Equivalence of two Fixed-Point Semantics for Definitional Higher-Order Logic Programs*

Angelos Charalambidis
University of Athens
Athens, Greece
a.charalambidis@di.uoa.gr

Panos Rondogiannis
University of Athens
Athens, Greece
prondo@di.uoa.gr

Ioanna Symeonidou
University of Athens
Athens, Greece
i.symeonidou@di.uoa.gr

Two distinct research approaches have been proposed for assigning a purely extensional semantics to higher-order logic programming. The former approach uses classical domain-theoretic tools while the latter builds on a fixed-point construction defined on a syntactic instantiation of the source program. The relationships between these two approaches had not been investigated until now. In this paper we demonstrate that for a very broad class of programs, namely the class of *definitional programs* introduced by W. W. Wadge, the two approaches coincide (with respect to ground atoms that involve symbols of the program). On the other hand, we argue that if existential higher-order variables are allowed to appear in the bodies of program rules, the two approaches are in general different. The results of the paper contribute to a better understanding of the semantics of higher-order logic programming.

1 Introduction

Extensional higher-order logic programming has been proposed [10, 1, 2, 7, 5, 4] as a promising generalization of classical logic programming. The key idea behind this paradigm is that the predicates defined in a program essentially denote sets and therefore one can use standard extensional set theory in order to understand their meaning and reason about them. The main difference between the extensional and the more traditional *intensional* approaches to higher-order logic programming [9, 6] is that the latter approaches have a much richer syntax and expressive capabilities but a non-extensional semantics.

Actually, despite the fact that only very few articles have been written regarding extensionality in higher-order logic programming, two main semantic approaches can be identified. The work described in [10, 7, 5, 4] uses classical domain-theoretic tools in order to capture the meaning of higher-order logic programs. On the other hand, the work presented in [1, 2] builds on a fixed-point construction defined on a syntactic instantiation of the source program in order to achieve an extensional semantics. Until now, the relationships between the above two approaches had not yet been investigated.

In this paper we demonstrate that for a very broad class of programs, namely the class of *definitional programs* introduced by W. W. Wadge [10], the two approaches coincide. Intuitively, this means that for any given definitional program, the sets of true ground atoms of the program are identical under the two different semantic approaches. This result is interesting since it suggests that definitional programs are of fundamental importance for the further study of extensional higher-order logic programming. On the other hand, we argue that if we try to slightly extend the source language, the two approaches give different results in general. Overall, the results of the paper contribute to a better understanding of the

*This research was supported by the project "Handling Uncertainty in Data Intensive Applications", co-financed by the European Union (European Social Fund) and Greek national funds, through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Program: THALES, Investing in knowledge society through the European Social Fund.

R. Matthes, M. Mio (Eds.): Fixed Points
in Computer Science 2015 (FICS 2015)
EPTCS 191, 2015, pp. 18–32, doi:10.4204/EPTCS.191.4

semantics of higher-order logic programming and pave the road for designing a realistic extensional higher-order logic programming language.

The rest of the paper is organized as follows. Section 2 briefly introduces extensional higher-order logic programming and presents in an intuitive way the two existing approaches for assigning meaning to programs of this paradigm. Section 3 contains background material, namely the syntax of definitional programs and the formal details behind the two aforementioned semantic approaches. Section 4 demonstrates the equivalence of the two semantics for definitional programs. Finally, Section 5 concludes the paper with discussion regarding non-definitional programs and with pointers to future work.

2 Intuitive Overview of the two Extensional Approaches

In this section we introduce extensional higher-order logic programming and present the two existing approaches for assigning meaning to programs of this paradigm. Since these two proposals were initially introduced by W. W. Wadge and M. Bezem respectively, we will refer to them as *Wadge's semantics* and *Bezem's semantics* respectively. The key idea behind both approaches is that in order to achieve an extensional semantics, one has to consider a fragment of higher-order logic programming that has a restricted syntax.

2.1 Extensional Higher-Order Logic Programming

The main differences between extensional and intensional higher-order logic programming can be easily understood through two simple examples (borrowed from [5]). Due to space limitations, we avoid a more extensive discussion of this issue; the interested reader can consult [5].

Example 1. Suppose we have a database of professions, both of their membership and their status. We might have rules such as:

```
engineer(tom).
engineer(sally).
programmer(harry).
```

with `engineer` and `programmer` used as predicates. In intensional higher-order logic programming we could also have rules in which these are arguments, eg:

```
profession(engineer).
profession(programmer).
```

Now suppose `tom` and `sally` are also avid users of Twitter. We could have rules:

```
tweeter(tom).
tweeter(sally).
```

The predicates `tweeter` and `engineer` are equal as sets (since they are true for the same objects, namely `tom` and `sally`). If we attempted to understand the above program from an extensional point of view, then we would have to accept that `profession(tweeter)` must also hold (since `tweeter` and `engineer` are indistinguishable as sets). It is clear that the extensional interpretation in this case is completely unnatural. The program can however be understood intensionally: the predicate `profession` is true of the *name* `engineer` (which is different than the name `tweeter`). □

On the other hand, there are cases where predicates can be understood extensionally:

Example 2. Consider a program that consists only of the following rule:

```
p(Q):-Q(0),Q(1).
```

In an extensional language, predicate `p` above can be intuitively understood in purely set-theoretic terms: `p` is the set of all those sets that contain both `0` and `1`.

It should be noted that the above program is also a syntactically acceptable program of the existing intensional logic programming languages. The difference is that in an extensional language the above program has a purely set-theoretic semantics. □

From the above examples it can be understood that extensional higher-order logic programming sacrifices some of the rich syntax of intensional higher-order logic programming in order to achieve semantic clarity.

2.2 Wadge's Semantics

The first proposal for an extensional semantics for higher-order logic programming was given in [10] (and later refined and extended in [7, 5, 4]). The basic idea behind Wadge's approach is that if we consider a properly restricted higher-order logic programming language, then we can use standard ideas from denotational semantics in order to assign an extensional meaning to programs. The basic syntactic assumptions introduced by Wadge in [10] are the following:

- In the head of every rule in a program, each argument of predicate type must be a variable; all such variables must be distinct.
- The only variables of predicate type that can appear in the body of a rule, are variables that appear in its head.

Programs that satisfy the above restrictions are named *definitional* in [10].

Example 3. The program[1]:

```
p(a).
q(b).
r(P,Q):-P(a),Q(b).
```

is definitional because the arguments of predicate type in the head of the rule for `r` are distinct variables. Moreover, the only predicate variables that appear in the body of the same rule, are the variables in its head (namely `P` and `Q`). □

Example 4. The program:

```
q(a).
r(q).
```

is not definitional because the predicate constant `q` appears as an argument in the second clause. For a similar reason, the program in Example 1 is not definitional. The program:

```
p(Q,Q):-Q(a).
```

is also not definitional because the predicate variable `Q` is used twice in the head of the above rule. Finally, the program:

```
p(a):-Q(a).
```

is not definitional because the predicate variable `Q` that appears in the body of the above rule, does not appear in the head of the rule. □

[1] For simplicity reasons, the syntax that we use in our example programs is Prolog-like. The syntax that we adopt in the next section is slightly different and more convenient for the theoretical developments that follow.

As it is argued in [10], if a program satisfies the above two syntactic restrictions, then it has a *unique minimum model* (this notion will be precisely defined in Section 3). Consider again the program of Example 3. In the minimum model of this program, the meaning of predicate p is the relation $\{\mathtt{a}\}$ and the meaning of predicate q is the relation $\{\mathtt{b}\}$. On the other hand, the meaning of predicate r in the minimum model is a relation that contains the pairs $(\{\mathtt{a}\},\{\mathtt{b}\})$, $(\{\mathtt{a},\mathtt{b}\},\{\mathtt{b}\})$, $(\{\mathtt{a}\},\{\mathtt{a},\mathtt{b}\})$ and $(\{\mathtt{a},\mathtt{b}\},\{\mathtt{a},\mathtt{b}\})$. As remarked by W. W. Wadge (and formally demonstrated in [7, 5]), the minimum model of every definitional program is monotonic and continuous[2]. Intuitively, monotonicity means that if in the minimum model the meaning of a predicate is true of a relation, then it is also true of every superset of this relation. For example, we see that since the meaning of r is true of $(\{\mathtt{a}\},\{\mathtt{b}\})$, then it is also true of $(\{\mathtt{a},\mathtt{b}\},\{\mathtt{b}\})$ (because $\{\mathtt{a},\mathtt{b}\}$ is a superset of $\{\mathtt{a}\}$).

The minimum model of a given definitional program can be constructed as the least fixed-point of an operator that is associated with the program, called the *immediate consequence operator* of the program. As it is demonstrated in [10, 7], the immediate consequence operator is monotonic, and this guarantees the existence of the least fixed-point which is constructed by a bottom-up iterative procedure (more formal details will be given in the next section).

