, ..., a_n)\vartheta_{i-1} \xrightarrow[\tilde{\pi}]{f} {}^{*}_{P} (\tilde{b} \mid \tilde{B} :: \langle a_{i+1}, ..., a_n \rangle)\vartheta_{i-1}\tilde{\sigma}$$

where $\tilde{\sigma} = mgu(a_i \vartheta_{i-1}, \tilde{h})$. Because of (b)

$$(\tilde{b} \mid \tilde{B} :: \langle a_{i+1}, ..., a_n \rangle)\vartheta_{i-1}\tilde{\sigma} \xrightarrow[l]{f} {}_{P} (Q :: \tilde{B} :: \langle a_{i+1}, ..., a_n \rangle)\vartheta_{i-1}\tilde{\sigma} mgu(\tilde{b}\vartheta_{i-1}\tilde{\sigma}, w).$$

Since $\sigma = mgu(\tilde{b}, w)$ and $\tilde{\sigma} = mgu(a_i \vartheta_{i-1}, \tilde{h})$, we prove that:

$$(Q :: \tilde{B} :: \langle a_{i+1}, ..., a_n \rangle)\vartheta_{i-1} mgu(a_i \vartheta_{i-1}, \tilde{h}) mgu(\tilde{b}\vartheta_{i-1} mgu(a_i \vartheta_{i-1}, \tilde{h}), w) \ \sim$$

$$(Q :: \tilde{B} :: \langle a_{i+1}, ..., a_n \rangle) mgu(\tilde{b}, w)\vartheta_{i-1} mgu(a_i \vartheta_{i-1}, \tilde{h} mgu(\tilde{b}, w)).$$

By Lemma A.1 and because $dom(\vartheta_{i-1}) \cap vars(\tilde{h}) = \emptyset$, $dom(\vartheta_{i-1}) \cap vars(w) = \emptyset$ and w is renamed apart from a_i and $\tilde{h}$:

$$\begin{aligned}
\vartheta_{i-1} mgu(a_i\vartheta_{i-1}, \tilde{h}) mgu(\tilde{b}\vartheta_{i-1} mgu(a_i\vartheta_{i-1}, \tilde{h}), w) &= \\
mgu(\mathcal{E}(\vartheta_{i-1}) \cup \{a_i = \tilde{h}\}) mgu(\tilde{b} mgu(\mathcal{E}(\vartheta_{i-1}) \cup \{a_i = \tilde{h}\}), w) &= \\
mgu(\mathcal{E}(\vartheta_{i-1}) \cup \{a_i = \tilde{h}\} \cup \{\tilde{b} = w\})
\end{aligned}$$

and

$$\begin{aligned}
mgu(\tilde{b}, w)\vartheta_{i-1} mgu(a_i\vartheta_{i-1}, \tilde{h} mgu(\tilde{b}, w)) &= \\
mgu(\tilde{b}, w) mgu((\mathcal{E}(\vartheta_{i-1}) \cup \{a_i = \tilde{h}\}) mgu(\tilde{b}, w)) &= \\
mgu(\mathcal{E}(\vartheta_{i-1}) \cup \{a_i = \tilde{h}\} \cup \{\tilde{b} = w\}).
\end{aligned}$$

Therefore, for $1 \leq i \leq k$: $(a_i, ..., a_n)\vartheta_{i-1} \xrightarrow[\pi_i]{f}{}^*_P (B_i :: \langle a_{i+1}, ..., a_n\rangle)\vartheta_i$. Because $B_i = true$ for $i < k$ then: $G \xrightarrow[\pi]{f}{}^*_P (B_k :: \langle a_{i+1}, ..., a_n\rangle)\vartheta'$ where $G\vartheta_k \sim G\vartheta'$ and the corresponding resolvents are $\sim$-equivalent. This concludes the proof.

∎

Proposition **4.13.**
T^f_P is continuous.

Proof. It follows by the oracle additivity. ∎

To prove equivalence of top-down and bottom-up oracle semantics for successful computations we need the following lemmata.

