Approximation Fixpoint Theory and the Well-Founded Semantics of Higher-Order Logic Programs

Angelos Charalambidis, Panos Rondogiannis and Ioanna Symeonidou

May 30, 2018

Approximation Fixpoint Theory and the Well-Founded Semantics of Higher-Order Logic Programs

ANGELOS CHARALAMBIDIS

Institute of Informatics and Telecommunications, NCSR "Demokritos", Greece
(*e-mail:* `acharal@iit.demokritos.gr`)

PANOS RONDOGIANNIS, IOANNA SYMEONIDOU

Department of Informatics and Telecommunications, University of Athens, Greece
(*e-mail:* `{prondo,sioanna}@di.uoa.gr`)

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

Abstract

We define a novel, extensional, three-valued semantics for higher-order logic programs with negation. The new semantics is based on interpreting the types of the source language as three-valued Fitting-monotonic functions at all levels of the type hierarchy. We prove that there exists a bijection between such Fitting-monotonic functions and pairs of two-valued-result functions where the first member of the pair is monotone-antimonotone and the second member is antimonotone-monotone. By deriving an extension of *consistent approximation fixpoint theory* (Denecker et al. 2004) and utilizing the above bijection, we define an iterative procedure that produces for any given higher-order logic program a distinguished extensional model. We demonstrate that this model is actually a *minimal* one. Moreover, we prove that our construction generalizes the familiar well-founded semantics for classical logic programs, making in this way our proposal an appealing formulation for capturing the *well-founded semantics for higher-order logic programs*.

KEYWORDS: Higher-Order Logic Programming, Negation in Logic Programming, Approximation Fixpoint Theory.

1 Introduction

An intriguing and difficult question regarding logic programming, is whether it can be extended to a higher-order setting without sacrificing its semantic simplicity and clarity. Research results in this direction (Wadge 1991; Bezem 1999; Charalambidis et al. 2013; Rondogiannis and Symeonidou 2016; Rondogiannis and Symeonidou 2017) strongly suggest that it is possible to design higher-order logic programming languages that have powerful expressive capabilities, and which, at the same time, retain all the desirable semantic properties of classical first-order logic programming. In particular, it has been shown that higher-order logic programming can be given an *extensional semantics*, namely one in which program predicates denote sets. Under such a semantics one can use standard set-theoretic concepts in order to understand the meaning of programs and reason about them. For a more detailed discussion of extensionality and its importance for higher-order logic programming, the interested reader can consult the discussion in Section 2 of (Rondogiannis and Symeonidou 2017).

The above line of research started many years ago by W. W. Wadge (Wadge 1991) who considered *positive* higher-order logic programs (i.e., programs without negation in clause bodies). Wadge argued that if such a program obeys some simple and natural syntactic rules, then it has a unique *minimum* Herbrand model. It is well-known that the *minimum model property* is a cornerstone of the theory of first-order logic programming (van Emden and Kowalski 1976). In this respect, Wadge's result suggested that it might be possible to extend all the elegant theory of classical logic programming to the higher-order case. The results in (Wadge 1991) were obtained using standard techniques from denotational semantics involving *continuous* interpretations and Kleene's least fixpoint theorem. A few years after Wadge's initial result, M. Bezem came to similar conclusions (Bezem 1999) but from a different direction. In particular, Bezem demonstrated that by using a fixpoint construction on the ground instantiation of the source higher-order program, one can obtain a model of the original program that satisfies an extensionality condition defined in (Bezem 1999). Despite their different philosophies, Wadge's and Bezem's approaches have recently been shown (Charalambidis et al. 2017) to have close connections. Apart from the above results, recent work (Charalambidis et al. 2013) has also shown that we can define a sound and complete proof procedure for positive higher-order logic programs, which generalizes classical SLD-resolution. In other words, the central results for positive first-order logic programs, generalize to the higher-order case.

A natural question that arises is whether one can still obtain an extensional semantics if negation is added to programs. Surprisingly, this question proved harder to resolve. The first result in this direction was reported in (Charalambidis et al. 2014), where it was demonstrated that every higher-order logic program with negation has a minimum extensional Herbrand model constructed over a logic with an infinite number of truth values. This result was obtained using domain-theoretic techniques as-well-as an extension of Kleene's fixpoint theorem that applies to a class of functions that are potentially non-monotonic (Ésik and Rondogiannis 2015). More recently, it was shown in (Rondogiannis and Symeonidou 2016) that Bezem's technique for positive programs can also be extended to apply to higher-order logic programs with negation, provided that it is interpreted under the same infinite-valued logic used in (Charalambidis et al. 2014). The above results, although satisfactory from a mathematical point of view, left open an annoying natural question: "Is it possible to define a *three-valued* extensional semantics for higher-order logic programs with negation that generalizes the standard well-founded semantics for classical logic programs?".

The above question was recently undertaken in (Rondogiannis and Symeonidou 2017). The surprising result was obtained that if Bezem's approach is interpreted under a three-valued logic, then the resulting semantics *can not be extensional* in the general case. One can see that similar arguments hold for the technique of (Charalambidis et al. 2014). Therefore, if we seek an extensional three-valued semantics for higher-order logic programs with negation, we need to follow an approach that is radically different from both (Charalambidis et al. 2014) and (Rondogiannis and Symeonidou 2017).

In this paper we undertake exactly the above problem. We demonstrate that we can indeed define a three-valued extensional semantics for higher-order logic programs with negation, which generalizes the familiar well-founded semantics of first-order logic programs (Gelder et al. 1991). Our results heavily utilize the technique of *approximation fixpoint theory* (Denecker et al. 2000; Denecker et al. 2004), which proved to be an indispensable tool in our investigation. The main contributions of the present paper can be outlined as follows:

- We define the first (to our knowledge) extensional three-valued semantics for higher-order logic programs with negation. Our semantics is based on interpreting the predicate types of our language as three-valued Fitting-monotonic functions (at all levels of the type hierarchy). We prove that there exists a bijection between such Fitting-monotonic functions and pairs of two-valued-result functions of the form (f_1, f_2), where f_1 is monotone-antimonotone, f_2 is antimonotone-monotone, and $f_1 \leq f_2$ (these notions will be explained in detail in Section 5).
- By deriving an extension of *consistent approximation fixpoint theory* (Denecker et al. 2004) and utilizing the above bijection, we define an iterative procedure that produces for any given higher-order logic program a distinguished extensional model. We prove that this model is actually a *minimal* one and we demonstrate that our construction generalizes the familiar well-founded semantics for classical logic programs. Therefore, we argue that our proposal is an appealing formulation for capturing the *well-founded semantics for higher-order logic programs*, paving in this way the road for a further study of negation in higher-order logic programming.