Example 5. Consider the definitional program:

```
q(a).
q(b).
p(Q):-Q(a).
id(R)(X):-R(X).
```

In the minimum model of the above program, the meaning of q is the relation $\{\mathtt{a},\mathtt{b}\}$. The meaning of p is the set of all relations that contain (at least) a; more formally, it is the relation $\{r \mid \mathtt{a} \in r\}$. The meaning of id is the set of all pairs (r,d) such that d belongs to r; more formally, it is the relation $\{(r,d) \mid d \in r\}$. □

Notice that in the construction of the minimum model, all predicates are initially assigned the empty relation. The rules of the program are then used in order to improve the meaning assigned to each predicate symbol. More specifically, at each step of the fixed-point computation, the meaning of each predicate symbol either stabilizes or it becomes richer than the previous step.

Example 6. Consider again the definitional program of the previous example. In the iterative construction of the minimum model, all predicates are initially assigned the empty relation (of the corresponding type). After the first step of the construction, the meaning assigned to predicate q is the relation $\{\mathtt{a},\mathtt{b}\}$ due to the first two facts of the program. At this same step, the meaning of p becomes the relation $\{r \mid \mathtt{a} \in r\}$. Also, the meaning of id becomes equal to the relation $\{(r,d) \mid d \in r\}$. Additional iterations will not alter the relations we have obtained at the first step; in other words, we have reached the fixed-point of the bottom-up computation. □

In the above example, we obtained the meaning of the program in just one step. If the source program contained recursive definitions, convergence to the least fixed-point would in general require more steps.

2.3 Bezem's Semantics

In [1, 2], M. Bezem proposed an alternative extensional semantics for higher-order logic programs. Again, the syntax of the source language has to be appropriately restricted. Actually, the class of programs adopted in [1, 2] is a proper superset of the class of definitional programs. In particular, Bezem proposes the class of *hoapata programs* which extend definitional programs:

[2] The notion of continuity will not play any role in the remaining part of this paper.

- A predicate variable that appears in the body of a rule, need not necessarily appear in the head of that rule.
- The head of a rule can be an atom that starts with a predicate variable.

Example 7. All definitional programs of the previous subsection are also hoapata. The following non-definitional program of Example 4 is hoapata:

```
p(a):-Q(a).
```

Intuitively, the above program states that `p` is true of `a` if there exists a predicate that is defined in the program that is true of `a`. We will use this program in our discussion at the end of the paper.

The following program is also hoapata (but not definitional):

```
P(a,b).
```

Intuitively, the above program states that every binary relation is true of the pair $(\texttt{a},\texttt{b})$. □

Given a hoapata program, the starting idea behind Bezem's approach is to take its "ground instantiation" in which we replace variables with well-typed terms of the Herbrand Universe of the program (ie., terms that can be created using only predicate and individual constants that appear in the program). For example, given the program:

```
q(a).
q(b).
p(Q):-Q(a).
id(R)(X):-R(X).
```

the ground instantiation is the following infinite "program":

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

One can now treat the new program as an infinite propositional one (ie., each ground atom can be seen as a propositional one). This implies that we can use the standard least fixed-point construction of classical logic programming (see for example [8]) in order to compute the set of atoms that should be taken as "true". In our example, the least fixed-point will contain atoms such as `q(a)`, `q(b)`, `p(q)`, `id(q)(a)`, `p(id(q))`, and so on.

A main contribution of Bezem's work was that he established that the least fixed-point of the ground instantiation of every hoapata program is *extensional*. This notion can intuitively be explained as follows. It is obvious in the above example that the relations `q` and `id(q)` are equal (they are both true of only the constant `a`, and therefore they both correspond to the relation $\{\texttt{a}\}$). Therefore, we would expect that (for example) if `p(q)` is true then `p(id(q))` is also true because `q` and `id(q)` should be considered as interchangeable. This property of "interchangeability" is formally defined in [1, 2] and it is demonstrated that it holds in the least fixed-point of the immediate consequence operator of the ground instance of every hoapata program.

2.4 The Differences Between the two Approaches

It is not hard to see that the two semantic approaches outlined in the previous subsections, have some important differences. First, they operate on different source languages. Therefore, in order to compare them we have to restrict Bezem's approach to the class of definitional programs[3].

The main difference however between the two approaches is the way that the least fixed-point of the immediate consequence operator is constructed in each case. In Wadge's semantics the construction starts by initially assigning to every predicate constant the empty relation; these relations are then improved at each step until they converge to their final meaning. In other words, Wadge's semantics *manipulates relations*. On the other hand, Bezem's semantics works with the ground instantiation of the source program and, at first sight, it appears to have a more syntactic flavor. In our running example, Wadge's approach converges in a single step while Bezem's approach takes an infinite number of steps in order to converge. However, one can easily verify that the ground atoms that belong to the least fixed-point under Bezem's semantics, are also true in the minimum model under Wadge's semantics. This poses the question whether under both approaches, the sets of ground atoms that are true, are identical. This is the question that we answer positively in the rest of the paper.

3 Definitional Programs and their Semantics

In this section we define the source language $\mathscr{H}$ of definitional higher-order logic programs. Moreover, we present in a formal way the two different extensional semantics that have been proposed for such programs, namely Wadge's and Bezem's semantics respectively.

3.1 Syntax

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 function symbols), predicate (assigned to predicate symbols) and argument (assigned to parameters of predicates).

Definition 1. A type can either be functional, argument, or predicate, denoted by σ, ρ and π 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, argument or predicate 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 $\iota^n \to \iota$, $n \geq 1$. Moreover, it can be easily seen that every predicate type π can be written in the form $\rho_1 \to \cdots \to \rho_n \to o$, $n \geq 0$ (for $n = 0$ we assume that $\pi = o$).

We proceed by defining the syntax of $\mathscr{H}$:

Definition 2. 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 $\mathsf{P}, \mathsf{Q}, \mathsf{R}, \ldots$).

[3] Actually, we could alternatively extend Wadge's approach to a broader class of programs. Such an extension has already been performed in [5], and we will discuss its repercussions in the concluding section.

2. Individual variables of type ι (denoted by capital letters such as $\mathsf{X}, \mathsf{Y}, \mathsf{Z}, \ldots$).
3. Predicate constants of every predicate type π (denoted by lowercase letters such as $\mathsf{p}, \mathsf{q}, \mathsf{r}, \ldots$).
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 logical conjunction constant $\wedge$, the inverse implication constant $\leftarrow$, the left and right parentheses, and the equality constant $\approx$ for comparing terms of type ι.

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 3. The set of *terms* of the higher-order language $\mathscr{H}$ is defined as follows:

- Every predicate variable (respectively predicate constant) of type π is a term of type π; every individual variable (respectively individual constant) of type ι is a term of type ι;
- if f is an n-ary function symbol and $\mathsf{E}_1, \ldots, \mathsf{E}_n$ are terms of type ι then $(\mathsf{f}\ \mathsf{E}_1 \cdots \mathsf{E}_n)$ is a term of type ι;
- if E_1 is a term of type $\rho \to \pi$ and E_2 a term of type ρ then $(\mathsf{E}_1\ \mathsf{E}_2)$ is a term of type π.

Definition 4. The set of *expressions* of the higher-order language $\mathscr{H}$ is defined as follows:

- A term of type ρ is an expression of type ρ;
- if E_1 and E_2 are terms of type ι, then $(\mathsf{E}_1 \approx \mathsf{E}_2)$ is an expression of type o.

We write $\mathit{vars}(\mathsf{E})$ to denote the set of all the variables in E. Expressions (respectively terms) that have no variables will often be referred to as *ground expressions* (respectively *ground terms*). Expressions of type o will often be referred to as *atoms*. We will omit parentheses when no confusion arises. To denote that an expression E has type ρ we will often write $\mathsf{E} : \rho$.

Definition 5. A *clause* is a formula $\mathsf{p}\,\mathsf{V}_1 \cdots \mathsf{V}_n \leftarrow \mathsf{E}_1 \wedge \cdots \wedge \mathsf{E}_m$, where p is a predicate constant, $\mathsf{p}\,\mathsf{V}_1 \cdots \mathsf{V}_n$ is a term of type o and $\mathsf{E}_1, \ldots, \mathsf{E}_m$ are expressions of type o. The term $\mathsf{p}\,\mathsf{V}_1 \cdots \mathsf{V}_n$ is called the *head* of the clause, the variables $\mathsf{V}_1, \ldots, \mathsf{V}_n$ are the *formal parameters* of the clause and the conjunction $\mathsf{E}_1 \wedge \cdots \wedge \mathsf{E}_m$ is its *body*. A *definitional clause* is a clause that additionally satisfies the following two restrictions:

1. All the formal parameters are distinct variables (ie., for all i, j such that $1 \leq i, j \leq n$, $\mathsf{V}_i \neq \mathsf{V}_j$).
2. The only variables that can appear in the body of the clause are its formal parameters and possibly some additional individual variables (namely variables of type ι).

A *program* P is a set of definitional program clauses.