Lemma **4.14.** (AND-compositionality)
Let P be a program, $G = b_1, ..., b_n$ be a goal and f be an oracle for P. $\langle b_1, ..., b_n\rangle \xrightarrow[\pi]{f}{}^*_P true$ is a refutation for G with substitution ϑ iff

1. *for each $1 \leq i \leq n$: ϑ_i and π_i are the substitution and path associated with the refutation $(b_i)\vartheta_1 ... \vartheta_{i-1} \xrightarrow[\pi_i]{f}{}^*_P true$ for the goal $(b_i)\vartheta_1 ... \vartheta_{i-1}$;*
2. *$\pi = \pi_1 :: \cdots :: \pi_n$;*
3. *$f(\{(b_i \leftarrow true)\vartheta_1 ... \vartheta_i; \pi_i)\}, (b_i, ..., b_n)\vartheta_1 ... \vartheta_{i-1}) \neq \emptyset$; and*
4. *$(b_1, ..., b_n)\vartheta \sim (b_1, ..., b_n)\vartheta_1 ... \vartheta_n$.*

Proof.
The proof is straightforward by the oracle downward closure and soundness. ∎

Lemma **4.15.**
Let P be a program, f be an oracle for P, $h \in Atom$ and $a \mid G$ be a goal. If $h \xrightarrow[\pi]{f}{}^*_P B$ with substitution ϑ, $f(\{(h \leftarrow B; \pi)\vartheta\}, a \mid G) \neq \emptyset$ and $mgu(h\vartheta, a) \neq fail$ then $a \mid G \xrightarrow[\pi]{f}{}^*_P (B :: G)\vartheta mgu(h\vartheta, a)$.

Proof. The proof is by induction on the length n of the computation.

__**Base case:**__ is straightforward by definition.

Inductive case: assume

$$h \xrightarrow[\pi]{f}{}^{n}_{P} (b \mid B)\vartheta \xrightarrow[l]{f}{}_{P} (B' :: B)\vartheta mgu(h', b\vartheta)$$

where $(h' \leftarrow B'; l) \in f(P, (b \mid B)\vartheta)$, $f(\{(h \leftarrow B' :: B; \pi :: \langle l \rangle)\vartheta mgu(h', b\vartheta)\}, a \mid G) \neq \emptyset$ and $mgu(h\vartheta mgu(h', b\vartheta), a) \neq fail$.

By the soundness of the unfolding-based semantics, $(h \leftarrow b \mid B; \pi)\vartheta \in lfp(\mathcal{U}^f_P) \subseteq lfp(\mathcal{U}^{slp}_P)$ (i.e. $(h \leftarrow b \mid B; \pi)$ is $\mathcal{U}$-derivable), moreover, by definition:

$$(h \leftarrow B' :: B; \pi :: \langle l \rangle)\vartheta mgu(h', b\vartheta) = Unf^{slp}_{\{(h' \leftarrow B'; l)\}}(\{(h \leftarrow b \mid B; \pi)\vartheta\})$$

and

$$f((h \leftarrow B' :: B; \pi :: \langle l \rangle)\vartheta mgu(h', b\vartheta), a \mid G)) \neq \emptyset$$

Thus, by the oracle completeness:

1. $f(\{(h \leftarrow b \mid B; \pi)\vartheta\}, a \mid G) \neq \emptyset$, from which we obtain by the inductive hypothesis:

$$a \mid G \xrightarrow[\pi]{f}{}^{n}_{P} (b \mid B :: G)\vartheta mgu(h\vartheta, a)$$

2. $f(\{(h' \leftarrow B'; l)\}, (b \mid B :: G)\vartheta mgu(h\vartheta, a)) \neq \emptyset$

Therefore:

$$a \mid G \xrightarrow[\pi::\langle l \rangle]{f}{}^{n+1}_{P} (B' :: B :: G)\vartheta mgu(h\vartheta, a) mgu(h', b\vartheta mgu(h\vartheta, a))$$

Because all the clauses are considered to be renamed apart form each other, by Lemma A.1 we have:

$$\begin{array}{rcl} \vartheta mgu(h\vartheta, a) mgu(h', b\vartheta mgu(h\vartheta, a)) & = & mgu(\{h' = b\} \cup \mathcal{E}(mgu(\{h = a\} \cup \mathcal{E}(\vartheta)))) \\ & = & mgu(\{h = a, h' = b\} \cup \mathcal{E}(\vartheta)) \end{array}$$

and

$$\begin{array}{rcl} \vartheta mgu(h', b\vartheta) mgu(a, h\vartheta mgu(h', b\vartheta)) & = & mgu(\{h = a\} \cup \mathcal{E}(mgu(\{h' = b\} \cup \mathcal{E}(\vartheta)))) \\ & = & mgu(\{h = a, h' = b\} \cup \mathcal{E}(\vartheta)) \end{array}$$

thus both leading to the same resolvent $(B' :: B :: G) mgu(\{h = a, h' = b\} \cup \mathcal{E}(\vartheta))$ (up to variable renaming).