The rest of the paper is organized as follows. Section 2 presents in an intuitive way the main ideas developed in the paper. Section 3 introduces the syntax and Section 4 the semantics of our source language. Section 5 demonstrates the bijection between Fitting-monotonic functions and pairs of monotone-antimonotone and antimonotone-monotone functions. Section 6 develops the well-founded semantics of higher-order logic programs with negation, based on an extension of consistent approximation fixpoint theory. Section 7 compares the present work with that of (Charalambidis et al. 2014; Rondogiannis and Symeonidou 2017), and concludes by identifying some promising research directions. The proofs of most results of the paper are given in the appendices.

2 An Intuitive Overview of the Proposed Approach

In this section we describe in an intuitive way the main ideas and results obtained in the paper. As we have already mentioned, our goal is to derive a generalization of the well-founded semantics for higher-order logic programs with negation.

We start with our source language $\mathcal{HOL}$ which, intuitively speaking, allows *distinct* predicate variables (but not predicate constants) to appear in the heads of clauses. This is a syntactic restriction initially introduced in (Wadge 1991), which has been preserved and used by all subsequent articles in the area. As an example, consider the following program (for the moment we use ad-hoc Prolog-like syntax):

Example 1
The program below defines the `subset` relation over two unary predicates `P` and `Q`:

$$\texttt{subset(P,Q)} \leftarrow \sim \texttt{nonsubset(P,Q)}.$$
$$\texttt{nonsubset(P,Q)} \leftarrow \texttt{P(X)},\ \sim \texttt{Q(X)}.$$

Intuitively, P is a subset of Q if it is not the case that P is a non-subset of Q; and P is a non-subset of Q if there exists some X for which P is true while Q is false.

The syntax we will introduce in Section 3 will allow a more compact notation using λ-expressions as the bodies of clauses (see Example 2 later in the paper).

We would like, for programs such as the above that are higher-order and use negation, to devise a three-valued extensional semantics. The key idea when assigning extensional semantics to *positive* higher-order logic programs (Wadge 1991; Charalambidis et al. 2013) is to interpret the predicate types of the language as monotonic and continuous functions. This is a well-known idea in the area of denotational semantics (Tennent 1991) and is a key assumption for obtaining the least fixpoint semantics for functional programs. This same idea was used in (Wadge 1991; Charalambidis et al. 2013) for obtaining the minimum Herbrand model semantics for positive higher-order logic programs. Unfortunately, this idea breaks down when we consider programs with negation: predicates defined using negation in clause bodies are not-necessarily monotonic. Non-monotonicity means that a higher-order predicate may be true of an input relation, but it may be false for a superset of this relation. For example, consider the predicate p below:

```
p(Q) ← ∼ Q(a).
```

Obviously, p is true of the empty relation $\{\ \}$ but it is false of the relation $\{\mathtt{a}\}$. Notice that the notion of monotonicity we just discussed is usually called *monotonicity with respect to the (standard) truth ordering.*

Fortunately, there is another notion of monotonicity which is obeyed by higher-order logic programs with negation, namely *Fitting-monotonicity* (or *monotonicity with respect to the information ordering*) (Fitting 2002). Consider the program:

```
p(Q) ← ∼ Q(a).
r(a) ← ∼ r(a).
s(a).
```

Under the standard well-founded semantics for classical (first-order) logic programs, the truth value assigned to r(a) is *undefined*; on the other hand, s(a) is *true* in the same semantics. In other words, r corresponds to the 3-valued relation $\{(\mathtt{a}, \mathit{undef})\}$ while s to the relation $\{(\mathtt{a}, \mathit{true})\}$. Fitting-monotonicity intuitively states that if a relation takes as argument a *more defined* relation, then it returns a more defined result. In our case this means that we expect the answer to the query p(r) to be less defined (alternatively, to *have less information*) than the answer to the query p(s) (more specifically, we expect p(r) to be *undefined* and p(s) to be *false*).

Based on the above discussion, we interpret the predicate types of our language as Fitting-monotonic functions. Then, an *interpretation* of a program is a function that assigns Fitting-monotonic functions to the predicates of the program. Given a program P, it is straightforward to define its *immediate consequence operator* Ψ_{P}, which, as usual, takes as input a Herbrand interpretation of the program and returns a new one. It is easy to prove that Ψ_{P} is Fitting-monotonic. It is now tempting to assume that the least fixpoint of Ψ_{P} with respect to the Fitting ordering, is the well-founded model that we are looking for. However, this is not the case: the least fixpoint of Ψ_{P} is minimal with respect to the Fitting (i.e., information) ordering, while the well-founded model should be minimal with respect to the standard truth ordering. In order to get the correct model, we need a few more steps.

We prove that there exists a bijection between Fitting-monotonic functions and pairs of functions of the form (f_1, f_2), where f_1 is monotone-antimonotone, f_2 is antimonotone-monotone, and $f_1 \leq f_2$ (where $\leq$ corresponds to the standard truth ordering). A similar bijection is established between three-valued interpretations and pairs of two-valued-result ones. This bijection allows us to use the powerful tool of approximation fixpoint theory (Denecker et al. 2000; Denecker et al. 2004). In particular, starting from a pair consisting of an underdefined interpretation and an overdefined one, and by iterating an appropriate operator, we demonstrate that we get to a pair of interpretations that is the limit of this sequence. Using our bijection, we show that this limit pair can be converted to a three-valued interpretation $\mathcal{M}_{\mathsf{P}}$ which is a three-valued model of our program P and actually a minimal one with respect to the standard truth ordering. We argue that this is the well-founded semantics of P, because its construction is a generalization of the construction in (Denecker et al. 2004) for the well-founded semantics of classical logic programs.

3 The Syntax of the Higher-Order Language $\mathcal{HOL}$

In this section we introduce $\mathcal{HOL}$, a higher-order language 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 1
A type can either be *functional*, *predicate*, or *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).

The binary operator $\to$ is right-associative. A functional type that is different from ι will often be written in the form $\iota^n \to \iota$, $n \geq 1$ (which stands for $\iota \to \iota \to \cdots \to \iota$ $(n+1)$-times). It can be easily seen that every predicate type π can be written uniquely in the form $\rho_1 \to \cdots \to \rho_n \to o$, $n \geq 0$ (for $n = 0$ we assume that $\pi = o$). We now define the alphabet, the expressions, and the program clauses of $\mathcal{HOL}$:

Definition 2
The *alphabet* of the higher-order language $\mathcal{HOL}$ consists of the following:

1. *Predicate variables* of every predicate type π (denoted by capital letters such as P and Q).
2. *Predicate constants* of every predicate type π (denoted by lowercase letters such as p and q).

3. *Individual variables* of type ι (denoted by capital letters such as X and Y).
4. *Individual constants* of type ι (denoted by lowercase letters such as a and b).
5. *Function symbols* of every functional type $\sigma \neq \iota$ (denoted by lowercase letters such as f and g).
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 $\mathcal{HOL}$ will be called the set of *argument variables* of $\mathcal{HOL}$. Argument variables will be denoted by R.