In the rest of the paper, when we refer to "clauses" we will mean definitional ones. For simplicity, we will follow the usual logic programming convention and we will write $\mathsf{p}\,\mathsf{V}_1 \cdots \mathsf{V}_n \leftarrow \mathsf{E}_1, \ldots, \mathsf{E}_m$ instead of $\mathsf{p}\,\mathsf{V}_1 \cdots \mathsf{V}_n \leftarrow \mathsf{E}_1 \wedge \cdots \wedge \mathsf{E}_m$.

Our syntax differs slightly from the Prolog-like syntax that we have used in Section 2. However, one can easily verify that we can transform every program from the former syntax to the latter.

Definition 6. For a program P, we define the Herbrand universe for every argument type ρ, denoted by $U_{\mathsf{P},\rho}$ to be the set of all ground terms of type ρ, that can be formed out of the individual constants, function symbols and predicate constants in the program.

In the following, we will often talk about the "ground instantiation of a program". This notion is formally defined below.

Definition 7. A *ground substitution* θ is a finite set of the form $\{\mathsf{V}_1/\mathsf{E}_1,\ldots,\mathsf{V}_n/\mathsf{E}_n\}$ where the V_i's are different argument variables and each E_i is a ground term having the same type as V_i. We write $dom(\theta)=\{\mathsf{V}_1,\ldots,\mathsf{V}_n\}$ to denote the domain of θ.

We can now define the application of a substitution to an expression.

Definition 8. Let θ be a substitution and E be an expression. Then, $\mathsf{E}\theta$ is an expression obtained from E as follows:

- $\mathsf{E}\theta=\mathsf{E}$ if E is a predicate or individual constant;
- $\mathsf{V}\theta=\theta(\mathsf{V})$ if $\mathsf{V}\in dom(\theta)$; otherwise, $\mathsf{V}\theta=\mathsf{V}$;
- $(\mathsf{f}\ \mathsf{E}_1\cdots\mathsf{E}_n)\theta=(\mathsf{f}\ \mathsf{E}_1\theta\cdots\mathsf{E}_n\theta)$;
- $(\mathsf{E}_1\ \mathsf{E}_2)\theta=(\mathsf{E}_1\theta\ \mathsf{E}_2\theta)$;
- $(\mathsf{E}_1\approx\mathsf{E}_2)\theta=(\mathsf{E}_1\theta\approx\mathsf{E}_2\theta)$.

Definition 9. Let E be an expression and θ be a ground substitution such that $vars(\mathsf{E})\subseteq dom(\theta)$. Then, the ground expression $\mathsf{E}\theta$ is called a *ground instantiation* of E. A *ground instantiation of a clause* $\mathsf{p}\ \mathsf{V}_1\cdots\mathsf{V}_n\leftarrow\mathsf{E}_1,\ldots,\mathsf{E}_m$ with respect to a ground substitution θ is the formula $(\mathsf{p}\ \mathsf{V}_1\cdots\mathsf{V}_n)\theta\leftarrow\mathsf{E}_1\theta,\ldots,\mathsf{E}_m\theta$. The *ground instantiation of a program* P is the (possibly infinite) set that contains all the ground instantiations of the clauses of P with respect to all possible ground substitutions.

3.2 Wadge's Semantics

The key idea behind Wadge's semantics is (intuitively) to assign to program predicates monotonic relations. In the following, given posets A and B, we write $[A\stackrel{m}{\rightarrow}B]$ to denote the set of all monotonic relations from A to B.

Before specifying the semantics of expressions of $\mathscr{H}$ we need to provide the set-theoretic meaning of the types of expressions of $\mathscr{H}$ with respect to an underlying domain. It is customary in logic programming to take the underlying domain to be the Herbrand universe $U_{\mathsf{P},\iota}$. In the following definition we define simultaneously and recursively two things: the semantics $[\![\tau]\!]$ of a type τ and a corresponding partial order $\sqsubseteq_\tau$ on the elements of $[\![\tau]\!]$. We adopt the usual ordering of the truth values *false* and *true*, i.e. *false* $\leq$ *false*, *true* $\leq$ *true* and *false* $\leq$ *true*.

Definition 10. Let P be a program. Then,

- $[\![\iota]\!]=U_{\mathsf{P},\iota}$ and $\sqsubseteq_\iota$ is the trivial partial order that relates every element to itself;
- $[\![\iota^n\rightarrow\iota]\!]=U^n_{\mathsf{P},\iota}\rightarrow U_{\mathsf{P},\iota}$. A partial order for this case is not needed;
- $[\![o]\!]=\{\mathit{false},\mathit{true}\}$ and $\sqsubseteq_o$ is the partial order $\leq$ on truth values;
- $[\![\rho\rightarrow\pi]\!]=[[\![\rho]\!]\stackrel{m}{\rightarrow}[\![\pi]\!]]$ and $\sqsubseteq_{\rho\rightarrow\pi}$ is the partial order defined as follows: for all $f,g\in[\![\rho\rightarrow\pi]\!]$, $f\sqsubseteq_{\rho\rightarrow\pi}g$ iff $f(d)\sqsubseteq_\pi g(d)$ for all $d\in[\![\rho]\!]$.

We now proceed to define Herbrand interpretations and states.

Definition 11. A Herbrand interpretation I of a program P is an interpretation such that:

1. for every individual constant c that appears in P, $I(\mathsf{c})=\mathsf{c}$;
2. for every predicate constant $\mathsf{p}:\pi$ that appears in P, $I(\mathsf{p})\in[\![\pi]\!]$;
3. for every n-ary function symbol f that appears in P and for all $\mathsf{t}_1,\ldots\mathsf{t}_n\in U_{\mathsf{P},\iota}$, $I(\mathsf{f})\ \mathsf{t}_1\ \cdots\mathsf{t}_n=\mathsf{f}\ \mathsf{t}_1\ \cdots\mathsf{t}_n$.

Definition 12. A Herbrand state s of a program P is a function that assigns to each argument variable V of type ρ, an element $s(\mathsf{V}) \in [\![\rho]\!]$.

In the following, $s[\mathsf{V}/d]$ is used to denote a state that is identical to s the only difference being that the new state assigns to V the value d.

Definition 13. Let P be a program, I be a Herbrand interpretation of P and s be a Herbrand state. Then, the semantics of the expressions of P is defined as follows:

1. $[\![\mathsf{V}]\!]_s(I) = s(\mathsf{V})$ if V is a variable;
2. $[\![\mathsf{c}]\!]_s(I) = I(\mathsf{c})$ if c is an individual constant;
3. $[\![\mathsf{p}]\!]_s(I) = I(\mathsf{p})$ if p is a predicate constant;
4. $[\![(\mathsf{f}\ \mathsf{E}_1 \cdots \mathsf{E}_n)]\!]_s(I) = I(\mathsf{f})\ [\![\mathsf{E}_1]\!]_s(I) \cdots [\![\mathsf{E}_n]\!]_s(I)$;
5. $[\![(\mathsf{E}_1\ \mathsf{E}_2)]\!]_s(I) = [\![\mathsf{E}_1]\!]_s(I)\ [\![\mathsf{E}_2]\!]_s(I)$;
6. $[\![(\mathsf{E}_1 \approx \mathsf{E}_2)]\!]_s(I) = \mathit{true}$ if $[\![\mathsf{E}_1]\!]_s(I) = [\![\mathsf{E}_2]\!]_s(I)$ and *false* otherwise.

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

It is straightforward to confirm that the above definition assigns to every expression an element of the corresponding semantic domain, as stated in the following lemma:

Lemma 1. *Let P be a program and let $\mathsf{E} : \rho$ be an expression. Also, let I be a Herbrand interpretation and s be a Herbrand state. Then $[\![\mathsf{E}]\!]_s(I) \in [\![\rho]\!]$.*

Definition 14. Let P be a program and M be a Herbrand interpretation of P. Then, M is a Herbrand model of P iff for every clause $\mathsf{p}\ \mathsf{V}_1 \cdots \mathsf{V}_n \leftarrow \mathsf{E}_1, \ldots \mathsf{E}_m$ in P and for every Herbrand state s, if for all $i \in \{1, \ldots, m\}$, $[\![\mathsf{E}_i]\!]_s(M) = \mathit{true}$ then $[\![\mathsf{p}\ \mathsf{V}_1 \cdots \mathsf{V}_n]\!]_s(M) = \mathit{true}$.

