TLP **14** (4–5): 725–737, 2014. © Cambridge University Press 2014
doi:10.1017/S1471068414000313

*Minimum Model Semantics for Extensional Higher-order Logic Programming with Negation**

ANGELOS CHARALAMBIDIS
Department of Informatics & Telecommunications, University of Athens, Greece
(*e-mail:* `a.charalambidis@di.uoa.gr`)

ZOLTÁN ÉSIK
Department of Computer Science, University of Szeged, Hungary
(*e-mail:* `ze@inf.u-szeged.hu`)

PANOS RONDOGIANNIS
Department of Informatics & Telecommunications, University of Athens, Greece
(*e-mail:* `prondo@di.uoa.gr`)

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Abstract

Extensional higher-order logic programming has been introduced as a generalization of classical logic programming. An important characteristic of this paradigm is that it preserves all the well-known properties of traditional logic programming. In this paper we consider the semantics of negation in the context of the new paradigm. Using some recent results from non-monotonic fixed-point theory, we demonstrate that every higher-order logic program with negation has a unique *minimum* infinite-valued model. In this way we obtain the first purely model-theoretic semantics for negation in extensional higher-order logic programming. Using our approach, we resolve an old paradox that was introduced by W. W. Wadge in order to demonstrate the semantic difficulties of higher-order logic programming.

1 Introduction

Extensional higher-order logic programming has been proposed (Wadge 1991; Charalambidis *et al.* 2010; Charalambidis *et al.* 2013) as a generalization of classical logic programming. The key idea behind this paradigm is that all predicates defined in a program denote sets and therefore one can use standard extensional set theory in order to understand their meaning and to reason about them. For example, consider

* This research is being supported by the Greek General Secretariat for Research and Technology, the National Development Agency of Hungary, and the European Commission (European Regional Development Fund) under a Greek-Hungarian intergovernmental programme of Scientific and Technological collaboration. Project title: "Extensions and Applications of Fixed Point Theory for Non-Monotonic Formalisms". It is also supported by grant no. ANN 110883 from the National Foundation of Hungary for Scientific Research.

the following simple extensional higher-order program (Charalambidis *et al.* 2013) stating that a band (musical ensemble) is a group that has at least a singer and a guitarist:

```
band(B):-singer(S),B(S),guitarist(G),B(G).
```

Suppose that we also have a database of musicians:

```
singer(sally).
singer(steve).
guitarist(george).
guitarist(grace).
```

We can then ask the query `?-band(B)`. Since predicates denote sets, an extensional higher-order language will return answers such as $\mathtt{B} = \{\mathtt{sally}, \mathtt{george}\} \cup \mathtt{L}$, having the meaning that every set that contains at least `sally` and `george` is a potential band.

A consequence of the set-theoretic nature of extensional higher-order logic programming is the fact that its semantics and its proof theory smoothly extend the corresponding ones for traditional (ie., first-order) logic programming. In particular, every program has a unique minimum Herbrand model which is the greatest lower bound of all Herbrand models of the program and the least fixed-point of an immediate consequence operator associated with the program; moreover, there exists an SLD resolution proof-procedure which is sound and complete with respect to the minimum model semantics.

One basic property of all the higher-order predicates that can be defined in the language of (Charalambidis *et al.* 2013) is that they are *monotonic*. Intuitively, the monotonicity property states that if a predicate is true of a relation `R` then it is also true of every superset of `R`. In the above example, it is clear that if `band` is true of a relation `B` then it is also true of any band that is a superset of `B`. However, there are many natural higher-order predicates that are *non-monotonic*. Consider for example a predicate `single_singer_band` which (apparently) defines a band that has a unique singer:

```
single_singer_band(B):-band(B),not two_singers(B).
two_singers(B):-B(S1),B(S2),singer(S1),singer(S2),not(S1=S2).
```

The predicate `single_singer_band` is obviously non-monotonic since it is satisfied by the set $\{\mathtt{sally}, \mathtt{george}\}$ but not by the set $\{\mathtt{sally}, \mathtt{steve}, \mathtt{george}\}$. In other words, the semantics of (Charalambidis *et al.* 2013) is not applicable to this extended higher-order language. We are therefore facing the same problem that researchers faced more than twenty years ago when they attempted to provide a sensible semantics to classical logic programs with negation; the only difference is that the problem now reappears in a much more general context, namely in the context of higher-order logic programming.

The solution we adopt is relatively simple to state (but non-trivial to materialize): it suffices to generalize the well-founded construction (van Gelder *et al.* 1991; Przymusinski 1989) to higher-order programs. For this purpose, we have found convenient to use a relatively recent logical characterization of the well-founded semantics through an infinite-valued logic (Rondogiannis and Wadge 2005) and also the recent abstract fixed-point theory for non-monotonic functions developed in (Ésik and Rondogiannis 2013; Ésik and Rondogiannis 2014). This brings us to the two main contributions of the present paper:

- We provide the first model-theoretic semantics for extensional higher-order logic programming with negation. In this way we initiate the study of a non-monotonic formalism that is much broader than classical logic programming with negation.
- We provide further evidence that extensional higher-order logic programming is a natural generalization of classical logic programming, by showing that all the well-known properties of the latter also hold for the new paradigm.