Definition 3
The set of *expressions* of the higher-order language $\mathcal{HOL}$ 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 $\mathsf{E}_1, \dots, \mathsf{E}_n$ 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 R is an argument variable of type ρ and E is an expression of type π, then $(\lambda\mathsf{R}.\mathsf{E})$ 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 R is a variable of type ρ then $(\exists_\rho \mathsf{R}\,\mathsf{E})$ 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. An expression of type ι will be called a *term*; if it does not contain any individual variables, it will be called a *ground term*.

Definition 4
A *program clause* of $\mathcal{HOL}$ is of the form $\mathsf{p} \leftarrow_\pi \mathsf{E}$ 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 2
We rewrite the program of Example 1 using the syntax of $\mathcal{HOL}$. For every argument type ρ, the `subset` predicate of type $(\rho \to o) \to (\rho \to o) \to o$ takes as arguments two relations of type $\rho \to o$ and returns *true* if the first relation is a subset of the second:

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

The use of λ-expressions obviates the need to have the formal parameters of the predicate in the left-hand side of the definition.

4 The Semantics of the Higher-Order Language $\mathcal{HOL}$

In this section we begin the development of the semantics of the language $\mathcal{HOL}$. We start with the semantics of types, proceed with the semantics of expressions, and then with that of programs. We assume a familiarity with the basic notions of partially ordered sets (see Appendix A for the main definitions).

The semantics of the base boolean domain is three-valued. The semantics of types of the form $\pi_1 \to \pi_2$ is the set of *Fitting-monotonic* functions from the domain of type π_1 to that of type π_2. We define, simultaneously with the meaning of every type τ, two partial orders on the elements of type τ: the relation $\leq_\tau$ which represents the *truth* ordering, and the relation $\preceq_\tau$ which represents the *information* or *Fitting* ordering.

Definition 5
Let D be a nonempty set. For every type τ we define recursively the set of possible meanings of elements of $\mathcal{HOL}$ of type τ, denoted by $[\![\tau]\!]_D$, as follows:

- $[\![o]\!]_D = \{\mathit{false}, \mathit{true}, \mathit{undef}\}$. The partial order $\leq_o$ is the usual one induced by the ordering $\mathit{false} <_o \mathit{undef} <_o \mathit{true}$; the partial order $\preceq_o$ is the one induced by the ordering $\mathit{undef} \prec_o \mathit{false}$ and $\mathit{undef} \prec_o \mathit{true}$.
- $[\![\iota]\!]_D = D$. The partial order $\leq_\iota$ is defined as $d \leq_\iota d$ for all $d \in D$. The partial order $\preceq_\iota$ is also defined as $d \preceq_\iota d$ for all $d \in D$.
- $[\![\iota^n \to \iota]\!]_D = D^n \to D$. No ordering relations are defined for these types.
- $[\![\iota \to \pi]\!]_D = D \to [\![\pi]\!]_D$. The partial order $\leq_{\iota\to\pi}$ is defined as follows: for all $f, g \in [\![\iota \to \pi]\!]_D$, $f \leq_{\iota\to\pi} g$ iff $f(d) \leq_\pi g(d)$ for all $d \in D$. The partial order $\preceq_{\iota\to\pi}$ is defined as follows: for all $f, g \in [\![\iota \to \pi]\!]_D$, $f \preceq_{\iota\to\pi} g$ iff $f(d) \preceq_\pi g(d)$ for all $d \in D$.
- $[\![\pi_1 \to \pi_2]\!]_D = [[\![\pi_1]\!]_D \to [\![\pi_2]\!]_D]$, namely the $\preceq$-monotonic functions[1] from $[\![\pi_1]\!]_D$ to $[\![\pi_2]\!]_D$. The partial order $\leq_{\pi_1\to\pi_2}$ is defined as follows: for all $f, g \in [\![\pi_1 \to \pi_2]\!]_D$, $f \leq_{\pi_1\to\pi_2} g$ iff $f(d) \leq_{\pi_2} g(d)$ for all $d \in [\![\pi_1]\!]_D$. The partial order $\preceq_{\pi_1\to\pi_2}$ is defined as follows: for all $f, g \in [\![\pi_1 \to \pi_2]\!]_D$, $f \preceq_{\pi_1\to\pi_2} g$ iff $f(d) \preceq_{\pi_2} g(d)$ for all $d \in [\![\pi_1]\!]_D$.

The subscripts in the above partial orders will often be omitted when they are obvious from context. For every type π, the set $[\![\pi]\!]_D$ has a least element $\bot_{\leq_\pi}$ and a greatest element $\top_{\leq_\pi}$, called the *bottom* and the *top* elements of $[\![\pi]\!]_D$ with respect to $\leq_\pi$, respectively. In particular, $\bot_{\leq_o} = \mathit{false}$ and $\top_{\leq_o} = \mathit{true}$; $\bot_{\leq_{\iota\to\pi}}(d) = \bot_{\leq_\pi}$ and $\top_{\leq_{\iota\to\pi}}(d) = \top_{\leq_\pi}$, for all $d \in D$; $\bot_{\leq_{\pi_1\to\pi_2}}(d) = \bot_{\leq_{\pi_2}}$ and $\top_{\leq_{\pi_1\to\pi_2}}(d) = \top_{\leq_{\pi_2}}$, for all $d \in [\![\pi_1]\!]_D$. Moreover, for every type π, the set $[\![\pi]\!]_D$ has a least element with respect

[1] Function $f \in [\![\pi_1 \to \pi_2]\!]_D$ is $\preceq$-monotonic if for all $d_1, d_2 \in [\![\pi_1]\!]_D$, $d_1 \preceq_{\pi_1} d_2$ implies $f(d_1) \preceq_{\pi_2} f(d_2)$.

to $\preceq_\pi$, denoted by $\perp_{\preceq_\pi}$ and called the *bottom* element of $[\![\pi]\!]_D$ with respect to $\preceq_\pi$. In particular, $\perp_{\preceq_o}= \mathit{undef}$. The element $\perp_{\preceq_\pi}$ for $\pi \neq o$ can be defined in the obvious way as above. We will simply write $\perp$ to denote the bottom element of any of the above partially ordered sets, when the ordering relation and the specific domain are obvious from context.

We have the following proposition, whose proof is given in Appendix A:

Proposition 1
Let D be a nonempty set. For every predicate type π, $([\![\pi]\!]_D, \leq_\pi)$ is a complete lattice and $([\![\pi]\!]_D, \preceq_\pi)$ is a chain complete poset.

We can now proceed to define the semantics of $\mathcal{HOL}$:

Definition 6
A (three-valued) interpretation $\mathcal{I}$ of $\mathcal{HOL}$ consists of:

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

Definition 7
Let D be a nonempty set. A *state* s of $\mathcal{HOL}$ over D is a function that assigns to each argument variable R of type ρ of $\mathcal{HOL}$, an element $s(\mathsf{R}) \in [\![\rho]\!]_D$.