In the following we denote the set of Herbrand interpretations of a program P with $\mathscr{I}_\mathsf{P}$. We define a partial order on $\mathscr{I}_\mathsf{P}$ as follows: for all $I, J \in \mathscr{I}_\mathsf{P}$, $I \sqsubseteq_{\mathscr{I}_\mathsf{P}} J$ iff for every predicate $\mathsf{p} : \pi$ that appears in P, $I(\mathsf{p}) \sqsubseteq_\pi J(\mathsf{p})$. Similarly, we denote the set of Herbrand states with $\mathscr{S}_\mathsf{P}$ and we define a partial order as follows: for all $s_1, s_2 \in \mathscr{S}_\mathsf{P}$, $s_1 \sqsubseteq_{\mathscr{S}_\mathsf{P}} s_2$ iff for all variables $\mathsf{V} : \rho$, $s_1(\mathsf{V}) \sqsubseteq_\rho s_2(\mathsf{V})$. The following lemmata are straightforward to establish:

Lemma 2. *Let P be a program. Then, $(\mathscr{I}_\mathsf{P}, \sqsubseteq_{\mathscr{I}_\mathsf{P}})$ is a complete lattice.*

Lemma 3. *Let P be a program and let $\mathsf{E} : \rho$ be an expression. Let I, J be Herbrand interpretations and s, s' be Herbrand states. Then,*

1. *If $I \sqsubseteq_{\mathscr{I}_\mathsf{P}} J$ then $[\![\mathsf{E}]\!]_s(I) \sqsubseteq_\rho [\![\mathsf{E}]\!]_s(J)$.*
2. *If $s \sqsubseteq_{\mathscr{I}_\mathsf{P}} s'$ then $[\![\mathsf{E}]\!]_s(I) \sqsubseteq_\rho [\![\mathsf{E}]\!]_{s'}(I)$.*

We can now define the *immediate consequence operator* for $\mathscr{H}$ programs, which generalizes the corresponding operator for classical (first-order) programs [8].

Definition 15. Let P be a program. The mapping $T_\mathsf{P} : \mathscr{I}_\mathsf{P} \to \mathscr{I}_\mathsf{P}$ is called the *immediate consequence operator for* P and is defined for every predicate $\mathsf{p} : \rho_1 \to \cdots \to \rho_n \to o$ and $d_i \in [\![\rho_i]\!]$ as

$$T_\mathsf{P}(I)(\mathsf{p})\ d_1 \cdots d_n = \begin{cases} \mathit{true} & \text{there exists a clause } \mathsf{p}\ \mathsf{V}_1 \cdots \mathsf{V}_n \leftarrow \mathsf{E}_1, \ldots \mathsf{E}_m \text{ such that} \\ & \text{for every state } s,\ [\![\mathsf{E}_i]\!]_{s[\mathsf{V}_1/d_1, \ldots, \mathsf{V}_n/d_n]}(I) = \mathit{true} \text{ for all } i \in \{1, \ldots, m\} \\ \mathit{false} & \text{otherwise.} \end{cases}$$

It is not hard to see that T_P is a monotonic function, and this leads to the following theorem [10, 7]:

Theorem 1. *Let P be a program. Then $M_\mathsf{P} = \mathit{lfp}(T_\mathsf{P})$ is the minimum, with respect to $\sqsubseteq_{\mathscr{I}_\mathsf{P}}$, Herbrand model of P.*

3.3 Bezem's Semantics

In contrast to Wadge's semantics which proceeds by constructing the meaning of predicates as relations, Bezem's approach takes a (seemingly) more syntax-oriented approach. In particular, Bezem's approach builds on the ground instantiation of the source program in order to retrieve the meaning of the program. In our definitions below, we follow relatively closely the exposition given in [1, 2, 3].

Definition 16. Let P be a program and let $\mathsf{Gr}(\mathsf{P})$ be its ground instantiation. An interpretation I for $\mathsf{Gr}(\mathsf{P})$ is defined as a subset of $U_{\mathsf{P},o}$ by the usual convention that, for any $\mathsf{A} \in U_{\mathsf{P},o}$, $I(\mathsf{A}) = \mathit{true}$ iff $\mathsf{A} \in I$. We also extend the interpretation I for every $(\mathsf{E}_1 \approx \mathsf{E}_2)$ atom as follows: $I(\mathsf{E}_1 \approx \mathsf{E}_2) = \mathit{true}$ if $\mathsf{E}_1 = \mathsf{E}_2$ and *false* otherwise.

Observe that the meaning of $(\mathsf{E}_1 \approx \mathsf{E}_2)$ is fixed and independent of the interpretation.

Definition 17. We define the immediate consequence operator, $\mathscr{T}_{\mathsf{Gr}(\mathsf{P})}$, of P as follows:

$$\mathscr{T}_{\mathsf{Gr}(\mathsf{P})}(I)(\mathsf{A}) = \begin{cases} \mathit{true} & \text{if there exists a clause } \mathsf{A} \leftarrow \mathsf{E}_1, \ldots, \mathsf{E}_m \text{ in } \mathsf{Gr}(\mathsf{P}) \\ & \text{such that } I(\mathsf{E}_i) = \mathit{true} \text{ for all } i \in \{1, \ldots, m\} \\ \mathit{false} & \text{otherwise.} \end{cases}$$

As it is well established in bibliography (for example [8]), the least fixed-point of the immediate consequence operator of a propositional program exists and is the minimum, with respect to set inclusion and equivalently $\leq$, model of $\mathsf{Gr}(\mathsf{P})$. This fixed-point, which we will henceforth denote by $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}$, is shown in [1, 2] to be directly related to a notion of a model capable of capturing the perceived semantics of the higher-order program P. In particular, this model by definition assigns to all ground atoms the same truth values as $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}$. It is therefore justified that we restrict our attention to $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}$, instead of the aforementioned higher-order model, in our attempt to prove the equivalence of Bezem's semantics and Wadge's semantics.

The following definition and subsequent theorem obtained in [3], identify a property of $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}$ that we will need in the next section.

Definition 18. Let P be a program and let $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}$ be the $\leq$-minimum model of $\mathsf{Gr}(\mathsf{P})$. For every argument type ρ we define a corresponding partial order as follows: for type ι, we define $\preceq_\iota$ as syntactical equality, i.e. $\mathsf{E} \preceq_\iota \mathsf{E}$ for all $\mathsf{E} \in U_{\mathsf{P},\iota}$. For type o, $\mathsf{E} \preceq_o \mathsf{E}'$ iff $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{E}) \leq \mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{E}')$. For a predicate type of the form $\rho \to \pi$, $\mathsf{E} \preceq_{\rho\to\pi} \mathsf{E}'$ iff $\mathsf{E}\,\mathsf{D} \preceq_\pi \mathsf{E}'\,\mathsf{D}$ for all $\mathsf{D} \in U_{\mathsf{P},\rho}$.

Theorem 2 ($\preceq$-Monotonicity Property)**.** *[3] Let P be a program and $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}$ be the $\leq$-minimum model of $\mathsf{Gr}(\mathsf{P})$. Then for all $\mathsf{E} \in U_{\mathsf{P},\rho\to\pi}$ and all $\mathsf{D}, \mathsf{D}' \in U_{\mathsf{P},\rho}$ such that $\mathsf{D} \preceq_\rho \mathsf{D}'$, it holds $\mathsf{E}\,\mathsf{D} \preceq_\pi \mathsf{E}\,\mathsf{D}'$.*

4 Equivalence of the two Semantics

In this section we demonstrate that the two semantics presented in the previous section, are equivalent for definitional programs. To help us transcend the differences between these approaches, we introduce two key notions, namely that of the *ground restriction* of a higher-order interpretation and its complementary notion of the *semantic extension* of ground expressions. But first we present the following *Substitution Lemma*, which will be useful in the proofs of later results.

Lemma 4 (Substitution Lemma)**.** *Let P be a program and I be a Herbrand interpretation of P. Also let E be an expression and θ be a ground substitution with $vars(\mathsf{E}) \subseteq dom(\theta)$. If s is a Herbrand state such that, for all $\mathsf{V} \in vars(\mathsf{E})$, $s(\mathsf{V}) = [\![\theta(\mathsf{V})]\!](I)$, then $[\![\mathsf{E}]\!]_s(I) = [\![\mathsf{E}\theta]\!](I)$.*