In the next section we provide an introduction to the proposed semantics for higher-order logic programming and the remaining sections provide the formal development of this semantics. The proofs of all the results have been moved to corresponding appendices.

2 An Intuitive Overview of the Proposed Semantics

The starting point for the semantics proposed in this paper is the *infinite-valued semantics* for ordinary logic programs with negation, as introduced in (Rondogiannis and Wadge 2005). In this section we give an intuitive introduction to the infinite-valued approach and discuss how it can be extended to the higher-order case.

The infinite-valued approach was introduced in order to provide a *minimum model* semantics to logic programs with negation. As we are going to see shortly, it is compatible with the well-founded semantics but it is purely model-theoretic[1]. The main idea of this approach can be explained with a simple example. Consider the program:

```
p ←
r ← ∼p
s ← ∼q
```

Under the well-founded semantics both `p` and `s` receive the value *True*. However, `p` is in some sense "truer" than `s`. Namely, `p` is true because there is a rule which says so, whereas `s` is true only because we are never obliged to make `q` true. In a sense, `s` is true only by default. This gave the idea of adding a "default" truth value T_1 just below the "real" truth T_0, and (by symmetry) a weaker false value F_1 just above ("not as false as") the real false F_0. We can then understand negation-as-failure as combining ordinary negation with a weakening. Thus $\sim F_0 = T_1$ and $\sim T_0 = F_1$. Since negations can effectively be iterated, the infinite-valued approach requires a whole sequence $\ldots, T_3, T_2, T_1$ of weaker and weaker truth values below T_0 but above the neutral value 0; and a mirror image sequence $F_1, F_2, F_3, \ldots$ above F_0 and below 0. In fact, to capture the well-founded model in full generality, we need a T_α and a F_α for every countable ordinal α. In other words, the underlying truth domain of

[1] In the same way that the equilibrium logic approach of (Pearce 1996) gives a purely logical reconstruction of the stable model semantics.

the infinite-valued approach is:

$$F_0 < F_1 < \cdots < F_\omega < \cdots < F_\alpha < \cdots < 0 < \cdots < T_\alpha < \cdots < T_\omega < \cdots < T_1 < T_0$$

As shown in (Rondogiannis and Wadge 2005), every logic program P with negation has a unique *minimum* infinite-valued model M_P. Notice that M_P is minimum with respect to a relation $\sqsubseteq$ which compares interpretations in a stage-by-stage manner (see (Rondogiannis and Wadge 2005) for details). As it is proven in (Rondogiannis and Wadge 2005), if we collapse all the T_α and F_α to *True* and *False* respectively, we get the well-founded model. For the example program above, the minimum model is $\{(\mathtt{p}, T_0), (\mathtt{q}, F_0), (\mathtt{r}, F_1), (\mathtt{s}, T_1)\}$. This collapses to $\{(\mathtt{p}, \mathit{True}), (\mathtt{q}, \mathit{False}), (\mathtt{r}, \mathit{False}), (\mathtt{s}, \mathit{True})\}$, which is the well-founded model of the program.

As shown in (Rondogiannis and Wadge 2005), one can compute the minimum infinite-valued model as the least fixed point of an operator T_P. It can easily be seen that T_P is *not* monotonic with respect to the ordering relation $\sqsubseteq$ and therefore one can not obtain the least fixed point using the classical Knaster-Tarski theorem. However, T_P possesses some form of *partial monotonicity*. More specifically, as it is shown in (Rondogiannis and Wadge 2005; Ésik and Rondogiannis 2014), T_P is α-monotonic for all countable ordinals α, a property that guarantees the existence of the least fixed point. Loosely speaking, the property of T_P being α-monotonic means that the operator is monotonic when we restrict attention to interpretations that are equal for all levels of truth values that are less than α. In other words, T_P is monotonic in stages (but not overall monotonic).

The T_P operator is a higher-order function since it takes as argument an interpretation and returns an interpretation as the result. This observation leads us to the main concept that helps us extend the infinite-valued semantics to the higher-order case. The key idea is to demonstrate that the denotation of every expression of predicate type in our higher-order language, is α-monotonic for all ordinals α (see Lemma 5). This property ensures that the immediate consequence operator of every program is also α-monotonic for all α (see Lemma 7), and therefore it has a least fixed-point which is a model of the program. Actually, this same model can also be obtained as the greatest lower bound of all the Herbrand models of the program (see Theorem 2, the *model intersection theorem*). In other words, the semantics of extensional higher-order logic programming with negation preserves all the familiar properties of classical logic programming and can therefore be considered as a natural generalization of the latter.

3 Non-Monotonic Fixed Point Theory

The main results of the paper will be obtained using some recent results from non-monotonic fixed point theory (Ésik and Rondogiannis 2013; Ésik and Rondogiannis 2014). The key objective of this area of research is to obtain novel fixed point results regarding functions that are not necessarily monotonic. In particular, the results obtained in (Ésik and Rondogiannis 2013; Ésik and Rondogiannis 2014) generalize the classical results of monotonic fixed-point theory (namely Kleene's theorem and also the Knaster-Tarski theorem). In this section we provide the necessary material from (Ésik and Rondogiannis 2013; Ésik and Rondogiannis 2014) that will be needed in the next sections.