We define: $\mathit{true}^{-1} = \mathit{false}$, $\mathit{false}^{-1} = \mathit{true}$ and $\mathit{undef}^{-1} = \mathit{undef}$.

Definition 8
Let D be a nonempty set, let $\mathcal{I}$ be an interpretation over D, and let s be a state over D. The semantics of expressions of $\mathcal{HOL}$ with respect to $\mathcal{I}$ and s, is defined as follows:

1. $[\![\mathsf{false}]\!]_s(\mathcal{I}) = \mathit{false}$, and $[\![\mathsf{true}]\!]_s(\mathcal{I}) = \mathit{true}$
2. $[\![\mathsf{c}]\!]_s(\mathcal{I}) = \mathcal{I}(\mathsf{c})$, for every individual constant c
3. $[\![\mathsf{p}]\!]_s(\mathcal{I}) = \mathcal{I}(\mathsf{p})$, for every predicate constant p
4. $[\![\mathsf{R}]\!]_s(\mathcal{I}) = s(\mathsf{R})$, for every argument variable R
5. $[\![(\mathsf{f}\ \mathsf{E}_1 \cdots \mathsf{E}_n)]\!]_s(\mathcal{I}) = \mathcal{I}(\mathsf{f})\ [\![\mathsf{E}_1]\!]_s(\mathcal{I}) \cdots [\![\mathsf{E}_n]\!]_s(\mathcal{I})$, for every n-ary function symbol f
6. $[\![(\mathsf{E}_1\mathsf{E}_2)]\!]_s(\mathcal{I}) = [\![\mathsf{E}_1]\!]_s(\mathcal{I})([\![\mathsf{E}_2]\!]_s(\mathcal{I}))$
7. $[\![(\lambda\mathsf{R}.\mathsf{E})]\!]_s(\mathcal{I}) = \lambda d.[\![\mathsf{E}]\!]_{s[\mathsf{R}/d]}(\mathcal{I})$, where if $\mathsf{R} : \rho$ then d ranges over $[\![\rho]\!]_D$
8. $[\![(\mathsf{E}_1 \bigvee_\pi \mathsf{E}_2)]\!]_s(\mathcal{I}) = \bigvee_{\leq_\pi}\{[\![\mathsf{E}_1]\!]_s(\mathcal{I}), [\![\mathsf{E}_2]\!]_s(\mathcal{I})\}$
9. $[\![(\mathsf{E}_1 \bigwedge_\pi \mathsf{E}_2)]\!]_s(\mathcal{I}) = \bigwedge_{\leq_\pi}\{[\![\mathsf{E}_1]\!]_s(\mathcal{I}), [\![\mathsf{E}_2]\!]_s(\mathcal{I})\}$
10. $[\![(\sim\!\mathsf{E})]\!]_s(\mathcal{I}) = ([\![\mathsf{E}]\!]_s(\mathcal{I}))^{-1}$
11. $[\![(\mathsf{E}_1 \approx \mathsf{E}_2)]\!]_s(\mathcal{I}) = \begin{cases} \mathit{true}, & \text{if } [\![\mathsf{E}_1]\!]_s(\mathcal{I}) = [\![\mathsf{E}_2]\!]_s(\mathcal{I}) \\ \mathit{false}, & \text{otherwise} \end{cases}$
12. $[\![(\exists_\rho \mathsf{R}\, \mathsf{E})]\!]_s(\mathcal{I}) = \bigvee_{\leq_o}\{[\![\mathsf{E}]\!]_{s[\mathsf{R}/d]}(\mathcal{I}) \mid d \in [\![\rho]\!]_D\}$

For closed expressions E we will often write $[\![\mathsf{E}]\!](\mathcal{I})$ instead of $[\![\mathsf{E}]\!]_s(\mathcal{I})$ (since, in this case, the meaning of E is independent of s). The following lemma demonstrates that our semantic valuation function returns elements that belong to the appropriate domain (the proof of the lemma by structural induction on E, is easy and omitted).

Lemma 1
Let $\mathsf{E} : \rho$ be an expression and let D be a nonempty set. Moreover, let s be a state over D and let $\mathcal{I}$ be an interpretation over D. Then, $[\![\mathsf{E}]\!]_s(\mathcal{I}) \in [\![\rho]\!]_D$.

Finally, we define the notion of *model* for $\mathcal{HOL}$ programs:

Definition 9
Let P be a $\mathcal{HOL}$ program and let M be an interpretation of P. Then M will be called a *model* of P iff for all clauses $\mathsf{p} \leftarrow_\pi \mathsf{E}$ of P, it holds $[\![\mathsf{E}]\!](M) \leq_\pi M(\mathsf{p})$.

5 An Alternative View of Fitting-Monotonic Functions

In this section we demonstrate that every Fitting-monotonic function f can be equivalently represented as a pair of functions (f_1, f_2), where f_1 is monotone-antimonotone, f_2 is antimonotone-monotone and $f_1 \leq f_2$. Consider for example a function f of type $o \to o$, i.e., $f : \{\mathit{true}, \mathit{false}, \mathit{undef}\} \to \{\mathit{true}, \mathit{false}, \mathit{undef}\}$. One can view the truth values as pairs where *true* corresponds to $(\mathit{true}, \mathit{true})$, *false* corresponds to $(\mathit{false}, \mathit{false})$, and *undef* corresponds to $(\mathit{false}, \mathit{true})$. Therefore, f can also equivalently be seen as a function f' that takes pairs and returns pairs. We can then "break" f' into two components f_1 and f_2 where f_1 returns the first element of the pair that f' returns while f_2 returns the second. The monotone-antimonotone and antimonotone-monotone requirements ensure that the pair (f_1, f_2) retains the property of Fitting-monotonicity of the original function f. These ideas can be generalized to arbitrary types. The formal details of this equivalence are described below. The following definitions will be used:

Definition 10
Let L_1, L_2 be sets and let $\leq$ be a partial order on $L_1 \cup L_2$. We define: $L_1 \otimes_\leq L_2 = \{(x, y) \in L_1 \times L_2 : x \leq y\}$.

We will omit the $\leq$ from $\otimes_\leq$ when it is obvious from context.

Definition 11
Let L_1, L_2 be sets and let $\leq$ be a partial order on $L_1 \cup L_2$. Also, let $(A, \leq_A)$ be a partially ordered set. A function $f : (L_1 \otimes L_2) \to A$ will be called *monotone-antimonotone* (respectively *antimonotone-monotone*) if for all $(x, y), (x', y') \in L_1 \otimes L_2$ with $x \leq x'$ and $y' \leq y$, it holds that $f(x, y) \leq_A f(x', y')$ (respectively $f(x', y') \leq_A f(x, y)$). We denote by $[(L_1 \otimes L_2) \overset{\mathsf{ma}}{\to} A]$ the set of functions that are monotone-antimonotone and by $[(L_1 \otimes L_2) \overset{\mathsf{am}}{\to} A]$ those that are antimonotone-monotone.