Proof. By a structural induction on E. For the basis case, if $\mathsf{E} = \mathsf{p}$ or $\mathsf{E} = \mathsf{c}$ then the statement reduces to an identity and if $\mathsf{E} = \mathsf{V}$ then it holds by assumption. For the induction step, we first examine the case that $\mathsf{E} = (\mathsf{f}\ \mathsf{E}_1\ \cdots\ \mathsf{E}_n)$; then $[\![\mathsf{E}]\!]_s(I) = I(\mathsf{f})\ [\![\mathsf{E}_1]\!]_s(I)\ \cdots\ [\![\mathsf{E}_n]\!]_s(I)$ and $[\![\mathsf{E}\theta]\!](I) = I(\mathsf{f})\ [\![\mathsf{E}_1\theta]\!](I)\ \cdots\ [\![\mathsf{E}_n\theta]\!](I)$. By the induction hypothesis, $[\![\mathsf{E}_1]\!]_s(I) = [\![\mathsf{E}_1\theta]\!](I), \ldots, [\![\mathsf{E}_n]\!]_s(I) = [\![\mathsf{E}_n\theta]\!](I)$, thus we have $[\![\mathsf{E}]\!]_s(I) = [\![\mathsf{E}\theta]\!](I)$. Now consider the case that $\mathsf{E} = \mathsf{E}_1\,\mathsf{E}_2$. We have $[\![\mathsf{E}]\!]_s(I) = [\![\mathsf{E}_1]\!]_s(I)[\![\mathsf{E}_2]\!]_s(I)$ and $[\![\mathsf{E}\theta]\!](I) = [\![\mathsf{E}_1\theta]\!](I)[\![\mathsf{E}_2\theta]\!](I)$. Again, applying the induction hypothesis, we conclude that $[\![\mathsf{E}]\!]_s(I) = [\![\mathsf{E}\theta]\!](I)$. Finally, if $\mathsf{E} = (\mathsf{E}_1 \approx \mathsf{E}_2)$ we have that $[\![\mathsf{E}]\!]_s(I) = \mathit{true}$ iff $[\![\mathsf{E}_1]\!]_s(I) = [\![\mathsf{E}_2]\!]_s(I)$, which, by the induction hypothesis, holds iff $[\![\mathsf{E}_1\theta]\!](I) = [\![\mathsf{E}_2\theta]\!](I)$. Moreover, we have $[\![\mathsf{E}\theta]\!](I) = \mathit{true}$ iff $[\![\mathsf{E}_1\theta]\!](I) = [\![\mathsf{E}_2\theta]\!](I)$, therefore we conclude that $[\![\mathsf{E}]\!]_s(I) = \mathit{true}$ iff $[\![\mathsf{E}\theta]\!](I) = \mathit{true}$. □

Given a Herbrand interpretation I of a definitional program, it is straightforward to devise a corresponding interpretation of the ground instantiation of the program, by restricting I to only assigning truth values to ground atoms. As expected, such a restriction of a model of the program produces a model of its ground instantiation. This idea is formalized in the following definition and theorem.

Definition 19. Let P be a program, I be a Herbrand interpretation of P and $\mathsf{Gr}(\mathsf{P})$ be the ground instantiation of P. We define the *ground restriction* of I, which we denote by $I|_{\mathsf{Gr}(\mathsf{P})}$, to be an interpretation of $\mathsf{Gr}(\mathsf{P})$, such that, for every ground atom A, $I|_{\mathsf{Gr}(\mathsf{P})}(\mathsf{A}) = [\![\mathsf{A}]\!](I)$.

Theorem 3. *Let* P *be a program and* $\mathsf{Gr}(\mathsf{P})$ *be its ground instantiation. Also let* M *be a Herbrand model of* P *and* $M|_{\mathsf{Gr}(\mathsf{P})}$ *be the ground restriction of* M*. Then* $M|_{\mathsf{Gr}(\mathsf{P})}$ *is a model of* $\mathsf{Gr}(\mathsf{P})$*.*

Proof. By definition, each clause in $\mathsf{Gr}(\mathsf{P})$ is of the form $\mathsf{p}\,\mathsf{E}_1 \cdots \mathsf{E}_n \leftarrow \mathsf{B}_1\theta, \ldots, \mathsf{B}_k\theta$, i.e. the ground instantiation of a clause $\mathsf{p}\,\mathsf{V}_1 \cdots \mathsf{V}_n \leftarrow \mathsf{B}_1, \ldots, \mathsf{B}_k$ in P with respect to a ground substitution θ, such that $\mathit{dom}(\theta)$ includes $V_1, \ldots, V_n$ and all other (individual) variables appearing in the body of the clause and $\theta(\mathsf{V}_i) = \mathsf{E}_i$, for all $i \in \{1, \ldots, n\}$. Let s be a Herbrand state such that $s(\mathsf{V}) = [\![\theta(\mathsf{V})]\!](M)$, for all $V \in \mathit{dom}(\theta)$. By the Substitution Lemma (Lemma 4) and the definition of $M|_{\mathsf{Gr}(\mathsf{P})}$, $[\![\mathsf{p}\,\mathsf{V}_1 \cdots \mathsf{V}_n]\!]_s(M) = [\![\mathsf{p}\,\mathsf{E}_1 \cdots \mathsf{E}_n]\!](M) = M|_{\mathsf{Gr}(\mathsf{P})}(\mathsf{p}\,\mathsf{E}_1 \cdots \mathsf{E}_n)$. Similarly, for each atom B_i in the body of the clause, we have $[\![\mathsf{B}_i]\!]_s(M) = [\![\mathsf{B}_i\theta]\!](M) = M|_{\mathsf{Gr}(\mathsf{P})}(\mathsf{B}_i\theta), 1 \leq i \leq k$. Consequently, if $M|_{\mathsf{Gr}(\mathsf{P})}(\mathsf{B}_i\theta) = \mathit{true}$ for all $i \in \{1, \ldots, k\}$, we also have that $[\![\mathsf{B}_i]\!]_s(M) = \mathit{true}, 1 \leq i \leq k$. As M is a model of P, this implies that $[\![\mathsf{p}\,\mathsf{V}_1 \cdots \mathsf{V}_n]\!]_s(M) = M|_{\mathsf{Gr}(\mathsf{P})}(\mathsf{p}\,\mathsf{E}_1 \cdots \mathsf{E}_n) = \mathit{true}$ and therefore $M|_{\mathsf{Gr}(\mathsf{P})}$ is a model of $\mathsf{Gr}(\mathsf{P})$. □

The above theorem is of course useful in connecting the $\sqsubseteq_{\mathscr{I}_\mathsf{P}}$-minimum Herbrand model of a program to its ground instantiation. However, in order to prove the equivalence of the two semantics under consideration, we will also need to go in the opposite direction and connect the $\leq$-minimum model of the ground program to the higher-order program. To this end we introduce the previously mentioned *semantic extensions* of a ground expression.

Definition 20. Let P be a program and $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}$ be the $\leq$-minimum model of $\mathsf{Gr}(\mathsf{P})$. Let E be a ground expression of argument type ρ and d be an element of $[\![\rho]\!]$. We will say that d is a semantic extension of E and write $d \rhd_\rho \mathsf{E}$ if

- $\rho = \iota$ and $d = \mathsf{E}$;
- $\rho = o$ and $d = \mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{E})$;
- $\rho = \rho' \to \pi$ and for all $d' \in [\![\rho']\!]$ and $\mathsf{E}' \in U_{P,\rho'}$, such that $d' \rhd_{\rho'} \mathsf{E}'$, it holds that $d\,d' \rhd_\pi \mathsf{E}\,\mathsf{E}'$.

Compared to that of the ground restriction presented earlier, the notion of extending a syntactic object to the realm of semantic elements, is more complicated. In fact, even the existence of a semantic extension is not immediately obvious. The next lemma guarantees that not only can such an extension

be constructed for any expression of the language, but it also has an interesting property of mirroring the ordering of semantic objects with respect to $\sqsubseteq_\tau$ in a corresponding ordering of the expressions with respect to $\preceq_\tau$.

Lemma 5. *Let* P *be a program,* $\mathsf{Gr}(\mathsf{P})$ *be its ground instantiation and* $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}$ *be the* $\leq$*-minimum model of* $\mathsf{Gr}(\mathsf{P})$*. For every argument type* ρ *and every ground term* $\mathsf{E} \in U_{\mathsf{P},\rho}$

1. *There exists* $e \in [\![\rho]\!]$ *such that* $e \rhd_\rho \mathsf{E}$.
2. *For all* $e, e' \in [\![\rho]\!]$ *and all* $\mathsf{E}' \in U_{\mathsf{P},\rho}$*, if* $e \rhd_\rho \mathsf{E}$*,* $e' \rhd_\rho \mathsf{E}'$ *and* $e \sqsubseteq_\rho e'$*, then* $\mathsf{E} \preceq_\rho \mathsf{E}'$.

Proof. We prove both statements simultaneously, performing an induction on the structure of ρ. Specifically, the first statement is proven by showing that in each case we can construct a function e of type ρ, which is monotonic with respect to $\sqsubseteq_\rho$ and satisfies $e \rhd_\rho \mathsf{E}$.

In the basis case, the construction of e for types ι and o is trivial. Also, if $\rho = \iota$, then both $\rhd_\rho$ and $\sqsubseteq_\rho$ reduce to equality, so we have $\mathsf{E} = \mathsf{E}'$, which in this case is equivalent to $\mathsf{E} \preceq_\rho \mathsf{E}'$. On the other hand, for $\rho = o$, $\rhd_\rho$ identifies with equality, while $\sqsubseteq_\rho$ and $\preceq_\rho$ identify with $\leq$, so we have that $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{E}) = e \leq e' = \mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{E}')$ implies $\mathsf{E} \preceq_\rho \mathsf{E}'$.