Suppose that $(L, \leqslant)$ is a complete lattice in which the least upper bound operation is denoted by $\bigvee$ and the least element is denoted by $\perp$. Let $\kappa > 0$ be a fixed ordinal. We assume that for each ordinal $\alpha < \kappa$, there exists a preordering $\sqsubseteq_\alpha$ on L. We write $x =_\alpha y$ iff $x \sqsubseteq_\alpha y$ and $y \sqsubseteq_\alpha x$. We define $x \sqsubset_\alpha y$ iff $x \sqsubseteq_\alpha y$ but $x =_\alpha y$ does not hold. Moreover, we write $x \sqsubset y$ iff $x \sqsubset_\alpha y$ for some $\alpha < \kappa$. Finally, we define $x \sqsubseteq y$ iff $x \sqsubset y$ or $x = y$.

Let $x \in L$ and $\alpha < \kappa$. We define $(x]_\alpha = \{y : \forall \beta < \alpha \; x =_\beta y\}$.

A key property that will be used throughout the paper is that if the above preordering relations satisfy certain simple axioms, then the structure $(L, \sqsubseteq)$ is a complete lattice; moreover, every function $f : L \to L$ that satisfies some restricted form of monotonicity, has a least fixed point. These ideas are formalized by the following definitions and results.

Definition 1
Let $(L, \leqslant)$ be a complete lattice equipped with preorderings $\sqsubseteq_\alpha$ for all $\alpha < \kappa$. Then, L will be called a *basic model* if and only if it satisfies the following axioms:

1. For all $x, y \in L$ and all $\alpha < \beta < \kappa$, if $x \sqsubseteq_\beta y$ then $x =_\alpha y$.
2. For all $x, y \in L$, if $x =_\alpha y$ for all $\alpha < \kappa$ then $x = y$.
3. Let $x \in L$ and $\alpha < \kappa$. Let $X \subseteq (x]_\alpha$. Then, there exists y (denoted by $\bigsqcup_\alpha X$) such that $X \sqsubseteq_\alpha y$[2] and for all $z \in (x]_\alpha$ such that $X \sqsubseteq_\alpha z$, it holds $y \sqsubseteq_\alpha z$ and $y \leqslant z$.
4. If $x_j, y_j \in L$ and $x_j \sqsubseteq_\alpha y_j$ for all $j \in J$ then $\bigvee\{x_j : j \in J\} \sqsubseteq_\alpha \bigvee\{y_j : j \in J\}$.

Lemma 1
Let L be a basic model. Then, $(L, \sqsubseteq)$ is a complete lattice.

Definition 2
Let A, B be basic models and let $\alpha < \kappa$. A function $f : A \to B$ is called α-monotonic if for all $x, y \in A$ if $x \sqsubseteq_\alpha y$ then $f(x) \sqsubseteq_\alpha f(y)$.

It should be noted that even if a function f is α-monotonic for all $\alpha < \kappa$, then it need not be necessarily monotonic with respect to the relation $\sqsubseteq$ (for a counterexample, see (Rondogiannis and Wadge 2005, Example 5.7, pages 453–454)). Therefore, the standard tools of classical fixed point theory (such as the Knaster-Tarski theorem), do not suffice in order to find the least fixed point of f with respect to the relation $\sqsubseteq$.

Let us denote by $[A \overset{m}{\to} B]$ the set of functions from A to B that are α-monotonic for all $\alpha < \kappa$.

Theorem 1
Let L be a basic model and assume that $f \in [L \overset{m}{\to} L]$. Then, f has a $\sqsubseteq$-least pre-fixed point, which is also the $\sqsubseteq$-least fixed point of f.

[2] We write $X \sqsubseteq_\alpha y$ iff forall $x \in X$ it holds $x \sqsubseteq_\alpha y$.

The above theorem will be our main tool for establishing the fact that the immediate consequence operator of any extensional higher order logic program, always has a least fixed point, which is a model of the program.

4 The Syntax of the Higher-Order Language $\mathscr{H}$

In this section we introduce the higher-order language $\mathscr{H}$, which extends classical first-order logic programming to a higher-order setting. The language $\mathscr{H}$ is based on a simple type system that supports two base types: o, the boolean domain, and ι, the domain of individuals (data objects). The composite types are partitioned into three classes: functional (assigned to individual constants, individual variables and function symbols), predicate (assigned to predicate constants and variables) and argument (assigned to parameters of predicates).

Definition 3
A type can either be functional, predicate, argument, denoted by σ, π and ρ respectively and defined as:

$$\sigma := \iota \mid \iota \to \sigma$$
$$\pi := o \mid \rho \to \pi$$
$$\rho := \iota \mid \pi$$

We will use τ to denote an arbitrary type (either functional, predicate or argument one).