∎

THEOREM **4.16.**
Let P be a program and f an oracle for P. Then,

$$lfp(T^f_P) = \left\{ (h\vartheta \leftarrow true; \pi) \,\middle|\, \begin{array}{l} h \xrightarrow[\pi]{f}{}^{*}_{P} true \\ \text{with substitution } \vartheta \end{array} \right\}.$$

PROOF.
$\supseteq$. The proof is by induction on the length m of the refutation:

Base case: let a be an atom. Assume a has a refutation of length 1 with substitution ϑ, i.e. there exists $(h \leftarrow true; l) \in f(P,a)$ and $\vartheta = mgu(a,h)$.
By the downward closure and additivity,

$$(h \leftarrow true; l)\vartheta \in f(P\vartheta, a) \quad \Rightarrow \quad (h \leftarrow true; l)\vartheta \in f(lwr(P), a).$$

The thesis follows by T_P^f definition; i.e. $(h\vartheta \leftarrow true; l) \in T_P^f(\emptyset)$.

Inductive case: assume the hypothesis for any atomic goal h and $k \leq m$, i.e.

$$h \xrightarrow[\pi]{f} {}_P^k\, true \text{ with substitution } \vartheta \quad \Rightarrow \quad [(h \leftarrow true; \pi)\vartheta]_\sim \in lfp(T_P^f).$$

Let

$$a \xrightarrow[l]{f}{}_P (b_1, ..., b_n)\vartheta_0 \xrightarrow[\pi']{f}{}_P^m\, true.$$

be a refutation of length $m+1$ for the atomic goal a with computed answer substitution $\vartheta_0\vartheta$ and let π be the corresponding sequence of program clauses applied in the computation, i.e. $\pi = \langle l \rangle :: \pi'$, where $(h \leftarrow b_1, ..., b_n; l)$ is the first clause applied in the refutation and $\vartheta_0 = mgu(a,h)$.

By definition: $(h \leftarrow b_1, ..., b_n; l) \in f(P,a)$. Moreover, by Lemma 4.14, for each $i = 1, ..., n$:

$$b_i\vartheta_0\vartheta_1..\vartheta_{i-1} \xrightarrow[\pi_i]{f}{}_P^{m_i}\, true$$

with computed substitution ϑ_i, where $m_i \leq n$, $\pi' = \pi_1 :: \cdots :: \pi_n$, $(b_1, ..., b_n)\vartheta_0\vartheta \sim (b_1, ..., b_n)\vartheta_0\vartheta_1..\vartheta_n$ and $f(\{(b_i \leftarrow true; \pi_i)\vartheta_0..\vartheta_i\}, (b_i, ..., b_n)\vartheta_0..\vartheta_{i-1}) \neq \emptyset$.

By the inductive hypothesis, for $i = 1, ..., n$:

$$[(b_i \leftarrow true; \pi_i)\vartheta_0..\vartheta_i]_\sim \in lfp(T_P^f).$$

Thus, by additivity, for $i = 1, ..., n$:

$$[(b_i \leftarrow true; \pi_i)\vartheta_0..\vartheta_i]_\sim \in f(lfp(T_P^f), (b_i, ..., b_n)\vartheta_0..\vartheta_{i-1}).$$

Since $(h \leftarrow b_1, ..., b_n; l)$ is the first clause applied to the refutation of a and $a\vartheta_0 = h\vartheta_0$, by downward closure we have: $(h \leftarrow b_1, ..., b_n; l)\vartheta_0 \in f(lwr(P), a\vartheta_0)$ which implies that $(h \leftarrow b_1, ..., b_n; l)\vartheta_0 \in f(lwr(P), h\vartheta_0)$. The thesis follows by T_P^f definition; i.e. $[(h \leftarrow true)\vartheta_0\vartheta''; \pi]_\sim \in lfp(T_P^f)$ where, for $i = 1, ..., n$ by denoting ρ_i suitable renaming substitutions:

$$\vartheta'' = mgu(\langle b_1, ..., b_n \rangle\vartheta_0, \langle b_1\vartheta_0\vartheta_1\rho_1, ...b_n\vartheta_0..\vartheta_n\rho_n \rangle)$$

we have $\langle b_1, ..., b_n \rangle\vartheta_0\vartheta'' \sim \langle b_1, ..., b_n \rangle\vartheta_0..\vartheta_n$, which completes the proof.