In order to establish the bijection between Fitting-monotonic functions and pairs of monotone-antimonotone and antimonotone-monotone functions, we reinterpret the predicate types of $\mathcal{HOL}$ in an alternative way.

Definition 12
Let D be a nonempty set. For every type τ we define the monotone-antimonotone and the antimonotone-monotone meanings of the elements of type τ with respect to D, denoted respectively by $[\![\tau]\!]_D^{\mathsf{ma}}$ and $[\![\tau]\!]_D^{\mathsf{am}}$. At the same time we define a partial order $\leq_\tau$ between the elements of $[\![\tau]\!]_D^{\mathsf{ma}} \cup [\![\tau]\!]_D^{\mathsf{am}}$.

- $[\![o]\!]_D^{\mathsf{ma}} = [\![o]\!]_D^{\mathsf{am}} = \{\mathit{false}, \mathit{true}\}$. The partial order $\leq_o$ is the usual one induced by the ordering $\mathit{false} \leq_o \mathit{true}$.

- $[\![\iota]\!]^{\mathsf{ma}} = [\![\iota]\!]^{\mathsf{am}} = D$. The partial order $\leq_\iota$ is defined as $d \leq_\iota d$, for all $d \in D$.
- $[\![\iota^n \to \iota]\!]^{\mathsf{ma}} = [\![\iota^n \to \iota]\!]^{\mathsf{am}} = D^n \to D$. There is no partial order for elements of type $\iota^n \to \iota$.
- $[\![\iota \to \pi]\!]_D^{\mathsf{ma}} = D \to [\![\pi]\!]_D^{\mathsf{ma}}$ and $[\![\iota \to \pi]\!]_D^{\mathsf{am}} = D \to [\![\pi]\!]_D^{\mathsf{am}}$. The partial order $\leq_{\iota\to\pi}$ is defined as follows: for all $f, g \in [\![\iota \to \pi]\!]_D^{\mathsf{ma}} \cup [\![\iota \to \pi]\!]_D^{\mathsf{am}}$, $f \leq_{\iota\to\pi} g$ iff $f(d) \leq_\pi g(d)$ for all $d \in D$.
- $[\![\pi_1 \to \pi_2]\!]_D^{\mathsf{ma}} = [([\![\pi_1]\!]_D^{\mathsf{ma}} \otimes [\![\pi_1]\!]_D^{\mathsf{am}}) \overset{\mathsf{ma}}{\to} [\![\pi_2]\!]_D^{\mathsf{ma}}]$, and $[\![\pi_1 \to \pi_2]\!]_D^{\mathsf{am}} = [([\![\pi_1]\!]_D^{\mathsf{ma}} \otimes [\![\pi_1]\!]_D^{\mathsf{am}}) \overset{\mathsf{am}}{\to} [\![\pi_2]\!]_D^{\mathsf{am}}]$. The relation $\leq_{\pi_1\to\pi_2}$ is the partial order defined as follows: for all $f, g \in [\![\pi_1 \to \pi_2]\!]_D^{\mathsf{ma}} \cup [\![\pi_1 \to \pi_2]\!]_D^{\mathsf{am}}$, $f \leq_{\pi_1\to\pi_2} g$ iff $f(d_1, d_2) \leq_{\pi_2} g(d_1, d_2)$ for all $(d_1, d_2) \in [\![\pi_1]\!]_D^{\mathsf{ma}} \otimes [\![\pi_1]\!]_D^{\mathsf{am}}$.

For every π, the bottom and top elements of $[\![\pi]\!]_D^{\mathsf{ma}}$ and $[\![\pi]\!]_D^{\mathsf{am}}$ can be defined in the obvious way. We have the following proposition (see Appendix B for the proof):

Proposition 2
Let D be a nonempty set. For every predicate type π, $([\![\pi]\!]_D^{\mathsf{ma}}, \leq_\pi)$ and $([\![\pi]\!]_D^{\mathsf{am}}, \leq_\pi)$ are complete lattices.

We extend, in a pointwise way, our orderings to apply to pairs. For simplicity, we overload our notation and use the same symbols $\leq$ and $\preceq$ for the new orderings.

Definition 13
Let D be a nonempty set and let π be a predicate type. We define the relations $\leq_\pi$ and $\preceq_\pi$, so that for all $(x, y), (x', y') \in [\![\pi]\!]_D^{\mathsf{ma}} \otimes [\![\pi]\!]_D^{\mathsf{am}}$:

- $(x, y) \leq_\pi (x', y')$ iff $x \leq_\pi x'$ and $y \leq_\pi y'$.
- $(x, y) \preceq_\pi (x', y')$ iff $x \leq_\pi x'$ and $y' \leq_\pi y$.

The following proposition is demonstrated in Appendix B:

Proposition 3
Let D be a nonempty set. For each predicate type π, $[\![\pi]\!]_D^{\mathsf{ma}} \otimes [\![\pi]\!]_D^{\mathsf{am}}$ is a complete lattice with respect to $\leq_\pi$ and a chain-complete poset with respect to $\preceq_\pi$.

In the rest of the paper we will denote the *first* and *second* selection functions on pairs with the more compact notation $[\cdot]_1$ and $[\cdot]_2$: given any pair (x, y), it is $[(x, y)]_1 = x$ and $[(x, y)]_2 = y$. We can now establish the bijection between $[\![\pi]\!]_D$ and $[\![\pi]\!]_D^{\mathsf{ma}} \otimes [\![\pi]\!]_D^{\mathsf{am}}$. The following definition and two propositions (whose proofs are given in Appendix B), explain how.

Definition 14
Let D be a nonempty set. For every predicate type π, we define recursively the functions $\tau_\pi : [\![\pi]\!]_D \to ([\![\pi]\!]_D^{\mathsf{ma}} \otimes [\![\pi]\!]_D^{\mathsf{am}})$ and $\tau_\pi^{-1} : ([\![\pi]\!]_D^{\mathsf{ma}} \otimes [\![\pi]\!]_D^{\mathsf{am}}) \to [\![\pi]\!]_D$, as follows.