As usual, the binary operator $\to$ is right-associative. A functional type that is different than ι will often be written in the form $\iota^n \to \iota$, $n \geqslant 1$ (which stands for $\iota \to \iota \to \cdots \to \iota$ $(n+1)$-times). Moreover, it can be easily seen that every predicate type π can be written uniquely in the form $\rho_1 \to \cdots \to \rho_n \to o$, $n \geqslant 0$ (for $n = 0$ we assume that $\pi = o$). We can now proceed to the definition of $\mathscr{H}$, starting from its alphabet and continuing with expressions and program clauses:

Definition 4
The *alphabet* of the higher-order language $\mathscr{H}$ consists of the following:

1. *Predicate variables* of every predicate type π (denoted by capital letters such as $\mathsf{P}, \mathsf{Q}, \mathsf{R}, \ldots$).
2. *Predicate constants* of every predicate type π (denoted by lowercase letters such as $\mathsf{p}, \mathsf{q}, \mathsf{r}, \ldots$).
3. *Individual variables* of type ι (denoted by capital letters such as $\mathsf{X}, \mathsf{Y}, \mathsf{Z}, \ldots$).
4. *Individual constants* of type ι (denoted by lowercase letters such as $\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots$).
5. *Function symbols* of every functional type $\sigma \neq \iota$ (denoted by lowercase letters such as $\mathsf{f}, \mathsf{g}, \mathsf{h}, \ldots$).
6. The following *logical constant symbols*: the constants false and true of type o; the equality constant $\approx$ of type $\iota \to \iota \to o$; the generalized disjunction and conjunction constants $\bigvee_\pi$ and $\bigwedge_\pi$ of type $\pi \to \pi \to \pi$, for every predicate type π; the generalized inverse implication constants $\leftarrow_\pi$, of type $\pi \to \pi \to o$, for

every predicate type π; the existential quantifier $\exists_\rho$, of type $(\rho \to o) \to o$, for every argument type ρ; the negation constant $\sim$ of type $o \to o$.
7. The *abstractor* λ and the parentheses "(" and ")".

The set consisting of the predicate variables and the individual variables of $\mathscr{H}$ will be called the set of *argument variables* of $\mathscr{H}$. Argument variables will be usually denoted by V and its subscripted versions.

Definition 5
The set of *expressions* of the higher-order language $\mathscr{H}$ is defined as follows:

1. Every predicate variable (respectively, predicate constant) of type π is an expression of type π; every individual variable (respectively, individual constant) of type ι is an expression of type ι; the propositional constants false and true are expressions of type o.
2. If f is an n-ary function symbol and $\mathsf{E}_1, \ldots, \mathsf{E}_n$ are expressions of type ι, then $(\mathsf{f}\ \mathsf{E}_1 \cdots \mathsf{E}_n)$ is an expression of type ι.
3. If E_1 is an expression of type $\rho \to \pi$ and E_2 is an expression of type ρ, then $(\mathsf{E}_1\ \mathsf{E}_2)$ is an expression of type π.
4. If V is an argument variable of type ρ and E is an expression of type π, then $(\lambda \mathsf{V}.\mathsf{E})$ is an expression of type $\rho \to \pi$.
5. If $\mathsf{E}_1, \mathsf{E}_2$ are expressions of type π, then $(\mathsf{E}_1 \bigwedge_\pi \mathsf{E}_2)$ and $(\mathsf{E}_1 \bigvee_\pi \mathsf{E}_2)$ are expressions of type π.
6. If E is an expression of type o, then $(\sim\mathsf{E})$ is an expression of type o.
7. If $\mathsf{E}_1, \mathsf{E}_2$ are expressions of type ι, then $(\mathsf{E}_1 \approx \mathsf{E}_2)$ is an expression of type o.
8. If E is an expression of type o and V is a variable of type ρ then $(\exists_\rho \mathsf{V}\,\mathsf{E})$ is an expression of type o.

To denote that an expression E has type τ we will write $\mathsf{E} : \tau$. The notions of *free* and *bound* variables of an expression are defined as usual. An expression is called *closed* if it does not contain any free variables.

Definition 6
A *program clause* is a clause $\mathsf{p} \leftarrow_\pi \mathsf{E}$ where p is a predicate constant of type π and E is a closed expression of type π. A *program* is a finite set of program clauses.

Example 1
The `subset` predicate can be defined in $\mathscr{H}$ as follows:

$$\texttt{subset} \leftarrow_{\pi \to \pi \to o} \lambda \texttt{P}.\lambda \texttt{Q}.\ {\sim}\exists \texttt{X}((\texttt{P X}) \wedge {\sim}(\texttt{Q X}))$$

The `subset` predicate is defined by a λ-expression (which obviates the need to have the formal parameters of the predicate in the left-hand side of the definition). Moreover, in the right-hand side we have an explicit existential quantifier for the variable `X` (in Prolog, if a variable appears in the body of a clause but not in the head, then it is implicitly existentially quantified).

5 The Semantics of the Higher-Order Language $\mathscr{H}$

In this section we specify the semantics of $\mathscr{H}$. We start with the semantics of types and proceed to the semantics of expressions.