$\subseteq$. The proof is by induction on the power n of T_P^f.

Base case: is straightforward by definition.

Inductive case: let $(h \leftarrow b_1, ..., b_m; l) \in f(lwr(P), h)$ and for $1 \leq i \leq m$: $(h_i \leftarrow true; \pi_i) \in f(T_P^f\!\uparrow\! n, (b_i, ..., b_m)\vartheta_{i-1})$ where $\vartheta_0 = \varepsilon$ and $\vartheta_i = \vartheta_{i-1}mgu(b_i\vartheta_{i-1}, h_i)$;

(i.e. $(h \leftarrow true; \langle l \rangle :: \pi_1 :: \cdots :: \pi_m)\vartheta_m \in T_P^f\!\uparrow\! n+1$).

By the inductive hypothesis, $\tilde{h}_i \xrightarrow[\pi_i]{f}{}^*_P true$ with substitution $\tilde{\vartheta}_i$ such that $h_i = \tilde{h}_i\tilde{\vartheta}_i$.
Since $f(\{(\tilde{h}_i \leftarrow true; \pi_i)\tilde{\vartheta}_i\}, (b_i, ..., b_m)\vartheta_{i-1}) \neq \emptyset$ and $mgu(b_i\vartheta_{i-1}, \tilde{h}_i\tilde{\vartheta}_i) \neq fail$, by Lemma 4.15 we have for $1 \leq i \leq m$:

$$(b_i, ..., b_m)\vartheta_{i-1} \xrightarrow[\pi_i]{f}{}^*_P (b_{i+1}, ..., b_m)\vartheta_{i-1}\tilde{\vartheta}_i mgu(b_i\vartheta_{i-1}, \tilde{h}_i\tilde{\vartheta}_i).$$

Note that: $(b_{i+1}, ..., b_m)\vartheta_{i-1}\tilde{\vartheta}_i mgu(b_i\vartheta_{i-1}, \tilde{h}_i\tilde{\vartheta}_i) \sim (b_{i+1}, ..., b_m)\vartheta_{i-1} mgu(b_i\vartheta_{i-1}, h_i)$.
Moreover, because $(h \leftarrow b_1, ..., b_m; l) \in f(lwr(P), h)$, there exists $(\tilde{h} \leftarrow \tilde{b}_1, ..., \tilde{b}_m; l) \in f(P, h)$ where $\sigma_0 = mgu(h, \tilde{h})$ and $(h \leftarrow b_1, ..., b_m; l) \sim (\tilde{h} \leftarrow \tilde{b}_1, ..., \tilde{b}_m; l)\sigma_0$. Therefore:

$$h \xrightarrow[l]{f}{}_P (\tilde{b}_1, ..., \tilde{b}_m)\sigma_0$$

Thus, by definition:

$$h \xrightarrow[\langle l\rangle :: \pi_1 :: \cdots :: \pi_m]{f}{}^*_P true$$

with substitution ϑ_m, which concludes the proof. ∎

PROPOSITION **5.1.**
prolog is a downward closed oracle.

PROOF. The proof is by contradiction. Let I be an interpretation, P be a program, $G = a_1, ..., a_n$ and B be goals and $[(c; \pi)]_\sim \in prolog(I, G :: B)$. By definition $mgp^{slp}_\pi(P, G :: B)$ is finite. Assume $mgp^{slp}_\pi(P, G\vartheta)$ be an infinite set, for a substitution ϑ. Let G' be the most general slp-partial answer for $G\vartheta$ in $lfp(\mathcal{U}^{slp}_P)$ with path $\pi' \preceq \pi$. Thus, by definition there exists a most general slp-partial answer G'' for G in $lfp(\mathcal{U}^{slp}_P)$ with path π'. $(G :: B)mgu(G, G'')$ is also a most general partial answer for $G :: B$ in $lfp(\mathcal{U}^{slp}_P)$ with the same path π'. Thus $|mgp^{slp}_\pi(P, G\vartheta)| \leq |mgp^{slp}_\pi(P, G :: B)|$ which is a contradiction. Moreover, because the set mgp^{slp}_π does not depend on the clause c, the applicability of any clause does not depend on the clause instantiation. ∎

To prove soundness and completeness of the *prolog* oracle we need the following

LEMMA **A.2.**
Let P be a program, G be a goal and π, π' be paths. Then

$$mgp^{slp}_{\pi::\pi'}(P, G) = mgp^{slp}_\pi(P, G) \;\cup\; \left\{ (G', \pi') \;\middle|\; \begin{array}{l} (G', \pi') \in mgp^{slp}(P, G) \\ \pi \preceq \pi' \prec \pi :: \pi' \end{array} \right\}$$

PROOF. The proof is straightforward by lexicographic ordering and by definition. ∎

PROPOSITION **5.2.**
prolog is sound and complete.

