A semantics for logic programs based on first order hereditary Harrop formulas

Ernesto Lastres, René Moreno

Abstract

The paper introduces a semantics for logic programs based on first order hereditary Harrop formulas which are expressed in terms of intuitionistic derivations. The derivations are constructed by means of an intuitionistic proof procedure that constitutes the resolution mechanism of the language. The semantics of a program is a goal independent denotation which can be equivalently specified by a denotational and an operational semantics. The denotational semantics is defined using a set of primitive semantic operators that act on derivations and are directly related to the properties of the derivations.

Keywords: Logic Programming, Abstract Interpretation, Harrop Formulas

1 Introduction

The aim of this work is to introduce a semantics for logic programs based on first order hereditary Harrop (*fohh*) formulas expressed in terms of intuitionistic proofs, in order to study the various properties of such programs. The proof procedure presented by Nadathur in [10] constitutes the basis of the proof mechanism of languages like λProlog and specifies an operational model of such kind of languages. There are various attempts of defining semantics for *fohh*-programs. In [9] is presented a semantics based on a generalization of the standard immediate consequences T_P ground operator [7], to describe a Kripke-like model theory of positive Horn clauses programs permitting implications. There is also a T_P-like approach using categories [4]. These semantics are based on the notion of success set and therefore are not adequate to reason about important properties of programs as well as about its behavior. One important point in the understanding of the program meaning is the equivalence of programs, which is based on our ability to detect when two programs can not be distinguished observing their behaviors. It is well known the inadequacy of the previously mentioned approaches to study the observational equivalence of classical logic programs [2]. In that sense the s-semantics [3, 5], extending the Herbrand interpretation, really captures the operational semantics, and is therefore suitable for semantic based logic analysis and to reason in terms of program equivalences. Unfortunately this approach can not be used in *fohh*-programs, because the introduction of implication in goals invalidates the use of the computed answers obtained from pure atomic goals for defining program equivalences. In fact, the implication in the *fohh* case is operationally treated by adding a clause to the program, which can dramatically change

Department of Computer Science, University of Pisa, Corso Italia 40, I-56125, Pisa, Italy , Tel.: +39 50 887279, Fax.: +39 50 887229, {lastres,moreno}@di.unipi.it

the meaning of programs that are equivalent in terms of the s-semantics. Hence, to have a goal-independent semantics, it is not enough to reason about the computed answers in the way it is achieved with the s-semantics. We need therefore a semantics capable to represent how the answers are calculated, furthermore it must be compositional w.r.t. all language operators. It means that our semantics must be compositional w.r.t. the union of programs, i.e., the addition of new program modules.

Our work follows the methodology of the semantics presented in [1], but we have to face the problem that the more complex structure of the derivations makes it difficult to find the properties on which all our approach is based, i.e., the properties of compositionality. We propose a more general operational semantics (a top-down denotation) based on derivations, associating to each program all derivations constructed by the proof procedure, from all possible pure atomic goals. More than a notion of derivation, we want also to have a denotational semantics which allows to define goal-independent denotations, and therefore the semantics of a program as a set of procedures. Furthermore, the denotational semantics we present is equivalent to the top-down denotation, and is defined by means of various operators that gives to it its compositional nature. The semantic operators used to define the bottom-up denotation are the counterpart of the syntactic ones. They allow us to describe operational properties of program derivations, considering that they deal with low-level operational details.

The reader is assumed to be familiar with notions of logic programming and semantics of logic programs. The paper is organized as follows: in section 2 we present the basic definitions of the *fohh*; in section 3 we describe the intuitionistic proof procedure; in sections 4 and 5 we introduce the semantics domains and denotational semantics respectively; and finally in sections 6 and 7 we present our operational model and semantics properties of the denotation.

2 First Order Hereditary Harrop Formulas

The first order hereditary Harrop formulas (*fohh*-formulas) [6, 8] are divided in two groups: the G-formulas (goals) and the D-formulas (definite clauses). They are defined by the following syntax rules, where A is an atom:

$$\begin{array}{ll} G & ::= A \mid G \wedge G \mid G \vee G \mid \exists x.G \mid D \supset G \mid \forall x.G \\ D & ::= A \mid G \supset A \mid D \wedge D \mid \forall x.D \end{array}$$

We assume that the formulas are defined over a first order signature where $Vars$ denotes the set of variables and $Consts$ the set of constants. By $vars$ and $consts$ we denote the functions that, respectively, give the set of variables and constants occurring in a formula or in a set of them.

These formulas define a logic programming language in the sense that a G -formula can be seen as a query or a goal to be resolved using a program which is a set of definite clauses (D-formulas). In this case the process of answering consists of constructing an intuitionistic proof of the existential closure of the query from the given program. In the following we will refer to a D-formula as a program clause and to a G-formula as a goal. We denote by $\mathbb{P}$ the set of all $fohh$-programs.