For a more complex type $\rho = \rho_1 \to \cdots \to \rho_n \to o$, $n > 0$, we can easily construct e, as follows:

$$e\, e_1 \cdots e_n = \begin{cases} \mathit{true}, & \text{if there exist } d_1, \ldots, d_n \text{ and ground terms } \mathsf{D}_1, \ldots, \mathsf{D}_n \text{ such that,} \\ & \text{for all } i, d_i \sqsubseteq_{\rho_i} e_i, d_i \rhd_{\rho_i} \mathsf{D}_i \text{ and } \mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{E}\,\mathsf{D}_1 \cdots \mathsf{D}_n) = \mathit{true} \\ \mathit{false}, & \text{otherwise.} \end{cases}$$

To see that e is monotonic, consider $e_1, \ldots, e_n, e'_1, \ldots, e'_n$, such that $e_1 \sqsubseteq_{\rho_1} e'_1, \ldots, e_n \sqsubseteq_{\rho_n} e'_n$ and observe that $e\, e_1 \cdots e_n = \mathit{true}$ implies $e\, e'_1 \cdots e'_n = \mathit{true}$, due to the transitivity of $\sqsubseteq_{\rho_i}$. We will now show that $e \rhd_\rho \mathsf{E}$, i.e. for all $e_1, \ldots, e_n$ and $\mathsf{E}_1, \ldots, \mathsf{E}_n$ such that $e_1 \rhd_{\rho_1} \mathsf{E}_1, \ldots, e_n \rhd_{\rho_n} \mathsf{E}_n$, it holds $e\, e_1 \cdots e_n = \mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{E}\,\mathsf{E}_1 \cdots \mathsf{E}_n)$. This is trivial if $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{E}\,\mathsf{E}_1 \cdots \mathsf{E}_n) = \mathit{true}$, since $e_i \sqsubseteq_{\rho_i} e_i$. Let us now examine the case that $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{E}\,\mathsf{E}_1 \cdots \mathsf{E}_n) = \mathit{false}$. For the sake of contradiction, assume $e\, e_1 \cdots e_n = \mathit{true}$. Then, by the construction of e, there must exist $d_1, \ldots, d_n$ and $\mathsf{D}_1, \ldots, \mathsf{D}_n$ such that, for all i, $d_i \sqsubseteq_{\rho_i} e_i$, $d_i \rhd_{\rho_i} \mathsf{D}_i$ and $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{E}\,\mathsf{D}_1 \cdots \mathsf{D}_n) = \mathit{true}$. By the induction hypothesis, we have that $\mathsf{D}_i \preceq_{\rho_i} \mathsf{E}_i$, for all $i \in \{1, \ldots, n\}$. This, by the $\preceq$-Monotonicity Property of $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}$ (Theorem 2), yields that $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{E}\,\mathsf{D}_1 \cdots \mathsf{D}_n) = \mathit{true} \leq \mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{E}\,\mathsf{E}_1 \cdots \mathsf{E}_n) = \mathit{false}$, which is obviously a contradiction. Therefore it has to be that $e\, e_1 \cdots e_n = \mathit{false}$.

Finally, in order to prove the second statement and conclude the induction step, we need to show that for all terms $\mathsf{D}_1 \in U_{\mathsf{P},\rho_1}, \ldots, \mathsf{D}_n \in U_{\mathsf{P},\rho_n}$, it holds $\mathsf{E}\,\mathsf{D}_1 \cdots \mathsf{D}_n \preceq_o \mathsf{E}'\,\mathsf{D}_1 \cdots \mathsf{D}_n$. By the induction hypothesis, there exist $d_1, \ldots, d_n$, such that $d_1 \rhd_{\rho_1} \mathsf{D}_1, \ldots, d_n \rhd_{\rho_n} \mathsf{D}_n$. Because $e \rhd_\rho \mathsf{E}$ and $\mathsf{E}\,\mathsf{D}_1 \cdots \mathsf{D}_n$ is of type o, we have $e\, d_1 \cdots d_n = \mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{E}\,\mathsf{D}_1 \cdots \mathsf{D}_n)$ by definition. Similarly, we also have $e'\, d_1 \cdots d_n = \mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{E}'\,\mathsf{D}_1 \cdots \mathsf{D}_n)$. Moreover, by $e \sqsubseteq_\rho e'$ we have that $e\, d_1 \cdots d_n \sqsubseteq_o e'\, d_1 \cdots d_n$. This yields the desired result, since $\sqsubseteq_o$ identifies with $\preceq_o$. □

The following variation of the Substitution Lemma states that if the building elements of an expression are assigned meanings that are semantic extensions of their syntactic counterparts, then the meaning of the expression is itself a semantic extension of the expression.

Lemma 6. *Let* P *be a program,* $\mathsf{Gr}(\mathsf{P})$ *be its ground instantiation and I be a Herbrand interpretation of* P*. Also, let* E *be an expression of some argument type* ρ *and let s be a Herbrand state and* θ *be a ground substitution, both with domain vars*(E)*. If, for all predicates* p *of type* π *appearing in* E*,* $[\![\mathsf{p}]\!](I) \rhd_\pi \mathsf{p}$ *and, for all variables* V *of type* ρ' *in vars*(E)*,* $s(\mathsf{V}) \rhd_{\rho'} \theta(\mathsf{V})$*, then* $[\![\mathsf{E}]\!]_s(I) \rhd_\rho \mathsf{E}\theta$.

Proof. The proof is by induction on the structure of E. The basis cases $\mathsf{E} = \mathsf{p}$ and $\mathsf{E} = \mathsf{V}$ hold by assumption and $\mathsf{E} = c : \iota$ is trivial. For the first case of the induction step, let $\mathsf{E} = (\mathsf{f}\ \mathsf{E}_1\ \cdots\ \mathsf{E}_n)$, where $\mathsf{E}_1, \ldots, \mathsf{E}_n$ are of type ι. By the induction hypothesis, we have that $[\![\mathsf{E}_1]\!]_s(I) \rhd_\iota \mathsf{E}_1\theta, \ldots, [\![\mathsf{E}_n]\!]_s(I) \rhd_\iota \mathsf{E}_n\theta$. As $\rhd_\iota$ is defined as equality, we have that $[\![\mathsf{E}]\!]_s(I) = I(\mathsf{f})\ [\![\mathsf{E}_1]\!]_s(I)\ \cdots\ [\![\mathsf{E}_n]\!]_s(I) = \mathsf{f}\ \mathsf{E}_1\theta\ \cdots\ \mathsf{E}_n\theta = \mathsf{E}\theta$ and therefore $[\![\mathsf{E}]\!]_s(I) \rhd_\iota \mathsf{E}\theta$. For the second case, let $\mathsf{E} = \mathsf{E}_1\,\mathsf{E}_2$, where E_1 is of type $\rho_1 = \rho_2 \to \pi$ and E_2 is of type ρ_2; then, $[\![\mathsf{E}]\!]_s(I) = [\![\mathsf{E}_1]\!]_s(I)\,[\![\mathsf{E}_2]\!]_s(I)$. By the induction hypothesis, $[\![\mathsf{E}_1]\!]_s(I) \rhd_{\rho_2\to\pi} \mathsf{E}_1\theta$ and $[\![\mathsf{E}_2]\!]_s(I) \rhd_{\rho_2} \mathsf{E}_2\theta$, thus, by definition, $[\![\mathsf{E}]\!]_s(I) = [\![\mathsf{E}_1]\!]_s(I)\ [\![\mathsf{E}_2]\!]_s(I) \rhd_\pi \mathsf{E}_1\theta\ \mathsf{E}_2\theta = (\mathsf{E}_1\,\mathsf{E}_2)\theta = \mathsf{E}\theta$. Finally, we have the case that $\mathsf{E} = (\mathsf{E}_1 \approx \mathsf{E}_2)$, where E_1 and E_2 are both of type ι. The induction hypothesis yields $[\![\mathsf{E}_1]\!]_s(I) \rhd_\iota \mathsf{E}_1\theta$ and $[\![\mathsf{E}_2]\!]_s(I) \rhd_\iota \mathsf{E}_2\theta$ or, since $\rhd_\iota$ is defined as equality, $[\![\mathsf{E}_1]\!]_s(I) = \mathsf{E}_1\theta$ and $[\![\mathsf{E}_2]\!]_s(I) = \mathsf{E}_2\theta$. Then $[\![\mathsf{E}_1]\!]_s(I) = [\![\mathsf{E}_2]\!]_s(I)$ iff $\mathsf{E}_1\theta = \mathsf{E}_2\theta$ and, equivalently, $[\![\mathsf{E}]\!]_s(I) = \mathit{true}$ iff $\mathsf{E}\theta = \mathit{true}$, which implies $[\![\mathsf{E}]\!]_s(I) \rhd_o \mathsf{E}\theta$. □

We are now ready to present the main result of this paper. The theorem establishes the equivalence of Wadge's semantics and Bezem's semantics, in stating that their respective minimum models assign the same meaning to all ground atoms.

Theorem 4. *Let* P *be a program and let* $\mathsf{Gr}(\mathsf{P})$ *be its ground instantiation. Let* M_P *be the* $\sqsubseteq_{\mathscr{I}_\mathsf{P}}$*-minimum Herbrand model of* P *and let* $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}$ *be the* $\leq$*-minimum model of* $\mathsf{Gr}(\mathsf{P})$*. Then, for every* $\mathsf{A} \in U_{\mathsf{P},o}$ *it holds* $[\![\mathsf{A}]\!](M_\mathsf{P}) = \mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{A})$.