PROOF. Let $a \mid G$ be a goal and P be a program ordered textually by ℓ.
<u>Soundness</u>
The proof is by induction on the length n of the computation.

Base case: is straightforward by definition;

Inductive case: assume the inductive hypothesis for $n \geq 1$ and let us consider:

$$a \mid G \xrightarrow[\pi]{prolog}{}^{n}_{P} (b \mid B :: G)\vartheta \xrightarrow[l]{prolog}{}_{P} (Q :: B :: G)\vartheta\sigma$$

where $(k \leftarrow Q; l) \in prolog(P, (b \mid B :: G)\vartheta)$ and $\sigma = mgu(k, b\vartheta)$. By the inductive hypothesis and by definition: $|mgp^{slp}_{\pi}(P, a \mid G)| < \omega$ and $|mgp^{slp}_{l}(P, (b \mid B :: G)\vartheta)| < \omega$.

Assume (as a contradiction) that $mgp^{slp}_{\pi::\langle l\rangle}(P, a \mid G)$ is infinite. By Lemma A.2 we have:

$$mgp^{slp}_{\pi::\langle l\rangle}(P, a \mid G) = mgp^{slp}_{\pi}(P, a \mid G) \ \cup \ \left\{ (G', \pi') \;\middle|\; \begin{array}{l} (G', \pi') \in mgp^{slp}(P, a \mid G) \\ \pi \preceq \pi' \prec \pi :: \langle l\rangle \end{array} \right\}.$$

Thus the assumption is true iff the set:

$$D = \left\{ (G', \pi') \;\middle|\; \begin{array}{l} (G', \pi') \in mgp^{slp}(P, a \mid G) \\ \pi \preceq \pi' \prec \pi :: \langle l\rangle \end{array} \right\}$$

is infinite. This is a contradiction because $|mgp^{slp}_{l}(P, (b \mid B :: G)\vartheta)| < \omega$ and since slp is an oracle, by the soundness and completeness of the slp-semantics:

$$D = \left\{ (G', \pi') \;\middle|\; \begin{array}{l} (G', \pi) \in mgp^{slp}(P, a \mid G) \\ (G'', \pi'') \in mgp^{slp}(P, (b \mid B :: G)\vartheta) \\ \pi' = \pi :: \pi'' \ \text{ and } \ \pi'' \prec \langle l\rangle \end{array} \right\}$$

is finite.

Completeness

Let $n < \omega$ and $C \ll_{a|G} P$, $(h \leftarrow b \mid B; \pi) \ll_{C,a|G} \mathcal{U}^{prolog}_{P}\uparrow n$.

Assume $prolog(Unf^{prolog}_{\{C\}}(\{(h \leftarrow b \mid B; \pi)\}), a \mid G) \neq \emptyset$ and $mgu(h, a) \neq fail$.

1. By definition $|mgp^{slp}_{\pi::\langle \ell(C)\rangle}(P, a \mid G)| < \omega$, and, since by Lemma A.2

$$mgp^{slp}_{\pi}(P, a \mid G) \subseteq mgp^{slp}_{\pi::\langle \ell(C)\rangle}(P, a \mid G)$$

then $|mgp^{slp}_{\pi}(P, a \mid G)| < \omega$, i.e. $prolog(\{(h \leftarrow b \mid B; \pi)\}, a \mid G) \neq \emptyset$.

2. By Lemma A.2

$$\left| \left\{ (G', \pi') \;\middle|\; \begin{array}{l} (G', \pi') \in mgp^{slp}(P, a \mid G) \\ \pi \preceq \pi' \prec \pi :: \langle \ell(C)\rangle \end{array} \right\} \right| < \omega$$