To show the results described in [10] we need first to define some notions concerning the logic programming language we are dealing with.

Definition 2.1. The *elaboration* of a program clause D, denoted by $elab(D)$, is a set defined as:

- If $D ::= A$ or $D ::= G \supset A$, then $elab(D) = \{D\}$.
- If $D ::= D_1 \wedge D_2$ then $elab(D) = elab(D_1) \cup elab(D_2)$.
- If $D ::= \forall x.D'$, then $elab(D) = \{\forall x.D'' \mid D'' \in elab(D')\}$

It is clear that in $elab(D)$ the conjunctions are eliminated, i.e., all the clauses are atoms or implication formulas universally quantified.

Definition 2.2. Let D be a clause. An instance of a formula $\forall x_1 ... \forall x_n.F \in elab(D)$ is any formula that can be written as $F\theta$, where θ is a substitution whose domain is $\{x_1, ..., x_n\}$. The instances of a D-formula are all the instances of the formulas in $elab(D)$.

3 Intuitionistic Proof Procedure

In this section we describe the proof procedure presented by Nadathur in [10] for determining a proof (if it exists) of a goal formula from a *fohh* -program. First are introduced some notions on unification mechanisms on which are based the proof procedure.

3.1 Labeled Unification

The non-deterministic proof procedure described in [10] makes use of a different notion of unification that restricts the standard unification, creating a sort of scoping between variables and constants.

Definition 3.1. The *labelling function* $L : Vars \cup Consts \to \mathbb{N}$ associates to each constant or variable a natural number. We call this number the *level* (*label*) of the variables and constants. The function $\mathcal{L}$, defined over formulas, gives the labelling function on the variables and constants in such formulas.

Definition 3.2. Let $\theta = \{[t_i/x_i] \mid 1 \leq i \leq n\}$ be a substitution and let L be a labelling function on the variables and constants in θ. The substitution θ is said to be a *proper labelled substitution* with respect to L if and only if, $L(c) \leq L(x_i)$ for every constant c appearing in t_i. The *induced labelling* is the labelling L' of the substitution θ which is obtained from L in the following manner:

$$L'(x) = min\,(\{L(x)\} \cup \{L(x_i) \mid [t_i/x_i] \in \theta \text{ and } x \text{ appears in } t_i\})$$

Definition 3.3. Let $T = \{\langle t_i, s_i\rangle \mid 1 \leq i \leq n\}$ be a set of pairs of terms and let L be a labelling function on the variables and constants occurring in T. A *labelled unifier* (*l-unifier*) for T under L is a substitution θ which is proper w.r.t. L and for each pair $\langle t_i, s_i\rangle \in T$ the equality $t_i\theta = s_i\theta$ holds. A *most general l-unifier* ($lmgu$) for T under L is a l-unifier θ such that for any other l-unifier λ for T there is a substitution σ with $\theta\sigma = \lambda$.

3.2 The proof procedure

This algorithm, as presented in [10], determines the transition of one computation state to other. The initial state represents the information about the goal and the program. The final state contains the empty goal (if there is a proof). The information about all substitutions needed to calculate the computed answer can be retrieved from all preceding states.

Definition 3.4. A state S is a tuple $\langle W, k, L, \theta \rangle$ where

- W is a set of tuples $\langle G, P, n \rangle$ where G is a goal, P is a program and n is a natural number,
- k is a clause used to reduce a goal (κ denotes the void clause, i.e., no reduction),
- L is a labelling function on $vars(W) \cup consts(W)$, and
- θ is a substitution (ϵ denotes the empty substitution).

The *initial state* of a computation G in P is defined as $S_0 = \langle \{\langle G, P, 0 \rangle\}, \kappa, L_0, \epsilon \rangle$, where $L_0 = \mathbf{0}$. A state is said to be *proper* if for each $w = \langle G, P, n \rangle \in W$ is the case that $L_{|w} \leq n$.

Definition 3.5 (proof procedure). The state $S_1 = \langle W_1, k_1, L_1, \theta_1 \rangle$ *derives in one step* to the state $S_2 = \langle W_2, k_2, L_2, \theta_2 \rangle$ using the following rules:

1. If $\exists w_1 = \langle G_1 \wedge G_2, P, n \rangle \in W_1$ then $W_2 = (W_1 - \{w_1\}) \cup \{\langle G_1, P, n \rangle, \langle G_2, P, n \rangle\}$, $L_2 = L_1$, $k_2 = \kappa$ and $\theta_2 = \epsilon$.
2. If $\exists w_1 = \langle G_1 \vee G_2, P, n \rangle \in W_1$ then $W_2 = (W_1 - \{w_1\}) \cup \{\langle G_i, P, n \rangle\}$ for $i \in \{1, 2\}$, $L_2 = L_1$, $k_2 = \kappa$ and $\theta_2 = \epsilon$.
3. If $\exists w_1 = \langle D \supset G, P, n \rangle \in W_1$ then $W_2 = (W_1 - \{w_1\}) \cup \{\langle G, P \cup \{D\}, n \rangle\}$, $L_2 = L_1$, $k_2 = \kappa$ and $\theta_2 = \epsilon$.
4. If $\exists w_1 = \langle \exists x.G, P, n \rangle \in W_1$ then $W_2 = (W_1 - \{w_1\}) \cup \{\langle G[y/x] \rangle, P, n\}$ for $y \notin vars(W_1)$, $L_2 = L_1 \cup \{\langle y, n \rangle\}$, $k_2 = \kappa$ and $\theta_2 = \epsilon$.
5. If $\exists w_1 = \langle \forall x.G, P, n \rangle \in W_1$ then $W_2 = (W_1 - \{w_1\}) \cup \{\langle G[c/x] \rangle, P, n + 1\}$ for $c \notin consts(W_1)$, $L_2 = L_1 \cup \{\langle c, n + 1 \rangle\}$, $k_2 = \kappa$ and $\theta_2 = \epsilon$.
6. If $\exists w_1 = \langle A, P, n \rangle \in W_1$ and $\exists k = \forall x_1 ... \forall x_n . A' \in elab(P)$ such that $\exists \theta = lmgu(A, A'\rho)$ relative to $L' = L_1 \cup \{\langle y, n \rangle \mid y \in range(\rho)\}$, where ρ is a renaming over all variables of A' with $range(\rho) \cap vars(W_1) = \emptyset$, then $W_2 = \{w\theta \mid w \in W_1 - \{w_1\}\}$, $k_2 = k\rho$, L_2 is the labelling induced by θ from L', and $\theta_2 = \theta$.
7. If $\exists w_1 = \langle A, P, n \rangle \in W_1$ and $\exists k = \forall x_1 ... \forall x_n . (G' \supset A') \in elab(P)$ such that $\exists \theta = lmgu(A, A'\rho)$ relative to $L' = L_1 \cup \{\langle y, n \rangle \mid y \in range(\rho)\}$, where ρ is a renaming over all variables of A' with $range(\rho) \cap vars(W_1) = \emptyset$, then $W_2 = \{w\theta \mid w \in W_1 - \{w_1\}\} \cup \{\langle G', P, n \rangle \theta\}$, $k_2 = k\rho$, L_2 is the labelling induced by θ from L', and $\theta_2 = \theta$.

The derivation step between states S_1 and S_2 is denoted by $S_1 \rightarrow S_2$

Definition 3.6. The sequence of states $S_0, ..., S_n$ is a *derivation sequence* ($S_0 \rightarrow^* S_n$) if $S_i \rightarrow S_{i+1}$, $i \geq 0$. The sequence is a *successful derivation* if S_0 is an initial state and $S_n = \langle \emptyset, k_n, L_n, \theta_n \rangle$ for some L_n, k_n, and θ_n. Moreover, we say that a non successful derivation is a *failure* if it can not be extended by applying a derivation step to its last state.

Given a derivation $d = \langle W_0, k_0, L_0, \theta_0 \rangle, ..., \langle W_n, k_n, L_n, \theta_n \rangle$ we denote by $first(d)$ and $last(d)$ the set of all goals occurring in the first and the last derivation state respectively. By $clauses(d)$ we denote the set of all program clauses selected to reduce goals at each state of a derivation d, and by $prefix(d)$ we denote the set of all derivations which are prefixes of d.

Lemma 3.1. ([10]). *Let $S_0, ..., S_n$ be a derivation sequence where S_0 is an initial state. Then for each S_i, $i < n$, the substitution $\tau_i = \theta_n \circ ... \circ \theta_{i+1}$ restricted to $vars(W_i)$ is proper w.r.t. L_i.*

Definition 3.7. Let $S_0 = \langle \{\langle G, P, 0\rangle\}, \kappa, L_0, \epsilon\rangle$ be an initial state and $S_0 \rightarrow^* S_n$ a successful derivation, its associated *answer substitution* is the restriction of $\sigma = \theta_1 ... \theta_n$ to $vars(G, P)$. The *computed answer* is the formula $G\sigma$.

Lemma 3.2. ([10]). *All states of a derivation starting from an initial state are proper.*

By completeness theorem (theorem 15, [10]) we know that a computed answer of a proof (successful derivation) exists regardless the goal selected at each stage in constructing the derivation. So, we can establish an equivalence relation between these derivations.