- $\tau_o(\mathit{false}) = (\mathit{false}, \mathit{false})$, $\tau_o(\mathit{true}) = (\mathit{true}, \mathit{true})$, $\tau_o(\mathit{undef}) = (\mathit{false}, \mathit{true})$
- $\tau_{\iota\to\pi}(f) = (\lambda d.[\tau_\pi(f(d))]_1, \lambda d.[\tau_\pi(f(d))]_2)$
- $\tau_{\pi_1\to\pi_2}(f) = (\lambda(d_1, d_2).[\tau_{\pi_2}(f(\tau_{\pi_1}^{-1}(d_1, d_2)))]_1, \lambda(d_1, d_2).[\tau_{\pi_2}(f(\tau_{\pi_1}^{-1}(d_1, d_2)))]_2)$

and

- $\tau_o^{-1}(\mathit{false}, \mathit{false}) = \mathit{false}$, $\tau_o^{-1}(\mathit{true}, \mathit{true}) = \mathit{true}$, $\tau_o^{-1}(\mathit{false}, \mathit{true}) = \mathit{undef}$
- $\tau_{\iota\to\pi}^{-1}(f_1, f_2) = \lambda d.\tau_\pi^{-1}(f_1(d), f_2(d))$
- $\tau_{\pi_1\to\pi_2}^{-1}(f_1, f_2) = \lambda d.\tau_{\pi_2}^{-1}(f_1(\tau_{\pi_1}(d)), f_2(\tau_{\pi_1}(d)))$.

Proposition 4
Let D be a nonempty set and let π be a predicate type. Then, for every $f, g \in [\![\pi]\!]_D$ and for every $(f_1, f_2), (g_1, g_2) \in [\![\pi]\!]_D^{\mathsf{ma}} \otimes [\![\pi]\!]_D^{\mathsf{am}}$, the following statements hold:

1. $\tau_\pi(f) \in ([\![\pi]\!]_D^{\mathsf{ma}} \otimes [\![\pi]\!]_D^{\mathsf{am}})$ and $\tau_\pi^{-1}(f_1, f_2) \in [\![\pi]\!]_D$.
2. If $f \preceq_\pi g$ then $\tau_\pi(f) \preceq_\pi \tau_\pi(g)$.
3. If $f \leq_\pi g$ then $\tau_\pi(f) \leq_\pi \tau_\pi(g)$.
4. If $(f_1, f_2) \preceq_\pi (g_1, g_2)$ then $\tau_\pi^{-1}(f_1, f_2) \preceq_\pi \tau_\pi^{-1}(g_1, g_2)$.
5. If $(f_1, f_2) \leq_\pi (g_1, g_2)$ then $\tau_\pi^{-1}(f_1, f_2) \leq_\pi \tau_\pi^{-1}(g_1, g_2)$.

Proposition 5
Let D be a nonempty set and let π be a predicate type. Then, for every $f \in [\![\pi]\!]_D$, $\tau_\pi^{-1}(\tau_\pi(f)) = f$, and for every $(f_1, f_2) \in [\![\pi]\!]_D^{\mathsf{ma}} \otimes [\![\pi]\!]_D^{\mathsf{am}}$, $\tau_\pi(\tau_\pi^{-1}(f_1, f_2)) = (f_1, f_2)$.

6 The Well-Founded Semantics for $\mathcal{HOL}$ Programs

In this section we demonstrate that every program of $\mathcal{HOL}$ has a distinguished *minimal Herbrand model* which can be obtained by an iterative procedure. This construction generalizes the familiar well-founded semantics. Our main results are based on a mild generalization of the consistent approximation fixpoint theory of (Denecker et al. 2004). 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 ground terms that can be formed out of the individual constants[2] and the function symbols of P.

Definition 16
A (three-valued) *Herbrand interpretation* $\mathcal{I}$ of a program P is an interpretation such that:

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

We denote the set of all three-valued Herbrand interpretations of a program P by $\mathcal{H}_{\mathsf{P}}$. A *Herbrand state* of P is a state whose underlying domain is U_{P}. A *Herbrand model* of P is a Herbrand interpretation that is a model of P. The truth and the information orderings easily extend to Herbrand interpretations:

Definition 17
Let P be a program. We define the partial orders $\leq$ and $\preceq$ on $\mathcal{H}_{\mathsf{P}}$ as follows: for all $\mathcal{I}, \mathcal{J} \in \mathcal{H}_{\mathsf{P}}$, $\mathcal{I} \leq \mathcal{J}$ (respectively, $\mathcal{I} \preceq \mathcal{J}$) iff for every predicate type π and for every predicate constant $\mathsf{p} : \pi$ of P, $\mathcal{I}(\mathsf{p}) \leq_\pi \mathcal{J}(\mathsf{p})$ (respectively, $\mathcal{I}(\mathsf{p}) \preceq_\pi \mathcal{J}(\mathsf{p})$).

The proof of the following proposition is analogous to that of Proposition 1 and omitted:

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

Proposition 6
Let P be a program. Then, $(\mathcal{H}_{\mathsf{P}}, \leq)$ is a complete lattice and $(\mathcal{H}_{\mathsf{P}}, \preceq)$ is a chain complete poset.

The following lemma is also easy to establish, and its proof is omitted:

Lemma 2
Let P be a program, let $\mathcal{I}, \mathcal{J} \in \mathcal{H}_{\mathsf{P}}$, and let s be a Herbrand state of P. For every expression E, if $\mathcal{I} \preceq \mathcal{J}$ then $[\![\mathsf{E}]\!]_s(\mathcal{I}) \preceq [\![\mathsf{E}]\!]_s(\mathcal{J})$.

The bijection established in Section 5 extends also to interpretations. More specifically, every three-valued Herbrand interpretation $\mathcal{I}$ of a program P can be mapped by (an extension of) τ to a pair of functions (I, J) such that:

- for every individual constant c of P, $I(\mathsf{c}) = J(\mathsf{c}) = \mathsf{c}$;
- for every predicate constant $\mathsf{p} : \pi$ of P, $I(\mathsf{p}) \in [\![\pi]\!]^{\mathsf{ma}}_{U_{\mathsf{P}}}$ and $J(\mathsf{p}) \in [\![\pi]\!]^{\mathsf{am}}_{U_{\mathsf{P}}}$;
- for every n-ary function symbol f of P and for all $\mathsf{t}_1, \ldots, \mathsf{t}_n \in U_{\mathsf{P}}$, $I(\mathsf{f})\,\mathsf{t}_1 \cdots \mathsf{t}_n = J(\mathsf{f})\,\mathsf{t}_1 \cdots \mathsf{t}_n = \mathsf{f}\,\mathsf{t}_1 \cdots \mathsf{t}_n$.

Functions of the form I above will be called "*monotone-antimonotone Herbrand interpretations*" and functions of the form J will be called "*antimonotone-monotone Herbrand interpretations*". We will denote by $\mathcal{H}^{\mathsf{ma}}_{\mathsf{P}}$ the set of functions of the former type and by $\mathcal{H}^{\mathsf{am}}_{\mathsf{P}}$ those of the latter type. As in Definition 17, we can define a partial order $\leq$ on $\mathcal{H}^{\mathsf{ma}}_{\mathsf{P}} \cup \mathcal{H}^{\mathsf{am}}_{\mathsf{P}}$. Similarly, as in Definition 13, we can define partial orders $\leq$ and $\preceq$ on $\mathcal{H}^{\mathsf{ma}}_{\mathsf{P}} \otimes \mathcal{H}^{\mathsf{am}}_{\mathsf{P}}$. The proof of the following proposition is a direct consequence of the proofs of Propositions 3 and 2, and therefore omitted.