Let us assume the set $mgp^{slp}_{\langle \ell(C)\rangle}(P, (b|B :: G)mgu(h, a))$ is infinite. Because $prolog$ is reductive, there exists $(\tilde{h} \leftarrow \tilde{b} \mid \tilde{B}; \pi) \in lfp(\mathcal{U}^{slp}_{P})$ such that $(a \mid G)mgu(\tilde{h}, a) \geq (a \mid G)mgu(h, a)$ and $(\tilde{b} \mid \tilde{B} :: G)mgu(\tilde{h}, a) \geq (b \mid B :: G)mgu(h, a)$. Thus, if $(b \mid B :: G)mgu(h, a)\vartheta \in mgp^{slp}_{\langle \ell(C)\rangle}(P, (b \mid B :: G)mgu(h, a))$ with $\vartheta = mgu(b\ mgu(h, a), h')$ and $C = (h' \leftarrow B'; \ell(C)) \in P$, then $(a|G)mgu(\tilde{h}, a)mgu(\tilde{b}\ mgu(\tilde{h}, a), h')$ is the most general partial answer for $a \mid G$ with path $\pi :: \langle \ell(C)\rangle$ in $lfp(\mathcal{U}^{slp}_{P})$. Therefore

$$|mgp^{slp}_{\langle \ell(C)\rangle}(P, (b \mid B :: G)mgu(h, a))| \leq \left| \left\{ (G', \pi') \;\middle|\; \begin{array}{l} (G', \pi') \in mgp^{slp}(P, a \mid G) \\ \pi \preceq \pi' \prec \pi :: \langle \ell(C)\rangle \end{array} \right\} \right|$$

which is a contradiction. Thus $f(\{C\}, (b \mid B :: G)mgu(h, a)) \neq \emptyset$.

■

THEOREM **5.3.**
Let P be a program textually ordered by ℓ, $C \in P$, $Q = \left\{ C' \;\middle|\; \ell(C') < \ell(C) \right\}$, and G be a goal. Then, $G \xrightarrow{slp}_Q G' \;\Rightarrow\; T(\Psi_P^{slp}, G')$ is finite iff $mgp^{slp}_{\langle \ell(C) \rangle}(P, G)$ is finite.

PROOF. Assume the hypothesis: $G \xrightarrow{slp}_Q G' \;\Rightarrow\; T(\Psi_P^{slp}, G')$ is finite. By Theorems 4.7 and 4.11 and Definition 3.4 this is equivalent to

$$G \xrightarrow{slp}_Q G' \;\Rightarrow\; mgp^{slp}(P, G') \text{ is finite.}$$

The set of most general *slp*-partial answers for G starting with any of the clauses in Q is then finite; namely the set

$$\left\{ (G''; \langle \ell(C') \rangle :: \pi) \;\middle|\; \begin{array}{l} \ell(C') < \ell(C) \\ (G''; \langle \ell(C') \rangle :: \pi) \in mgp^{slp}(P, G) \end{array} \right\}$$

is finite. By definition, the assumption is true iff $mgp^{slp}_{\langle \ell(C) \rangle}(P, G)$ is finite. ■

References

[Apt 90] K. R. Apt. Introduction to logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 495–574. Elsevier, Amsterdam and The MIT Press, Cambridge, 1990.

[Apt, Bol and Klop 89] K. R. Apt, R. N. Bol, and J. W. Klop. On the safe termination of Prolog programs. In G. Levi and M. Martelli, editors, *Proceedings of the Sixth International Conference on Logic Programming*, pages 353–368, Lisbon, 1989. The MIT Press.

[Apt and van Emden 82] K. R. Apt and M. H. van Emden. Contributions to the theory of logic programming. *Journal of the ACM*, 29(3):841–862, July 1982.

[Apt and Pedreschi 93] K. R. Apt and D. Pedreschi. Reasoning about termination of pure Prolog programs. *Information and Computation*, 106(1):109–157, September 1993.

[Arbab and Berry 87] B. Arbab and D. M. Berry. Operational and denotational semantics of Prolog. *The Journal of Logic Programming*, 4(4):309–330, December 1987.

[Barbuti, Codish, Giacobazzi and Levi 92] R. Barbuti, M. Codish, R. Giacobazzi, and G. Levi. Modeling Prolog control. In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Programming Languages*, pages 95–104, January 1992.

[Barbuti, Codish, Giacobazzi and Levi 93] R. Barbuti, M. Codish, R. Giacobazzi, and G. Levi. Modeling Prolog control. *Journal of Logic and Computation*, 3(6):579–603, 1993.

[Barbuti, Codish, Giacobazzi and Maher 92] R. Barbuti, M. Codish, R. Giacobazzi, and M. J. Maher. Oracle semantics for Prolog. In [Levi and Kirchner 92], pages 100–114.