Definition 3.8. The *computation tree* of a goal $w = \langle G, P, 0\rangle$ has w as root node, and each node (goal) has as children all the goals that it generates when is reduced. The *computation set* of a derivation d is composed by all trees generated by the goals in $first(d)$.

Definition 3.9. We say that two derivations $d_1 = S_1 \rightarrow^* S_n$ and $d_2 = S'_1 \rightarrow^* S'_n$ are *equivalent* if they have the same computation set, modulo renaming of variables and constants with label greater than 0. In that case we say that d_1 is a *variant* of d_2 ($d_1 \equiv_c d_2$).

Since there are new constants in the derivations (with label greater than 0), introduced by the non-deterministic proof procedure for resolving the universal quantification in goals, two equivalent derivations can contain different constants. In that case we say that a derivation is a variant of other, if there exists a renaming of constants and variables where each one is replaced by a constant or variable of the same level, that make both derivations syntactically equal.

4 Semantic Domains

Now we use the derivation concept introduced in the previous section to formalize the domains of the semantics definitions.

A set of derivations S is said to be *well formed* if and only if for any $d \in S$ we have $prefix(d) \subseteq S$. We denote by WFD the *complete lattice of well-formed set of derivations* ordered by $\subseteq$.

Definition 4.1. A *collection* is a partial function $C : Goals \rightarrow WFD$, such that if $C(G)$ is defined, then it is a well-formed set of derivations. $\mathbb{C}$ is the domain of all collections ordered by $\sqsubseteq$, where $C \sqsubseteq C'$ iff $\forall G.\, C(G) \subseteq C'(G)$. A *pure collection* is a collection defined only for pure atomic goals.

We introduce the relation of *equivalence modulo variance* $\equiv_{\mathbb{C}}$ defined on collections. Namely, $C \equiv_{\mathbb{C}} C'$ if and only if for any G there exists a variant G' of G such that, if $C(G)$ is defined, then $C'(G')$ is defined, and for any $d \in C(G)$ there exists $d' \in C'(G')$ such that $d \equiv_c d'$.

Definition 4.2. An interpretation I is a pure collection modulo variance. The set of all interpretations is denoted by $\mathbb{I}$. The pair $(\mathbb{I}, \sqsubseteq)$ is a complete lattice with the induced quotient order.

5 Denotational Semantics

We define a denotational semantics inductively on the following syntax of *fohh*-logic programs.

$$\begin{array}{lcl} QUERY & ::= & GOAL \text{ in } PROG \\ GOAL & ::= & G\text{-formula} \\ PROG & ::= & \{D\text{-formula}\} \cup PROG \mid \emptyset \end{array}$$

where the syntax of G-formulas and D-formulas is already defined in the section 2.

The semantic functions are

$$\begin{array}{ll} \mathcal{Q} & : QUERY \to \mathbb{C} \\ \mathcal{G} & : GOAL \to (\mathbb{N} \to \mathbb{P} \to \mathbb{I} \to \mathbb{C}) \\ \mathcal{P} & : PROG \to (\mathbb{I} \to \mathbb{I}) \\ \mathcal{C} & : D\text{-formula} \to (\mathbb{N} \to \mathbb{PI} \to \mathbb{I}) \end{array}$$

and are defined by means of some specific operators on which is based the compositionality. The operator $+$ computes the nondeterministic union of two classes of collections. The operator $\bowtie$ computes the interpretation obtained by replacement, adding to the first derivation step (computed by the operator *tree*) all possible derivations of the clause goal. The operator $\cdot$ restricts (instantiates) an interpretation to such derivations that can be "matched" with an atomic goal. The operator $\otimes$ computes the conjunction of two interpretations, leaving only the unifiable derivations of the two goals, while the operator $\oplus$ computes the extension of the union of two classes of interpretations. Finally we introduce the operator $\uplus$ to calculate an extended interpretation given a new clause.

1. $\mathcal{Q}[\![G \text{ in } P]\!] := \mathcal{G}[\![G]\!]^0_{\mathsf{lfp}\, \mathcal{P}[\![P]\!], P}$
2. $\mathcal{P}[\![P \bigcup \{c\}]\!]_I := \mathcal{C}[\![c]\!]^0_{I, P \bigcup \{c\}} + \mathcal{P}[\![P]\!]_I, \qquad \mathcal{P}[\![\emptyset]\!]_I := \lambda I.I$
3. $\mathcal{C}[\![p(\mathbf{t})]\!]^n_P := tree(p(\mathbf{t}))^n_P$
4. $\mathcal{C}[\![G \supset p(\mathbf{t})]\!]^n_{I,P} := tree(G \supset p(\mathbf{t}))^n_P \bowtie \mathcal{G}[\![G]\!]^n_{I,P}$
5. $\mathcal{G}[\![A]\!]_I := A \cdot I$
6. $\mathcal{G}[\![G_1 \wedge G_2]\!]^n_{I,P} := \mathcal{G}[\![G_1]\!]^n_{I,P} \otimes \mathcal{G}[\![G_2]\!]^n_{I,P}$
7. $\mathcal{G}[\![G_1 \vee G_2]\!]^n_{I,P} := \mathcal{G}[\![G_1]\!]^n_{I,P} \oplus \mathcal{G}[\![G_2]\!]^n_{I,P}$
8. $\mathcal{G}[\![\exists x.G]\!]^n_{I,P} := \exists^n_x \mathcal{G}[\![G]\!]^n_{I,P}$