Proof. We will construct an interpretation N for P and prove some key properties for this interpretation. Then we will utilize these properties to prove the desired result. The definition of N is as follows:

For every $\mathsf{p} : \rho_1 \to \cdots \to \rho_n \to o$ and all $d_1 \in [\![\rho_1]\!], \ldots, d_n \in [\![\rho_n]\!]$

$$N(\mathsf{p})\,d_1 \cdots d_n = \begin{cases} \mathit{false}, & \text{if there exist } e_1, \ldots, e_n \text{ and ground terms } \mathsf{E}_1, \ldots, \mathsf{E}_n \text{ such that,} \\ & \text{for all } i, d_i \sqsubseteq_{\rho_i} e_i, e_i \rhd_{\rho_i} \mathsf{E}_i \text{ and } \mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{p}\,\mathsf{E}_1 \cdots \mathsf{E}_n) = \mathit{false} \\ \mathit{true}, & \text{otherwise} \end{cases}$$

Observe that N is a valid Herbrand interpretation of P, in the sense that it assigns elements in $[\![\pi]\!]$ (i.e. functions that are monotonic with respect to $\sqsubseteq_\pi$) to every predicate of type π in P. Indeed, if it was not so, then for some predicate $\mathsf{p} : \pi = \rho_1 \to \cdots \to \rho_n \to o$, there would exist tuples $(d_1, \ldots, d_n)$ and $(d'_1, \ldots, d'_n)$ with $d_1 \sqsubseteq_{\rho_1} d'_1, \ldots, d_n \sqsubseteq_{\rho_n} d'_n$, such that $N(\mathsf{p})\,d_1 \cdots d_n = \mathit{true}$ and $N(\mathsf{p})\,d'_1 \cdots d'_n = \mathit{false}$. By definition, the fact that $N(\mathsf{p})\,d'_1 \cdots d'_n$ is assigned the value *false*, would imply that there exist $e_1, \ldots, e_n$ and $\mathsf{E}_1, \ldots, \mathsf{E}_n$ as in the above definition, such that $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{p}\,\mathsf{E}_1 \cdots \mathsf{E}_n) = \mathit{false}$ and $d'_1 \sqsubseteq_{\rho_1} e_1, \ldots, d'_n \sqsubseteq_{\rho_n} e_n$. Being that $\sqsubseteq_{\rho_i}$ are transitive relations, the latter yields that $d_1 \sqsubseteq_{\rho_1} e_1, \ldots, d_n \sqsubseteq_{\rho_n} e_n$. Therefore, by definition, $N(\mathsf{p})\,d_1 \cdots d_n$ should also evaluate to *false*, which constitutes a contradiction and thus confirms that the meaning of p is monotonic with respect to $\sqsubseteq_\pi$.

It is also straightforward to see that $N(\mathsf{p}) \rhd_\pi \mathsf{p}$, i.e. for all $d_1, \ldots, d_n$ and all ground terms $\mathsf{D}_1, \ldots, \mathsf{D}_n$ such that $d_1 \rhd_{\rho_1} \mathsf{D}_1, \ldots, d_n \rhd_{\rho_n} \mathsf{D}_n$, we have $N(\mathsf{p})\,d_1 \cdots d_n = \mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{p}\ \mathsf{D}_1 \cdots \mathsf{D}_n)$. Because $d_i \sqsubseteq_{\rho_i} d_i$, this holds trivially if $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{p}\ \mathsf{D}_1 \cdots \mathsf{D}_n) = \mathit{false}$. Now let $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{p}\ \mathsf{D}_1 \cdots \mathsf{D}_n) = \mathit{true}$ and assume, for the sake of contradiction, that $N(\mathsf{p})\,d_1 \cdots d_n = \mathit{false}$. Then, by the definition of N, there must exist $e_1, \ldots, e_n$ and $\mathsf{E}_1, \ldots, \mathsf{E}_n$ such that, for all i, $d_i \sqsubseteq_{\rho_i} e_i$, $e_i \rhd_{\rho_i} \mathsf{E}_i$ and $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{p}\,\mathsf{E}_1 \cdots \mathsf{E}_n) = \mathit{false}$. Thus, by the second part of Lemma 5, for all i, $\mathsf{D}_i \preceq_{\rho_i} \mathsf{E}_i$ and, by the $\preceq$-Monotonicity Property of $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}$, $\mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{p}\,\mathsf{D}_1 \cdots \mathsf{D}_n) \leq \mathscr{M}_{\mathsf{Gr}(\mathsf{P})}(\mathsf{p}\ \mathsf{E}_1 \cdots \mathsf{E}_n)$, which is obviously a contradiction. Thus we conclude that $N(\mathsf{p})\,d_1 \cdots d_n = \mathit{true}$.

Next we prove that N is a model of P. Let $\mathsf{p}\mathsf{V}_1 \cdots \mathsf{V}_n \leftarrow \mathsf{B}_1, \ldots, \mathsf{B}_k$ be a clause in P and let $\{\mathsf{V}_1, \ldots, \mathsf{V}_n, \mathsf{X}_1, \ldots, \mathsf{X}_m\}$, with $\mathsf{V}_i : \rho_i$, for all $i \in \{1, \ldots, n\}$, and $\mathsf{X}_i : \iota$, for all $i \in \{1, \ldots, m\}$, be the set of

variables appearing in the clause. Then, it suffices to show that, for any tuple $(d_1,\ldots,d_n)$ of arguments and any Herbrand state s such that $s(\mathsf{V}_i)=d_i$ for all $i\in\{1,\ldots,n\}$, $N(\mathsf{p})\,d_1\cdots d_n=\mathit{false}$ implies that, for at least one $j\in\{1,\ldots,k\}$, $[\![\mathsf{B}_j]\!]_s(N)=\mathit{false}$. Again, by the definition of N, we see that if $N(\mathsf{p})\,d_1\cdots d_n=\mathit{false}$, then there exist $e_1,\ldots,e_n$ and ground terms $\mathsf{E}_1,\ldots,\mathsf{E}_n$ such that $\mathscr{M}_{\mathsf{Gr(P)}}(\mathsf{p}\,\mathsf{E}_1\cdots\mathsf{E}_n)=\mathit{false}$, $d_1\sqsubseteq_{\rho_1} e_1,\ldots,d_n\sqsubseteq_{\rho_n} e_n$ and $e_1\rhd_{\rho_1}\mathsf{E}_1,\ldots,d_n\rhd_{\rho_n}\mathsf{E}_n$. Let θ be a ground substitution such that $\theta(\mathsf{V}_i)=\mathsf{E}_i$ for all $i\in\{1,\ldots,n\}$ and, for all $i\in\{1,\ldots,m\}$, $\theta(\mathsf{X}_i)=s(\mathsf{X}_i)$; then there exists a ground instantiation $\mathsf{p}\,\mathsf{E}_1\cdots\mathsf{E}_n\leftarrow\mathsf{B}_1\theta,\ldots,\mathsf{B}_k\theta$ of the above clause in $\mathsf{Gr(P)}$. As $\mathscr{M}_{\mathsf{Gr(P)}}$ is a model of the ground program, $\mathscr{M}_{\mathsf{Gr(P)}}(\mathsf{p}\,\mathsf{E}_1\cdots\mathsf{E}_n)=\mathit{false}$ implies that there exists at least one $j\in\{1,\ldots,k\}$ such that $\mathscr{M}_{\mathsf{Gr(P)}}(\mathsf{B}_j\theta)=\mathit{false}$. We are going to show that the latter implies that $[\![\mathsf{B}_j]\!]_s(N)=\mathit{false}$, which proves that N is a model of P. Indeed, let s' be a Herbrand state such that $s'(\mathsf{V}_i)=e_i\rhd_{\rho_i}\theta(\mathsf{V}_i)=\mathsf{E}_i$ for all $i\in\{1,\ldots,n\}$ and $s'(\mathsf{X}_i)=\theta(\mathsf{X}_i)=s(\mathsf{X}_i)$ for all $i\in\{1,\ldots,m\}$. As we have shown earlier, $N(\mathsf{p}')\rhd_{\pi'}\mathsf{p}'$ for any predicate $\mathsf{p}':\pi'$, thus by Lemma 6 we get $[\![\mathsf{B}_j]\!]_{s'}(N)\rhd_o\mathsf{B}_j\theta$. Since B_j is of type o, the latter reduces to $[\![\mathsf{B}_j]\!]_{s'}(N)=\mathscr{M}_{\mathsf{Gr(P)}}(\mathsf{B}_j\theta)=\mathit{false}$. Also, because $d_i\sqsubseteq_{\rho_i}e_i$, i.e. $s\sqsubseteq_{\mathscr{I}_\mathsf{P}}s'$, by the second part of Lemma 3 we get $[\![\mathsf{B}_j]\!]_s(N)\sqsubseteq_o[\![\mathsf{B}_j]\!]_{s'}(N)$, which makes $[\![\mathsf{B}_j]\!]_s(N)=\mathit{false}$.