[Barbuti and Giacobazzi 92] R. Barbuti and R. Giacobazzi. A bottom-up polymorphic type inference in logic programming. *Science of Computer Programming*, 19(3):281–313, 1992.

[Barbuti, Giacobazzi and Levi 93] R. Barbuti, R. Giacobazzi, and G. Levi. A general framework for semantics-based bottom-up abstract interpretation of logic programs. *ACM Transactions on Programming Languages and Systems*, 15(1):133–181, 1993.

[Börger 90] E. Börger. A logical operational semantics of full Prolog. In E.Börger, H. Kleine, H. Büning, and M. Richter, editors, *CSL 89. 3rd workshop on Computer Science Logic*, volume 440 of *Lecture Notes in Computer Science*, pages 36–64. Springer-Verlag, Berlin, 1990.

[Bossi, Gabbrielli, Levi and Meo 94] A. Bossi, M. Gabbrielli, G. Levi, and M. Meo. A compositional semantics for logic programs. *Theoretical Computer Science*, 122(1–2):3–47, 1994.

[de Bruin and de Vink 89] 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*, pages 178–192. Springer-Verlag, Berlin, 1989.

[Bruynooghe 91] M. Bruynooghe. A practical framework for the abstract interpretation of logic programs. *The Journal of Logic Programming*, 10(1,2,3 and 4):91–124, 1991.

[Le Charlier and Van Hentenryck 94] B. Le Charlier and P. Van Hentenryck. Experimental evaluation of a generic abstract interpretation algorithm for Prolog. *ACM Transactions on Programming Languages and Systems*, 16(1):35–101, January 1994.

[Codish, Dams and Yardeni 94] M. Codish, D. Dams, and E. Yardeni. Bottom-up abstract interpretation of logic programs. *Theoretical Computer Science*, 124(1):93–126, 1994.

[Debray and Lin 91] S. K. Debray and N.-W. Lin. Automatic complexity analysis of logic programs. In [Furukawa 91], pages 599–613.

[Debray and Mishra 87] S. K. Debray and P. Mishra. Denotational and operational semantics for Prolog. In M. Wirsing, editor, *Formal Description of Programming Concepts III*, pages 245–269. North-Holland, Amsterdam, 1987.

[Debray and Warren 89] S. K. Debray and D. S. Warren. Functional computations in logic programs. *ACM Transactions on Programming Languages and Systems*, 11(3):451–481, July 1989.

[De Schreye and Decorte 94] D. De Schreye and S. Decorte. Termination of Logic Programs: the Never-Ending Story, *Journal of Logic Programming*, 19 & 20:199–260, 1994.

[Eder 85] E. Eder. Properties of substitutions and unifications. *Journal of Symbolic Computation*, 1:31–46, 1985.

[van Emden and Kowalski 76] M. H. van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.

[Falaschi, Levi, Martelli and Palamidessi 89] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. Declarative modeling of the operational behavior of logic languages. *Theoretical Computer Science*, 69(3):289–318, 1989.

[Falaschi, Levi, Martelli and Palamidessi 93] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. A model-theoretic reconstruction of the operational semantics of logic programs. *Information and Computation*, 102(1):86–113, 1993.

[Furukawa 91] K. Furukawa, editor. *Proceedings of the Eighth International Conference on Logic Programming*, Paris, France, 1991. The MIT Press.

[Gabbrielli and Meo 92] M. Gabbrielli and M. C. Meo. Fixpoint semantics for partial computed answer substitutions and call patterns. In Levi and Kirchner [Levi and Kirchner 92], pages 84–99.

[Gaifman, Maher and Shapiro 91] H. Gaifman, M. J. Maher and E. Shapiro. Replay, Recovery, Replication and Snapshots of Nondeterministic Concurrent Programs, *Proceedings of the ACM Symposium on Principles of Distributed Computation,* 1991.

[Griswold 93] R. E. Griswold. The evaluation of expressions in Icon, *ACM Transactions on Programming Languages and Systems*, 4(4):563–584, 1993.

[Griswold, Poage and Polonsky 71] R. E. Griswold, J. F. Poage and I. P. Polonsky. *The SNOBOL4 Programming Language*, Prentice-Hall, 1971.

[Haridi and Janson 90] S. Haridi and S. Janson. Kernel Andorra Prolog and its computation model. In David H. D. Warren and Peter Szeredi, editors, *Proceedings of the Seventh International Conference on Logic Programming*, pages 31–46, Jerusalem, 1990. The MIT Press.