Proposition 7
Let P be a program. Then, $(\mathcal{H}^{\mathsf{ma}}_{\mathsf{P}}, \leq)$ and $(\mathcal{H}^{\mathsf{am}}_{\mathsf{P}}, \leq)$ are complete lattices having the same $\perp$ and $\top$ elements. Moreover, $(\mathcal{H}^{\mathsf{ma}}_{\mathsf{P}} \otimes \mathcal{H}^{\mathsf{am}}_{\mathsf{P}}, \leq)$ is a complete lattice and $(\mathcal{H}^{\mathsf{ma}}_{\mathsf{P}} \otimes \mathcal{H}^{\mathsf{am}}_{\mathsf{P}}, \preceq)$ is a chain-complete poset.

The bijection between $\mathcal{H}_{\mathsf{P}}$ and $\mathcal{H}^{\mathsf{ma}}_{\mathsf{P}} \otimes \mathcal{H}^{\mathsf{am}}_{\mathsf{P}}$ can be explained more formally as follows. Given $\mathcal{I} \in \mathcal{H}_{\mathsf{P}}$, we define $\tau(\mathcal{I}) = (I, J)$, where for every predicate constant $\mathsf{p} : \pi$ it holds $I(\mathsf{p}) = [\tau_\pi(\mathcal{I}(\mathsf{p}))]_1$ and $J(\mathsf{p}) = [\tau_\pi(\mathcal{I}(\mathsf{p}))]_2$. Conversely, given a pair $(I, J) \in \mathcal{H}^{\mathsf{ma}}_{\mathsf{P}} \otimes \mathcal{H}^{\mathsf{am}}_{\mathsf{P}}$, we define the three-valued Herbrand interpretation $\mathcal{I}$ as follows: $\mathcal{I}(\mathsf{p}) = \tau_\pi^{-1}(I(\mathsf{p}), J(\mathsf{p}))$. We now define the three-valued and two-valued immediate consequence operators:

Definition 18
Let P be a program. The *three-valued immediate consequence operator* $\Psi_{\mathsf{P}} : \mathcal{H}_{\mathsf{P}} \to \mathcal{H}_{\mathsf{P}}$ of P is defined for every $\mathsf{p} : \pi$ as: $\Psi_{\mathsf{P}}(\mathcal{I})(\mathsf{p}) = \bigvee_{\leq_\pi} \{[\![\mathsf{E}]\!](\mathcal{I}) \mid (\mathsf{p} \leftarrow_\pi \mathsf{E}) \in \mathsf{P}\}$.

Definition 19
Let P be a program. The *two-valued immediate consequence operator* $T_{\mathsf{P}} : (\mathcal{H}^{\mathsf{ma}}_{\mathsf{P}} \otimes \mathcal{H}^{\mathsf{am}}_{\mathsf{P}}) \to (\mathcal{H}^{\mathsf{ma}}_{\mathsf{P}} \otimes \mathcal{H}^{\mathsf{am}}_{\mathsf{P}})$ of P is defined as: $T_{\mathsf{P}}(I, J) = \tau(\Psi_{\mathsf{P}}(\tau^{-1}(I, J)))$.

From Proposition 14 in Appendix D it follows that T_{P} is well-defined. Moreover, it is Fitting-monotonic as the following lemma demonstrates (see Appendix D for the proof):

Lemma 3
Let P be a program and let $(I_1, J_1), (I_2, J_2) \in \mathcal{H}_{\mathsf{P}}^{\mathsf{ma}} \otimes \mathcal{H}_{\mathsf{P}}^{\mathsf{am}}$. If $(I_1, J_1) \preceq (I_2, J_2)$ then $\mathcal{T}_{\mathsf{P}}(I_1, J_1) \preceq \mathcal{T}_{\mathsf{P}}(I_2, J_2)$.

We will use $\mathcal{T}_{\mathsf{P}}$ to construct the well-founded model of program P. Our construction is based on a mild extension of consistent approximation fixpoint theory (Denecker et al. 2004). Therefore, in order for the following two definitions and subsequent theorem to be fully comprehended, it would be helpful if the reader had some familiarity with the material in (Denecker et al. 2004).

Definition 20
Let P be a program and let $(I, J) \in \mathcal{H}_{\mathsf{P}}^{\mathsf{ma}} \otimes \mathcal{H}_{\mathsf{P}}^{\mathsf{am}}$. Assume that $(I, J) \preceq \mathcal{T}_{\mathsf{P}}(I, J)$. We define $I^{\uparrow} = \mathit{lfp}([\mathcal{T}_{\mathsf{P}}(I, \cdot)]_2)$ and $J^{\downarrow} = \mathit{lfp}([\mathcal{T}_{\mathsf{P}}(\cdot, J)]_1)$, where by $\mathcal{T}_{\mathsf{P}}(\cdot, J)$ we denote the function $f(x) = \mathcal{T}_{\mathsf{P}}(x, J)$ and by $\mathcal{T}_{\mathsf{P}}(I, \cdot)$ the function $g(x) = \mathcal{T}_{\mathsf{P}}(I, x)$.

It can be shown (see Appendix C) that $I^{\uparrow}$ and $J^{\downarrow}$ are well-defined, and this is due to the crucial assumption $(I, J) \preceq \mathcal{T}_{\mathsf{P}}(I, J)$. This property was introduced in (Denecker et al. 2004) where it is named A-reliability (in our case A is the $\mathcal{T}_{\mathsf{P}}$ operator). Before proceeding to the definition of the well-founded semantics, we need to define one more operator, namely the *stable revision operator* (see (Denecker et al. 2004)[page 91] for the intuition and motivation behind this operator).

Definition 21
Let P be a program. We define the function $\mathcal{C}_{\mathcal{T}_{\mathsf{P}}}$ which for every pair $(I, J) \in \mathcal{H}_{\mathsf{P}}^{\mathsf{ma}} \otimes \mathcal{H}_{\mathsf{P}}^{\mathsf{am}}$ with $(I, J) \preceq \mathcal{T}_{\mathsf{P}}(I, J)$, returns the pair $(J^{\downarrow}, I^{\uparrow})$:

$$\mathcal{C}_{\mathcal{T}_{\mathsf{P}}}(I, J) = (J^{\downarrow}, I^{\uparrow}) = (\mathit{lfp}([\mathcal{T}_{\mathsf{P}}(\cdot, J)]_1), \mathit{lfp}([\mathcal{T}_{\mathsf{P}}(I, \cdot)]_2))$$

The function $\mathcal{C}_{\mathcal{T}_{\mathsf{P}}}$ will be called the *stable revision operator* for $\mathcal{T}_{\mathsf{P}}$.