9. $\mathcal{G}[\![\forall x.G]\!]^n_{I,P} := \mathbb{V}^n_x \mathcal{G}[\![G]\!]^{n+1}_{I,P}$

10. $\mathcal{G}[\![D \supset G]\!]^n_{I,P} := \mathcal{G}[\![G]\!]^n_{I',P'}$ where $I' = I \uplus \mathcal{C}[\![D]\!]^n_{I,P}$ and $P' = P \cup \{D\}$

Before describing the semantic operators, we need to define some concepts and operations on derivations that will help us to define all compositional operators over collections.

Definition 5.1. Let $d_1 = S^1_1, ..., S^1_n$ and $d_2 = S^2_1, ..., S^2_m$ be derivations, the *concatenation* of d_1 and d_2, $(d_1 :: d_2)$ is defined if $S^1_n = S^2_1$ and $vars(d_1) \cap vars(d_2) = vars(S^2_1)$, then $d_1 :: d_2 = S^1_1, ..., S^1_n, S^2_2, ..., S^2_m$.

The restriction introduced on the occurrence of variables asserts that all variables introduced in the second derivation must be new with respect to the first one.

Definition 5.2. Let $d = S_1, ..., S_n$ be a derivation and let W' be a set of goals. The *insertion of* W' *in* d $(d \triangleleft W')$ is defined if each state $S'_i = \langle W_i \cup (W'\sigma_i),\ k_i,\ L_i \cup L'_i,\ \theta_i \rangle$ is a proper state, where $S_i = \langle W_i,\ k_i,\ L_i,\ \theta_i \rangle \in d$ and L'_i is the labelling resulting when applying σ_i to W', with $\sigma_i = \theta_1 ... \theta_i$. In this case $d \triangleleft W' = S'_1, ..., S'_n$.

The insertion of a goal in a derivation means that the goal is added to each state, and it remains unsolved at the end of the derivation, but modified by all the substitutions executed in the preceding states.

Definition 5.3. Let $d_1 = S^1_1, ..., S^1_n$ and $d_2 = S^2_1, ..., S^2_m$ be derivations. The *fusion* of d_1 and d_2 $(d_1 \sharp d_2)$ is defined if $vars(d_1) \cap vars(d_2) \subseteq vars(first(d_1) \cup first(d_2))$ and if the concatenation $d'_1 :: d'_2$ is defined, where $d'_1 = d_1 \triangleleft first(d_2)$, $d'_2 = d_2 \triangleleft last(d_1)$ and $\forall x.\, x \in vars(d_1) \cap vars(d_2)$ is the case that $\sigma_1(x) = \sigma_2(x)$, where σ_1 and σ_2 are the computed substitution of d_1 and d_2 respectively.

Definition 5.4. Let $d = S_1, ..., S_n$ be a derivation and θ an idempotent substitution such that $vars(first(d)\theta) \cap vars(clauses(d)) = \emptyset$ and let L' be a labelling function for all variables in $range(\theta)$. Then the application of θ to d, $d\theta = S'_1, ..., S'_n$ is defined if for each state $S_i = \langle W_i,\ k_i,\ L_i,\ \theta_i \rangle \in d$, we have that $S'_i = \langle W'_i,\ k_i,\ L'_i,\ \theta'_i \rangle$ where

- if $k_i \neq \kappa$ then $\exists \theta'_i = lmgu(A\theta, A')$ relative to $L_i \cup L'$ where $k_i = \forall x_1 ... \forall x_n.(G' \supset A')$ and $\langle A, P, n \rangle \in W_{i-1}$. In that case $W'_i = \{\langle G, P, n \rangle \theta'_i \mid \langle G, P, n \rangle \in W_i - \{\langle A, P, n \rangle\}\} \cup \{\langle G', P, n \rangle \theta'_i\}$ and L'_i is the labelling induced by θ'_i from $L_i \cup L'$.