[Jaffar and Lassez 87] J. Jaffar and J-L. Lassez. Constraint Logic Programming, *Proc. 14th ACM Symposium on Principles of Programming Languages*, Munich (January 1987), 111–119.

[Jaffar and Maher 94] J. Jaffar and M. J. Maher. Constraint Logic Programming: A Survey, *Journal of Logic Programming*, 19 & 20:503–581, 1994.

[Jayaraman 89] B. Jayaraman. Eql: The Language and Its Implementation, *IEEE Transactions on Software Engineering* 15(6):771–779, 1989.

[Jayaraman 92] B. Jayaraman. Implementation of Subset-Equational Programs, *Journal of Logic Programming* 12(4):299–324, 1992.

[Jones and Mycroft 84] N. D. Jones and A. Mycroft. Stepwise development of operational and denotational semantics for Prolog. In *Proceedings of the International Symposium on Logic Programming*, pages 281–288, Atlantic City, 1984. IEEE, Computer Society Press.

[Kanomori and Horiuchi 88] T. Kanomori and K. Horiuchi. Polymorphic type inference in prolog by abstract interpretation. In *Logic Programming 87- Tokyo*, volume 315 of *Lecture Notes in Computer Science*, pages 195–214. Springer-Verlag, Berlin, 1988.

[Lassez, Maher, and Marriott 88] J.-L. Lassez, M. J. Maher, and K. Marriott. Unification revisited. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 587–625. Morgan Kaufmann, Los Altos, Ca., 1988.

[Lever 91] J. M. Lever. Proving program properties by means of SLS-resolution. In Furukawa [Furukawa 91], pages 614–628.

[Levi 88] G. Levi. Models, unfolding rules and fixpoint semantics. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, pages 1649–1665, Seattle, 1988. The MIT Press.

[Levi and Kirchner 92] G. Levi and H. Kirchner, editors. *Algebraic and Logic Programming, Third International Conference*, LNCS 632, Volterra, Italy, September 2–4, 1992. Springer-Verlag.

[Lloyd 87] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 2^{nd} edition, 1987.

[Maher and Ramakrishnan 89] M. J. Maher and R. Ramakrishnan. Déjà Vu in Fixpoints of Logic Programs, *Proc. North American Conference on Logic Programming,* Cleveland, October, 1989, pages 963–980.

[Naish 86] L. Naish. *Negation and Control in Prolog*, volume 238 of *Lecture Notes in Computer Science.* Springer-Verlag, Berlin, 1986.

[Palamidessi 90] C. Palamidessi. Algebraic properties of idempotent substitutions. In M. S. Paterson, editor, *Proceedings of the 17th International Colloquium on Automata, Languages and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 386–399. Springer-Verlag, Berlin, 1990.

[Przymusinski 89] T. Przymusinski. On the declarative and procedural semantics of logic programs. *Journal of Automated Reasoning*, 5(2):201–228, 1989.

[Pyo and Reddy 89] C. Pyo and U. S. Reddy. Inference of Polymorphic Types for Logic Programs. In Ewing L. Lusk and Ross A. Overbeek, editors, *Proceedings of the North American Conference on Logic Programming*, pages 1115–1134, Cleveland, Ohio, USA, 1989.

[Saraswat 90] V. A. Saraswat. Concurrent constraint programming. In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Programming Languages*, pages 232–245, January 1990.

[Saraswat, Weinbaum, Kahn and Shapiro 88] V. A. Saraswat, D. Weinbaum, K. Kahn and E. Shapiro. Detecting stable properties of networks in concurrent logic programming languages. *Proceedings of the ACM Symposium PODC*, 1988.

[Smith, Genesereth and Ginsberg 86] D. E. Smith, M. R. Genesereth, and M. L. Ginsberg. Controlling recursive inference. *Artificial Intellingence*, 30:343–389, 1986.

[Sterling and Shapiro 86] L. Sterling and E. Shapiro. *The Art of Prolog.* The MIT Press, Cambridge, Massachusetts, 1986.

[Vasak and Potter 86] T. Vasak and J. Potter. Characterisation of terminating logic programs. In *Symposium on Logic Programming*, pages 140–147. IEEE Computer Society, The Computer Society Press, September 1986.

[Verschaetse and De Schreye 91] K. Verschaetse and D. De Schreye. Deriving termination proofs for logic programs, using abstract procedures. In [Furukawa 91], pages 301–315.