Now we can proceed to prove that, for all $\mathsf{A}\in U_{\mathsf{P},o}$, $[\![\mathsf{A}]\!](M_\mathsf{P})=\mathscr{M}_{\mathsf{Gr(P)}}(\mathsf{A})$. Let A be of the form $\mathsf{p}\,\mathsf{E}_1\cdots\mathsf{E}_n$, where $\mathsf{p}:\rho_1\to\cdots\to\rho_n\to o\in\mathsf{P}$ and let $d_1=[\![\mathsf{E}_1]\!](M_\mathsf{P}),\ldots,d_n=[\![\mathsf{E}_n]\!](M_\mathsf{P})$. As we have shown, N is a Herbrand model of P, while M_P is the minimum, with respect to $\sqsubseteq_{\mathscr{I}_\mathsf{P}}$, of all Herbrand models of P, therefore we have that $M_\mathsf{P}\sqsubseteq_{\mathscr{I}_\mathsf{P}}N$. By definition, this gives us that $M_\mathsf{P}(\mathsf{p})\,d_1\cdots d_n\sqsubseteq_o N(\mathsf{p})\,d_1\cdots d_n$ (1) and, by the first part of Lemma 3, that $d_1\sqsubseteq_{\rho_1}[\![\mathsf{E}_1]\!](N),\ldots,d_n\sqsubseteq_{\rho_n}[\![\mathsf{E}_n]\!](N)$ (2). Moreover, for all predicates $\mathsf{p}':\pi'$ in P, we have $N(\mathsf{p}')\rhd_{\pi'}\mathsf{p}'$ and thus, by Lemma 6, taking s and θ to be empty, we get $[\![\mathsf{E}_i]\!](N)\rhd_{\rho_i}\mathsf{E}_i, 1\le i\le n$. In conjunction with (2), the latter suggests that if $\mathscr{M}_{\mathsf{Gr(P)}}(\mathsf{p}\ \mathsf{E}_1\ \cdots\ \mathsf{E}_n)=\mathit{false}$ then $N(\mathsf{p})\ d_1\ \cdots\ d_n=\mathit{false}$, or, in other words, that $N(\mathsf{p})\ d_1\cdots d_n\le\mathscr{M}_{\mathsf{Gr(P)}}(\mathsf{p}\ \mathsf{E}_1\ \cdots\ \mathsf{E}_n)$. Because of (1), this makes it that $M_\mathsf{P}(\mathsf{p})\ d_1\cdots d_n\le\mathscr{M}_{\mathsf{Gr(P)}}(\mathsf{p}\ \mathsf{E}_1\ \cdots\ \mathsf{E}_n)$ (3). On the other hand, by Theorem 3, $M_\mathsf{P}|_{\mathsf{Gr(P)}}$ is a model of $\mathsf{Gr(P)}$ and therefore $\mathscr{M}_{\mathsf{Gr(P)}}(\mathsf{p}\ \mathsf{E}_1\ \cdots\ \mathsf{E}_n)\le M_\mathsf{P}|_{\mathsf{Gr(P)}}(\mathsf{p}\ \mathsf{E}_1\ \cdots\ \mathsf{E}_n)$, since $\mathscr{M}_{\mathsf{Gr(P)}}$ is the minimum model of $\mathsf{Gr(P)}$. By the definition of $\mathsf{Gr(P)}$ and the meaning of application, the latter becomes $\mathscr{M}_{\mathsf{Gr(P)}}(\mathsf{p}\ \mathsf{E}_1\ \cdots\ \mathsf{E}_n)\le M_\mathsf{P}|_{\mathsf{Gr(P)}}(\mathsf{p}\ \mathsf{E}_1\ \cdots\ \mathsf{E}_n)=M_\mathsf{P}(\mathsf{p}\ \mathsf{E}_1\ \cdots\ \mathsf{E}_n)=M_\mathsf{P}(\mathsf{p})\ [\![\mathsf{E}_1]\!](M_\mathsf{P})\ \cdots\ [\![\mathsf{E}_n]\!](M_\mathsf{P})=M_\mathsf{P}(\mathsf{p})\ d_1\cdots d_n$. The last relation and (3) can only be true simultaneously, if all the above relations hold as equalities, in particular if $\mathscr{M}_{\mathsf{Gr(P)}}(\mathsf{p}\,\mathsf{E}_1\cdots\mathsf{E}_n)=[\![\mathsf{p}\,\mathsf{E}_1\cdots\mathsf{E}_n]\!](M_\mathsf{P})$. □

5 Discussion

We have considered the two existing extensional approaches to the semantics of higher-order logic programming, and have demonstrated that they coincide for the class of definitional programs. It is therefore natural to wonder whether the two semantic approaches continue to coincide if we extend the class of programs we consider. Unfortunately this is not the case, as we discuss below.

A seemingly mild extension to our source language would be to allow higher-order predicate variables that are not formal parameters of a clause, to appear in its body. Such programs are legitimate under Bezem's semantics (ie., they belong to the hoapata class). Moreover, a recent extension of Wadge's semantics [5] also allows such programs. However, for this extended class of programs the equivalence of the two semantic approaches no longer holds as the following example illustrates.

Example 8. Consider the following extended program:

```
p(a):-Q(a).
```

Following Bezem's semantics, we initially take the ground instantiation of the program, namely:

```
p(a):-p(a).
```

and then compute the least model of the above program which assigns to the atom `p(a)` the value *false*. On the other hand, under the approach in [5], the atom `p(a)` has the value *true* in the minimum Herbrand model of the initial program. This is due to the fact that under the semantics of [5], our initial program reads (intuitively speaking) as follows: "`p(a)` is true if there exists a relation that is true of `a`"; actually, there exists one such relation, namely the set $\{\mathtt{a}\}$. This discrepancy between the two semantics is due to the fact that Wadge's semantics is based on *sets* and not solely on the syntactic entities that appear in the program. □

Future work includes the extension of Bezem's approach to higher-order logic programs with negation. An extension of Wadge's approach for such programs has recently been performed in [4]. More generally, the addition of negation to higher-order logic programming appears to offer an interesting and nontrivial area of research, which we are currently pursuing.

References

[1] Marc Bezem (1999): *Extensionality of Simply Typed Logic Programs*. In Danny De Schreye, editor: *Logic Programming: The 1999 International Conference, Las Cruces, New Mexico, USA, November 29 - December 4, 1999*, MIT Press, pp. 395–410.

[2] Marc Bezem (2001): *An Improved Extensionality Criterion for Higher-Order Logic Programs*. In Laurent Fribourg, editor: *Computer Science Logic, 15th International Workshop, CSL 2001. 10th Annual Conference of the EACSL, Paris, France, September 10-13, 2001, Proceedings*, *Lecture Notes in Computer Science* 2142, Springer, pp. 203–216, doi:10.1007/3-540-44802-0_15.

[3] Marc Bezem (2002): *Hoapata programs are monotonic*. In: *Proceedings NWPT02, Institute of Cybernetics at TTU, Tallinn*, pp. 18–20.

[4] Angelos Charalambidis, Zoltán Ésik & Panos Rondogiannis (2014): *Minimum Model Semantics for Extensional Higher-order Logic Programming with Negation*. *TPLP* 14(4-5), pp. 725–737, doi:10.1017/S1471068414000313.

[5] Angelos Charalambidis, Konstantinos Handjopoulos, Panagiotis Rondogiannis & William W. Wadge (2013): *Extensional Higher-Order Logic Programming*. *ACM Trans. Comput. Log.* 14(3), p. 21, doi:10.1145/2499937.2499942.

[6] Weidong Chen, Michael Kifer & David Scott Warren (1993): *HILOG: A Foundation for Higher-Order Logic Programming*. *Journal of Logic Programming* 15(3), pp. 187–230, doi:10.1016/0743-1066(93)90039-J.

[7] Vassilis Kountouriotis, Panos Rondogiannis & William W. Wadge (2005): *Extensional Higher-Order Datalog*. In: *Short Paper Proceeding of the 12th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, pp. 1–5.

[8] John W. Lloyd (1987): *Foundations of Logic Programming*. Springer Verlag, doi:10.1007/978-3-642-83189-8.

[9] Dale Miller & Gopalan Nadathur (2012): *Programming with Higher-Order Logic*, 1st edition. Cambridge University Press, New York, NY, USA, doi:10.1017/CBO9781139021326.

[10] William W. Wadge (1991): *Higher-Order Horn Logic Programming*. In Vijay A. Saraswat & Kazunori Ueda, editors: *Logic Programming, Proceedings of the 1991 International Symposium, San Diego, California, USA, Oct. 28 - Nov 1, 1991*, MIT Press, pp. 289–303.