- if $k_i \neq \kappa$ then $\exists \theta'_i = lmgu(A\theta, A')$ relative to $L_i \cup L'$ where $k_i = \forall x_1 ... \forall x_n.\, A'$ and $\langle A, P, n \rangle \in W_{i-1}$. In that case $W'_i = \{\langle G, P, n \rangle \theta'_i \mid \langle G, P, n \rangle \in W_i - \{\langle A, P, n \rangle\}\}$ and L'_i is the labelling induced by θ'_i from $L_i \cup L'$.

- if $k_i = \kappa$ then S'_i is a proper state, where $W'_i = W_i\theta$, L'_i is the new induced labelling function for all variables in $range(\theta) \cup (Dom(L_1) - Dom(\theta))$ and $\theta'_i = \epsilon$.

Note that the substitution applied to a derivation attempts to reconstruct the derivation, starting from a new goal (more instantiated or just renamed) using the same clauses, until a failure of a $lmgu$ or a substitution that yields a non proper state. In any other case the substitution has success and the result is a derivation sequence.

Definition 5.5. Let $d = S_1, ..., S_n$ be a derivation and G be a goal, the *instantiation* $\delta_G(d)$ of d is defined if there exists a labelled substitution θ, such that $G = G'\theta$ and if $d\theta$ is defined, where $first(d) = \{G'\}$. In that case $\delta_G(d) = d\theta$.

Now we describe and define all semantic operators and auxiliary functions utilized in our denotational semantic.

- The operator lfp is the least fixed point over functions $f : \mathbb{I} \to \mathbb{I}$ and is defined as $\mathsf{lfp}\, f = f{\uparrow}\omega$ where $f^{n+1} = f(f^n)$ and $f^0 = \bot$.

- The operator *tree*, given a clause $G \supset p(\mathbf{t})$, a level n and a program P, creates the interpretation $I = \{\langle p(\mathbf{x}), \{d_0, d_1\}\rangle\}$ where $d_0 = \langle\{\langle p(\mathbf{x}), P, n\rangle\}, \kappa, L_0, \epsilon\rangle$ and $d_1 = \langle\{\langle p(\mathbf{x}), P, n\rangle\}, \kappa, L_0, \epsilon\rangle \to \langle\{\langle G, P, n\rangle\,\theta\}, G \supset p(\mathbf{t})\rho, L_1, \theta\rangle$. Given a clause $p(\mathbf{t})$, the operator creates the interpretation $I = \{\langle p(\mathbf{x}), \{d_0, d_1\}\rangle\}$ where $d_0 = \langle\{\langle p(\mathbf{x}), P, n\rangle\}, \kappa, L_0, \epsilon\rangle$ and $d_1 = \langle\{\langle p(\mathbf{x}), P, n\rangle\}, \kappa, L_0, \epsilon\rangle \to \langle\emptyset, p(\mathbf{t})\rho, L_1, \theta\rangle$. We have also, $L_0 = \mathcal{L}(P) \cup \{\langle \mathbf{x}, 0\rangle\}$, $L' = L_0 \cup \{\langle y, n\rangle \mid y \in range(\rho)\}$ where ρ is a renaming over $\mathbf{x}$ and L_1 is the labelling induced by $\theta = \{\mathbf{x}/\mathbf{t}\rho\}$ from L'.

- The operator $+$ is the sum of classes of interpretations:

$$C_1 + C_2 := \lambda G.C_1(G) \cup C_2(G)$$

  i.e., it assigns to each goal the union of its interpretations in C_1 and C_2.

- The operator $\bowtie$ computes the concatenation of the collection C_1 by C_2. It is defined as

$$C_1 \bowtie C_2 := \lambda G.C_1(G) \cup \{\ d_1 :: d_2' \mid d_1 \in C_1(G) \text{ and } d_2' = d_2\theta \text{ where } \text{ is the substitution applied in the last state of } d_1 \text{ and } d_2 \in C_2(G)\}$$

  By means of this operator the function $\mathcal{C}$ constructs the maximum number of derivations that can be obtained for a pure goal. Precisely by this reason the denotational semantics of a program is an interpretation.

- The operator $\cdot$ makes the instantiation of an interpretation I with an atom A. Is defined as

$$A \cdot I := \lambda A.\{\delta_A(d) \mid d \in I(A)\}$$

  This operator assigns to A all the instantiated derivations in I.

- The product of C_1 and C_2 is

$$C_1 \otimes C_2 = \lambda G.\{\ S_0, (d_1 \sharp d_2) \mid G = G_1 \wedge G_2,\ d_1 \in C_1(G_1),\ d_2 \in C_2(G_2) \text{ and } S_0 = \langle\{\langle G, P, n\rangle\}, \kappa, L, \epsilon\rangle, \text{ where } P \text{ and } n \text{ are the program and level occurring in the unique goal of } first(d_1) \text{ and } first(d_2)\}$$

  Note that the collection defined by this operator only operates over conjunctions of goals. The operator takes the derivations associated to each component of the conjunction and fuses them (when it is possible) in an unique derivation, adding a head state (S_0) containing the conjunction goal.