The meaning of the boolean type o is equal to a partially ordered set $(V, \leqslant)$ of truth values. The number of truth values of V will be specified with respect to an ordinal $\kappa > 0$. All the results of the paper hold for *every* initial selection of κ. The set $(V, \leqslant)$ is therefore

$$F_0 < F_1 < \cdots < F_\alpha < \cdots < 0 < \cdots < T_\alpha < \cdots < T_1 < T_0$$

where $\alpha < \kappa$.

Definition 7
The *order* of a truth value is defined as follows: $\mathit{order}(T_\alpha) = \alpha$, $\mathit{order}(F_\alpha) = \alpha$ and $\mathit{order}(0) = +\infty$.

We can now define the meaning of all the types of our language as well as the corresponding relations $\leqslant$ and $\sqsubseteq_\alpha$. This is performed in the following definitions:

Definition 8
We define the relation $\sqsubseteq_\alpha$ on the set V for each $\alpha < \kappa$ as follows:

1. $x \sqsubseteq_\alpha x$ if $\mathit{order}(x) < \alpha$;
2. $F_\alpha \sqsubseteq_\alpha x$ and $x \sqsubseteq_\alpha T_\alpha$ if $\mathit{order}(x) \geqslant \alpha$;
3. $x \sqsubseteq_\alpha y$ if $\mathit{order}(x), \mathit{order}(y) > \alpha$.

Notice that $x =_\alpha y$ iff either $x = y$ or $\mathit{order}(x) > \alpha$ and $\mathit{order}(y) > \alpha$.

Definition 9
Let D be a nonempty set. Then:

- $[\![\iota]\!]_D = D$, and $\leqslant_\iota$ is the trivial partial order such that $d \leqslant_\iota d$, for all $d \in D$;
- $[\![\iota^n \to \iota]\!]_D = D^n \to D$. A partial order in this case will not be needed;
- $[\![o]\!]_D = V$, and $\leqslant_o$ is the partial order of V;
- $[\![\iota \to \pi]\!]_D = D \to [\![\pi]\!]_D$, and $\leqslant_{\iota \to \pi}$ is the partial order defined as follows: for all $f, g \in [\![\iota \to \pi]\!]_D$, $f \leqslant_{\iota \to \pi} g$ iff $f(d) \leqslant_\pi g(d)$ for all $d \in D$;
- $[\![\pi_1 \to \pi_2]\!]_D = [[\![\pi_1]\!]_D \overset{m}{\to} [\![\pi_2]\!]_D]$, and $\leqslant_{\pi_1 \to \pi_2}$ is the partial order defined as follows: for all $f, g \in [\![\pi_1 \to \pi_2]\!]_D$, $f \leqslant_{\pi_1 \to \pi_2} g$ iff $f(d) \leqslant_{\pi_2} g(d)$ for all $d \in [\![\pi_1]\!]_D$.

The subscripts in the above partial orders will often be omitted when they are obvious from context.

Definition 10
Let D be a nonempty set and $\alpha < \kappa$. Then:

- The relation $\sqsubseteq_\alpha$ on $[\![o]\!]_D$ is the relation $\sqsubseteq_\alpha$ on V.
- The relation $\sqsubseteq_\alpha$ on $[\![\rho \to \pi]\!]_D$ is defined as follows: $f \sqsubseteq_\alpha g$ iff $f(d) \sqsubseteq_\alpha g(d)$ for all $d \in [\![\rho]\!]_D$. Moreover, $f \sqsubset_\alpha g$ iff $f \sqsubseteq_\alpha g$ and $f(d) \sqsubset_\alpha g(d)$ for some $d \in [\![\rho]\!]_D$.

The following lemma expresses the fact that all the predicate types correspond to semantic domains that are both complete lattices and basic models:

Lemma 2
Let D be a nonempty set and π be a predicate type. Then, $([\![\pi]\!]_D, \leqslant_\pi)$ is a complete lattice and a basic model.

We now proceed to formally define the semantics of $\mathcal{H}$:

Definition 11
An intepretation I of $\mathcal{H}$ consists of:

1. a nonempty set D called the domain of I;
2. an assignment to each individual constant symbol c, of an element $I(\mathsf{c}) \in D$;
3. an assignment to each predicate constant $\mathsf{p} : \pi$ of an element $I(\mathsf{p}) \in [\![\pi]\!]_D$;
4. an assignment to each function symbol $\mathsf{f} : \iota^n \to \iota$ of a function $I(\mathsf{f}) \in D^n \to D$.

Definition 12
Let D be a nonempty set. A state s of $\mathcal{H}$ over D is a function that assigns to each argument variable V of type ρ of $\mathcal{H}$, of an element $s(\mathsf{V}) \in [\![\rho]\!]_D$.

Definition 13
Let I be an interpretation of $\mathcal{H}$, let D be the domain of I, and let s be a state over D. Then, the semantics of expressions of $\mathcal{H}$ with respect to I and s, is defined as follows:

1. $[\![\mathsf{false}]\!]_s(I) = F_0$
2. $[\![\mathsf{true}]\!]_s(I) = T_0$
3. $[\![\mathsf{c}]\!]_s(I) = I(\mathsf{c})$, for every individual constant c
4. $[\![\mathsf{p}]\!]_s(I) = I(\mathsf{p})$, for every predicate constant p
5. $[\![\mathsf{V}]\!]_s(I) = s(\mathsf{V})$, for every argument variable V
6. $[\![(\mathsf{f}\ \mathsf{E}_1 \cdots \mathsf{E}_n)]\!]_s(I) = I(\mathsf{f})\ [\![\mathsf{E}_1]\!]_s(I) \cdots [\![\mathsf{E}_n]\!]_s(I)$, for every n-ary function symbol f
7. $[\![(\mathsf{E}_1\mathsf{E}_2)]\!]_s(I) = [\![\mathsf{E}_1]\!]_s(I)([\![\mathsf{E}_2]\!]_s(I))$
8. $[\![(\lambda\mathsf{V}.\mathsf{E})]\!]_s(I) = \lambda d.[\![\mathsf{E}]\!]_{s[\mathsf{V}/d]}(I)$, where d ranges over $[\![type(\mathsf{V})]\!]_D$
9. $[\![(\mathsf{E}_1 \bigvee_\pi \mathsf{E}_2)]\!]_s(I) = \bigvee_\pi\{[\![\mathsf{E}_1]\!]_s(I), [\![\mathsf{E}_2]\!]_s(I)\}$, where $\bigvee_\pi$ is the least upper bound function on $[\![\pi]\!]_D$
10. $[\![(\mathsf{E}_1 \bigwedge_\pi \mathsf{E}_2)]\!]_s(I) = \bigwedge_\pi\{[\![\mathsf{E}_1]\!]_s(I), [\![\mathsf{E}_2]\!]_s(I)\}$, where $\bigwedge_\pi$ is the greatest lower bound function on $[\![\pi]\!]_D$
11. $[\![(\sim\mathsf{E})]\!]_s(I) = \begin{cases} T_{\alpha+1} & \text{if } [\![\mathsf{E}]\!]_s(I) = F_\alpha \\ F_{\alpha+1} & \text{if } [\![\mathsf{E}]\!]_s(I) = T_\alpha \\ 0 & \text{if } [\![\mathsf{E}]\!]_s(I) = 0 \end{cases}$
12. $[\![(\mathsf{E}_1 \approx \mathsf{E}_2)]\!]_s(I) = \begin{cases} T_0, & \text{if } [\![\mathsf{E}_1]\!]_s(I) = [\![\mathsf{E}_2]\!]_s(I) \\ F_0, & \text{otherwise} \end{cases}$
13. $[\![(\exists\mathsf{V}\ \mathsf{E})]\!]_s(I) = \bigvee_{d \in [\![type(\mathsf{V})]\!]_D} [\![\mathsf{E}]\!]_{s[\mathsf{V}/d]}(I)$

For closed expressions E we will often write $[\![\mathsf{E}]\!](I)$ instead of $[\![\mathsf{E}]\!]_s(I)$ (since, in this case, the meaning of E is independent of s).

Lemma 3
Let $\mathsf{E} : \rho$ be an expression and let D be a nonempty set. Moreover, let s be a state over D and let I be an interpretation over D. Then, $[\![\mathsf{E}]\!]_s(I) \in [\![\rho]\!]_D$.

Definition 14
Let P be a program and let M be an interpretation over a nonempty set D. Then M will be called a *model* of P iff for all clauses $\mathsf{p} \leftarrow_\pi \mathsf{E}$ of P, it holds $[\![\mathsf{E}]\!](M) \leqslant_\pi M(\mathsf{p})$, where $M(\mathsf{p}) \in [\![\pi]\!]_D$.

6 Minimum Herbrand Model Semantics for $\mathscr{H}$

In this section we demonstrate that every program of $\mathscr{H}$ has a unique minimum Herbrand model which is the greatest lower bound of all the Herbrand models of the program, and also the least fixed point of the immediate consequence operator of the program. We start with the relevant definitions.

Definition 15
Let P be a program. The Herbrand universe U_P of P is the set of all terms that can be formed out of the individual constants[3] and the function symbols of P.

Definition 16
A Herbrand interpretation I of a program P is an interpretation such that:

1. the domain of I is the Herbrand universe U_P of P;
2. for every individual constant c of P, $I(\mathsf{c}) = \mathsf{c}$;
3. for every predicate constant $\mathsf{p} : \pi$ of P, $I(\mathsf{p}) \in [\![\pi]\!]_{U_\mathsf{P}}$;
4. for every n-ary function symbol f of P and for all $\mathsf{t}_1, \ldots, \mathsf{t}_n \in U_\mathsf{P}$, $I(\mathsf{f})\,\mathsf{t}_1 \cdots \mathsf{t}_n = \mathsf{f}\,\mathsf{t}_1 \cdots \mathsf{t}_n$.

A Herbrand state of a program P is a state whose underlying domain is U_P. We denote the set of Herbrand interpretations of a program P by $\mathscr{I}_\mathsf{P}$.

Definition 17
A Herbrand model of a program P is a Herbrand interpretation that is a model of P.