- The extension of the union of C_1 and C_2 is

$$C_1 \oplus C_2 = \lambda G.\{ \quad S_0, d \mid G = G_1 \vee G_2,\ d \in C_1(G_1) \cup C_2(G_2),\ d_1 \in C_1(G_1), d_2 \in C_2(G_2),\ S_0 = \langle \{\langle G, P, n\rangle\}\ ,\ \kappa,\ L,\ \epsilon\rangle\ , \text{ where } P \text{ and } n \text{ are the program and level occurring in the unique goal of } first(d_1) \text{ and } first(d_2)\}$$

  This operator is very similar to the sum of collections, it differs only in the head state added by the operator to the resulting derivations.

- The existential quantifier of variable x in a level n over a collection C is defined as

$$\exists^n_x C = \lambda G.\{ \quad S_0, d' \mid G = \exists x.G',\ d \in C(G'),\ d' = d[y/x], \text{ where } L(y) = n \text{ for a new variable } y, P \text{ is the program occurring in } first(d) \text{ and } S_0 = \langle \{\langle G, P, n\rangle\}\ ,\ \kappa,\ L,\ \epsilon\rangle\}$$

- The universal quantifier of variable x in a level n over a collection C is defined as

$$\forall^n_x C = \lambda G.\{ \quad S_0, d' \mid G = \forall x.G',\ d \in C(G'),\ d' = d[c/x], \text{ where } L(c) = n \text{ for a new constant } c, P \text{ is the program occurring in } first(d) \text{ and } S_0 = \langle \{\langle G, P, n\rangle\}\ ,\ \kappa,\ L,\ \epsilon\rangle\}$$

- The composition of two interpretations I_1 and I_2 is $I_1 \uplus I_2 = \mathsf{lfp}\, f_{I_1,I_2}$ where

$$f_{I_1,I_2}(I) = \begin{cases} I_1 & \text{if } I = \bot \\ (I \bowtie\!\!\lhd I_2) + (I_2 \bowtie\!\!\lhd I) & \text{otherwise.} \end{cases}$$

  This operator defines an OR-composition between program interpretations. It takes the least fixed point of all possible mutually extensions of both interpretations. It is easy to show that this operator is commutative analyzing the properties of the following operator.

- The operator $\bowtie$ computes the compatible extension of a collection C_1 by a collection C_2 and is defined as

$$C_1 \bowtie\!\!\lhd C_2 = \lambda G.C_1(G) \cup \{ \quad d_1 :: (d' \sharp d'_2) \text{ where } d_1 \in C_1(G),\ d' = S_n = \langle W_n, k_n, L_n, \theta_n\rangle\ ,\ d_1 = S_1, ..., S_n \text{ and } d'_2 = \delta_G(d_2) \text{ with } G \in W_n \text{ and } d_2 \in C_2(G)\}$$

  This operator extends the derivation in C_1 with all possible instances of the derivations in C_2. In other words, this operator continues an stopped computation (represented by a derivation of C_1), "executing" the steps of other computation (represented by a derivation of C_2). Note that the second derivation only reduces (derives) a pending goal in d_1, by this reason the extended derivation has all the remaining goals pending.

Note that the function $\mathcal{P}[\![P]\!]$ must be continuous in order to have a least fixed point. Hence, all the operators defined above must be also continuous. This can be easily proved using standard techniques.

6 Operational Semantics

The procedure presented in section 3 can be seen as a transition system between configuration states of an operational semantics. This operational semantics describes, in the most concrete way, the resolution principle of the *fohh*-programs. We present now an extended semantics based on derivations.

Definition 6.1. Let P be a *fohh*-program. The *behavior* of a query G in P is defined as

$$\mathcal{B}[\![G \ in \ P]\!] = \lambda G.\, \{S_0 \rightarrow^* S_n \mid S_0 \text{ is the initial state } \langle \{\langle G, P, 0\rangle\}, \kappa, \mathbf{0}, \epsilon\rangle\}/_{\equiv_{\mathbb{C}}}$$

The behavior modulo variance is a collection that assigns to every goal G the equivalence classes of derivation sequences which initial state contains only the goal G. The derivation sequences assigned to goals are constructed in a top-down way as described in definition 3.5, so we have not only all the successful derivations, but also all failed and infinite derivations, including their prefixes.

Definition 6.2. Let P be a *fohh*-program. Its *top-down intuitionistic derivation denotation* is

$$\mathcal{O}[\![P]\!] = \bigcup_{p(\mathbf{x}) \in Goals} \mathcal{B}[\![p(\mathbf{x}) \ in \ P]\!]$$

Note that $\mathcal{O}[\![P]\!]$ is defined only for pure atoms, assigning a non empty set of derivations to the atoms that match any head of clause in the program P.