Definition 18
Let P be a program. We define the following partial order on $\mathscr{I}_\mathsf{P}$: for all $I, J \in \mathscr{I}_\mathsf{P}$, $I \leqslant_{\mathscr{I}_\mathsf{P}} J$ iff for every π and for every predicate constant $\mathsf{p} : \pi$ of P, $I(\mathsf{p}) \leqslant_\pi J(\mathsf{p})$.

Definition 19
Let P be a program. We define the following preorder on $\mathscr{I}_\mathsf{P}$ for all $\alpha < \kappa$: for all $I, J \in \mathscr{I}_\mathsf{P}$, $I \sqsubseteq_\alpha J$ iff for every π and for every predicate constant $\mathsf{p} : \pi$ of P, $I(\mathsf{p}) \sqsubseteq_\alpha J(\mathsf{p})$.

The following two lemmas play a main role in establishing the two central theorems.

Lemma 4
Let P be a program. Then, $\mathscr{I}_\mathsf{P}$ is a complete lattice and a basic model.

[3] As usual, if P has no constants, we assume the existence of an arbitrary one.

Lemma 5 (*α-Monotonicity of Semantics*)
Let P be a program and let $\mathsf{E} : \pi$ be an expression. Let I, J be Herbrand interpretations and s be a Herbrand state of P. For all $\alpha < \kappa$, if $I \sqsubseteq_\alpha J$ then $[\![\mathsf{E}]\!]_s(I) \sqsubseteq_\alpha [\![\mathsf{E}]\!]_s(J)$.

Since by Lemma 4 the set $\mathscr{I}_\mathsf{P}$ is a basic model (and thus by Lemma 1 is a complete lattice with respect to $\sqsubseteq$), every $\mathscr{M} \subseteq \mathscr{I}_\mathsf{P}$ has a greatest lower bound $\bigsqcap \mathscr{M}$ with respect to $\sqsubseteq$. We have the following theorem which generalizes the familiar model intersection theorem for definite first-order logic programs (Lloyd 1987), the model intersection theorem for normal first-order logic programs (Rondogiannis and Wadge 2005, Theorem 8.6) and the model intersection theorem for definite higher-order logic programs (Charalambidis *et al.* 2013, Theorem 6.8).

Theorem 2 (*Model Intersection Theorem*)
Let P be a program and $\mathscr{M}$ be a nonempty set of Herbrand models of P. Then, $\bigsqcap \mathscr{M}$ is also a Herbrand model of P.

Definition 20
Let P be a program. The mapping $T_\mathsf{P} : \mathscr{I}_\mathsf{P} \to \mathscr{I}_\mathsf{P}$ is defined for every $\mathsf{p} : \pi$ and for every $I \in \mathscr{I}_\mathsf{P}$ as $T_\mathsf{P}(I)(\mathsf{p}) = \bigvee\{[\![\mathsf{E}]\!](I) : (\mathsf{p} \leftarrow_\pi \mathsf{E}) \in \mathsf{P}\}$. The mapping T_P will be called the *immediate consequence operator* for P.

The following two lemmas are crucial in establishing the least fixed point theorem.

Lemma 6
Let P be a program. For every predicate constant $\mathsf{p} : \pi$ in P and $I \in \mathscr{I}_\mathsf{P}$, $T_P(I)(\mathsf{p}) \in [\![\pi]\!]_{U_\mathsf{P}}$.

Lemma 7
Let P be a program. Then, T_P is α-monotonic for all $\alpha < \kappa$.

Theorem 3 (*Least Fixed Point Theorem*)
Let P be a program and let $\mathscr{M}$ be the set of all its Herbrand models. Then, T_P has a least fixed point M_P. Moreover, $M_\mathsf{P} = \bigsqcap \mathscr{M}$.

The construction of the least fixed point in the above theorem is similar to the one given for (potentially infinite) propositional programs in (Rondogiannis and Wadge 2005, Section 6). Due to space limitations, we provide a short outline of this procedure. In order to calculate the least fixed point, we start with an interpretation, say I_0, which for every predicate constant p of type $\rho_1 \to \cdots \rho_n \to o$, and for all $d_1 \in [\![\rho_1]\!]_{U_\mathsf{P}}, \ldots, d_n \in [\![\rho_n]\!]_{U_\mathsf{P}}$, $I_0(\mathsf{p})\, d_1 \cdots d_n = F_0$. We start iterating T_P on this interpretation until we get to a point where the additional iterations do not affect the F_0 and T_0 values. At this point, we reset all the remaining values (regarding predicate constants and arguments that have not stabilized) to F_1, getting an interpretation I_1. We start iterating T_P on I_1, until we get to a point where the additional iterations do not affect the F_1 and T_1 values. We repeat this process for higher ordinals. In particular, when we get to a limit ordinal, say α, we reset all the values that have not stabilized to a truth value of order less than α, to F_α. The whole process is repeated for κ times. If the value of certain predicate constants applied to certain arguments has not stabilized after the κ iterations, we assign to them the intermediate value 0. The resulting interpretation is the least fixed point M_P.

7 Resolving a Semantic Paradox of Higher-Order Logic Programming

One deficiency of extensional higher-order logic programming is the inability to define rules (or facts) that have predicate constants in their heads. The reason of this restriction is a semantic one and will be explained shortly. However, not all programs that use predicate constants in the heads of clauses are problematic. For example, the program

```
computer_scientist(john).
good_profession(computer_scientist).
```

has a clear declarative reading: the denotation of the `computer_scientist` predicate is the relation {`john`}, while the denotation of `good_profession` is the relation {{`john`}}.

In (Wadge 1991), W. W. Wadge argued that allowing rules to have predicate constants in their heads, creates tricky semantic problems to. Wadge gave a simple example (duplicated below) that revealed these problems; the example has since been used in other studies of higher-order logic programming (such as for example in (Bezem 2001)). We present the example in almost identical phrasing as it initially appeared.

Example 2
Consider the program:

```
p(a).
q(a).
phi(p).
q(b):-phi(q).
```

One candidate for minimum Herbrand model is the one in which `p` and `q` are true only of `a`, and `phi` is true only of `p`. However, this means that `p` and `q` have the same extension, and so themselves are equal. But since `p` and `q` are equal, and `phi` holds for `p`, it must also hold for `q`. The fourth rule forces us to add `q(b)`, so that the model becomes {`p(a)`,`phi(p)`,`q(a)`,`q(b)`} (in ad hoc notation). But this is problematic because `p` and `q` are no longer equal and `q(b)` has lost its justification.

Problems such as the above led Wadge to disallow such clauses from the syntax of the language proposed in (Wadge 1991). Similarly, the higher-order language introduced in (Charalambidis *et al.* 2013) also disallows this kind of clauses.

However, under the semantics presented in this paper, we can now assign a proper meaning to programs such as the above. Actually, higher order facts such as `phi(p).` above, can be seen as syntactic sugar in our fragment. A fact of this form simply states that `phi` is true of a relation if this relation *is equal* to `p`. This can simply be written as:

```
phi(P):-equal(P,p).
```

where `equal` is a higher-order equality relation that can easily be axiomatized in $\mathcal{H}$ using the `subset` predicate (see Example 1):

$$\texttt{equal} \leftarrow \lambda\texttt{P}.\lambda\texttt{Q}.(\texttt{subset P Q}) \wedge (\texttt{subset Q P}).$$

One can compute the minimum model of the resulting program using the techniques presented in this paper. The paradox of Example 2 is no longer valid since in the minimum infinite-valued model the atom `q(b)` has value 0. Intuitively, this means that it is not possible to decide whether `q(b)` should be true or false.

The above discussion leads to an easy way of handling rules with predicate constants in their heads. The predicate constants are replaced with predicate variables and higher-order equality atoms are added in the bodies of clauses. Then, appropriate clauses defining the `equal` predicates for all necessary types, are added to the program. The infinite valued semantics of the resulting program is taken as the meaning of the initial program.

8 Future Work

We have presented the first, to our knowledge, formal semantics for negation in extensional higher-order logic programming. The results we have obtained generalize the semantics of classical logic programming to the higher order setting. We believe that the most interesting direction for future work is the investigation of implementation techniques for (fragments of) $\mathcal{H}$, based on the semantics introduced in this paper. One possible option would be to examine the implementation of a higher order extension of Datalog with negation. We are currently examining these possibilities.

References

Bezem, M. 2001. An improved extensionality criterion for higher-order logic programs. In *Proceedings of the 15th International Workshop on Computer Science Logic (CSL)*. Springer-Verlag, London, UK, 203–216.

Charalambidis, A., Handjopoulos, K., Rondogiannis, P., and Wadge, W. W. 2010. Extensional higher-order logic programming. In *JELIA*, T. Janhunen and I. Niemelä, Eds. Lecture Notes in Computer Science, vol. 6341. Springer, 91–103.

Charalambidis, A., Handjopoulos, K., Rondogiannis, P., and Wadge, W. W. 2013. Extensional higher-order logic programming. *ACM Transactions on Computational Logic 14,* 3, 21:1–21:40.

Ésik, Z. and Rondogiannis, P. 2013. A fixed point theorem for non-monotonic functions. In *Proceedings of 13th Panhellenic Logic Symposium, Athens, Greece.*

Ésik, Z. and Rondogiannis, P. 2014. A fixed point theorem for non-monotonic functions. *CoRR abs/1402.0299.*

Lloyd, J. W. 1987. *Foundations of Logic Programming*. Springer Verlag.

Pearce, D. 1996. A new logical characterisation of stable models and answer sets. In *NMELP*, J. Dix, L. M. Pereira, and T. C. Przymusinski, Eds. Lecture Notes in Computer Science, vol. 1216. Springer, 57–70.

Przymusinski, T. C. 1989. Every logic program has a natural stratification and an iterated least fixed point model. In *PODS*, A. Silberschatz, Ed. ACM Press, 11–21.

Rondogiannis, P. and Wadge, W. W. 2005. Minimum model semantics for logic programs with negation-as-failure. *ACM Transactions on Computational Logic 6,* 2, 441–467.

van Gelder, A., Ross, K. A., and Schlipf, J. S. 1991. The well-founded semantics for general logic programs. *J. ACM 38,* 3, 620–650.

Wadge, W. W. 1991. Higher-order Horn logic programming. In *ISLP*. 289–303.