Theorem 6.1. *Let P be a* fohh*-program, then*

$$\mathcal{O}[\![P]\!] = \mathsf{lfp}\, \mathcal{P}[\![P]\!],$$

moreover

$$\mathcal{B}[\![G \ in \ P]\!] = \mathcal{Q}[\![G \ in \ P]\!].$$

Using the previous denotations we can define the equivalence of two programs P_1, P_2 as

$$P_1 \approx P_2 \iff \forall G \in Goals,\ \mathcal{B}[\![G \ in \ P_1]\!] = \mathcal{B}[\![G \ in \ P_2]\!].$$

7 Semantics Properties

The program denotation $\mathcal{O}[\![P]\!]$ has several interesting properties. These can all be viewed as compositionality properties, and are based on the semantics operators defined in section 5. The first result is a theorem that shows the compositionality of the semantic function $\mathcal{B}$ w.r.t. procedure calls and the different syntactic operators of the language.

Theorem 7.1. *Let A be an atom , D be a program clause and G, G_1, G_2 be goals. Then*

- $\mathcal{B}[\![A \ in \ P]\!] = A \cdot \mathcal{O}[\![P]\!]$,
- $\mathcal{B}[\![G_1 \wedge G_2 \ in \ P]\!] = \mathcal{B}[\![G_1 \ in \ P]\!] \otimes \mathcal{B}[\![G_2 \ in \ P]\!]$,
- $\mathcal{B}[\![G_1 \vee G_2 \ in \ P]\!] = \mathcal{B}[\![G_1 \ in \ P]\!] \oplus \mathcal{B}[\![G_2 \ in \ P]\!]$,

- $\mathcal{B}[\![\exists x.G \ in \ P]\!] = \exists_x \mathcal{B}[\![G \ in \ P]\!]$,
- $\mathcal{B}[\![\forall x.G \ in \ P]\!] = \forall_x \mathcal{B}[\![G \ in \ P]\!]$,
- $\mathcal{B}[\![D \supset G \ in \ P]\!] = \mathcal{B}[\![G \ in \ P \cup \{D\}]\!]$.

It can be easily shown that the denotation $\mathcal{O}$ is correct and minimal w.r.t. $\approx$, i.e.,

$$P_1 \approx P_2 \Longleftrightarrow \mathcal{O}[\![P_1]\!] = \mathcal{O}[\![P_2]\!]$$

Finally we can state the following theorem, which establishes the OR-composition between program denotations

Theorem 7.2. *Let P_1 and P_2 be programs. Then $\mathcal{O}[\![P_1 \cup P_2]\!] = \mathcal{O}[\![P_1]\!] \uplus \mathcal{O}[\![P_2]\!]$*

8 Conclusions

We have presented two equivalent semantic definitions for *fohh*-programs based on derivations; one with a top-down construction and other with a bottom-up specification. These semantics will help us to find models which really captures the operational and denotational semantics and are therefore useful for defining program equivalences and for semantics-based program analysis.

As mentioned in the introduction our semantics will be used to study the properties of *fohh*-programs. The characteristics of our definitions allow us to claim that they constitute a good basis for the construction of a semantic framework to systematically derive more abstract semantics, using the formal tools of abstract interpretation, and to study the relationship between semantics at different levels of abstraction.

References

[1] M. Comini and M. C. Meo. Compositionality properties of *SLD*-derivations. *Theoretical Computer Science*, 1997. To appear.

[2] 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.

[3] 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.

[4] S. E. Finkelstein, P. Freyd, and J. Lipton. A New Framework for Declarative Programming. *Submitted to Theoretical Computer Science (revised)*, 1996.

[5] M. Gabbrielli and G. Levi. On the Semantics of Logic Programs. In J. Leach Albert, B. Monien, and M. Rodriguez-Artalejo, editors, *Automata, Languages and Programming, 18th International Colloquium*, volume 510 of *Lecture Notes in Computer Science*, pages 1–19. Springer-Verlag, 1991.

[6] R. Harrop. Concerning formulas of the types $A \rightarrow B \vee C$, $A \rightarrow (Ex)B(x)$ in intuitionistic formal systems. *Journal of Symbolic Logic*, pages 27–32, 1960.

[7] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1987. Second edition.

[8] D. Miller. Hereditary Harrop Formula and Logic Programming. In *VIII International Congress of Logic, methodology and Philosophy of Science*, 1987.

[9] D. Miller. A Logical Analysis of Modules in Logic Programming. *Journal of Logic Programming*, 6:79–108, 1989.

[10] G. Nadathur. A Proof Procedure for the Logic of Hereditary Harrop Formulas. *Journal of Automated Reasoning*, 11:115–145, 1993.