The following theorem is a direct consequence of Theorem 5 given in Appendix C (which extends Theorem 3.11 in (Denecker et al. 2004) to our case):

Theorem 1
Let P be a program. We define the following sequence of pairs of interpretations:

$$\begin{array}{lll} (I_0, J_0) & = & (\bot, \top) \\ (I_{\lambda+1}, J_{\lambda+1}) & = & \mathcal{C}_{\mathcal{T}_{\mathsf{P}}}(I_\lambda, J_\lambda) \\ (I_\lambda, J_\lambda) & = & \bigvee_{\preceq} \{(I_\kappa, J_\kappa) \mid \kappa < \lambda\} \quad \text{for limit ordinals } \lambda \end{array}$$

Then, the above sequence of pairs of interpretations is well-defined. Moreover, there exists a least ordinal δ such that $(I_\delta, J_\delta) = \mathcal{C}_{\mathcal{T}_{\mathsf{P}}}(I_\delta, J_\delta)$ and $(I_\delta, J_\delta) \in \mathcal{H}_{\mathsf{P}}^{\mathsf{ma}} \otimes \mathcal{H}_{\mathsf{P}}^{\mathsf{am}}$.

In the following, we will denote with $\mathcal{M}_{\mathsf{P}}$ the interpretation $\tau^{-1}(I_\delta, J_\delta)$. The following two lemmas demonstrate that the pre-fixpoints of $\mathcal{T}_{\mathsf{P}}$ correspond exactly to the three-valued models of P (see Appendix D for the corresponding proofs).

Lemma 4
Let P be a program. If $(I, J) \in \mathcal{H}_{\mathsf{P}}^{\mathsf{ma}} \otimes \mathcal{H}_{\mathsf{P}}^{\mathsf{am}}$ is a pre-fixpoint of $\mathcal{T}_{\mathsf{P}}$ then $\tau^{-1}(I, J)$ is a model of P.

Lemma 5
Let $\mathcal{M} \in \mathcal{H}_{\mathsf{P}}$ be a model of P. Then, $\tau(\mathcal{M})$ is a pre-fixpoint of T_{P}.

Finally, the following two lemmas (see Appendix D for the proofs), provide evidence that $\mathcal{M}_{\mathsf{P}}$ is an extension of the classical well-founded semantics to the higher-order case:

Theorem 2
Let P be a program. Then, $\mathcal{M}_{\mathsf{P}}$ is a $\leq$-minimal model of P.

Theorem 3
For every propositional program P, $\mathcal{M}_{\mathsf{P}}$ coincides with the well-founded model of P.

In Appendix E we give an example construction of $\mathcal{M}_{\mathsf{P}}$ for a given program P.

7 Related and Future Work

In this section we compare our technique with the existing proposals for assigning semantics to higher-order logic programs with negation and we discuss possibly fruitful directions for future research.

The proposed extensional three-valued approach has important differences from the existing alternative ones, namely (Charalambidis et al. 2014), (Rondogiannis and Symeonidou 2016) and (Rondogiannis and Symeonidou 2017). As already mentioned in the introduction section, the technique in (Rondogiannis and Symeonidou 2017) is not extensional in the general case (it is however extensional if the source higher-order programs are *stratified* - see (Rondogiannis and Symeonidou 2017) for the formal definition of this notion). In this respect, the present approach is more general since it assigns an extensional semantics to *all* the programs of $\mathcal{HOL}$.

On the other hand, both of the techniques (Charalambidis et al. 2014) and (Rondogiannis and Symeonidou 2016) rely on an infinite-valued logic, and give a very fine-grained semantics to programs. This fine-grained nature of the infinite-valued approach makes it very appealing from a mathematical point of view. As it was recently demonstrated in (Ésik 2015; Carayol and Ésik 2016), in the case of first-order logic programs the infinite-valued approach satisfies all identities of *iteration theories* (Bloom and Ésik 1993), while the well-founded semantics does not. Since iteration theories provide an abstract framework for the evaluation of the merits of various semantic approaches for languages that involve recursion, these results appear to suggest that the infinite-valued approach has advantages from a mathematical point of view. On the other hand, the well-founded semantics is based on a much simpler three-valued logic, it is widely known to the logic programming community, and it has been studied and used for almost three decades. It is important however to emphasize that the differences between the infinite-valued and the well-founded approaches are not only a matter of mathematical elegance. In many programs, the two techniques behave differently. For example, given the program:

$$\mathtt{p} \leftarrow \sim (\sim \mathtt{p})$$

the approaches in (Charalambidis et al. 2014) and (Rondogiannis and Symeonidou 2016) will produce the model $\{(\mathtt{p}, \mathit{undef})\}$, while our present approach will produce the model $\{(\mathtt{p}, \mathit{false})\}$. In essence, our present approach *cancels such nested negations* (see also the discussion in (Denecker et al. 2012)[page 185, Example 1] on this issue), while the approaches in (Charalambidis et al. 2014) and (Rondogiannis and Symeonidou 2016) assign the value *undef* due to the circular dependence of `p` on itself through negation.

Similarly, for the following program (taken from (Rondogiannis and Symeonidou 2017)):

$$
\begin{aligned}
&\texttt{s} \leftarrow \lambda\texttt{Q}.\texttt{Q}\ (\texttt{s}\ \texttt{Q})\\
&\texttt{p} \leftarrow \lambda\texttt{R}.\texttt{R}\\
&\texttt{q} \leftarrow \lambda\texttt{R}.\sim (\texttt{w}\ \texttt{R})\\
&\texttt{w} \leftarrow \lambda\texttt{R}.(\sim \texttt{R})
\end{aligned}
$$

the infinite-valued approaches will return the value *false* for the query `(s p)` and *undef* for `(s q)`, while our present approach will return the value *false* for both queries.

It is an interesting topic for future research to identify large classes of programs where the infinite-valued approach and the present one coincide. Possibly a good candidate for such a comparison would be the class of stratified higher-order logic programs (Rondogiannis and Symeonidou 2016). More generally, we believe that an investigation of the connections between the well-founded semantics and the infinite-valued one, will be quite rewarding.

Another interesting direction for future research would be to consider other possible semantics that can be revealed using approximation fixpoint theory. It is well-known that for first-order logic programs, approximation fixpoint theory can be used in order to define other useful fixpoints such as *stable*, *Kripke-Kleene*, and *supported* ones. We argue that using the approach proposed in this paper, this can also be done for higher-order logic programs. In particular, as in the first-order case, the fixpoints of T_{P} correspond to *3-valued supported models of* P (recall that by Lemma 4 every fixpoint of T_{P} is a model of P). Moreover, since T_{P} is Fitting-monotonic over $\mathcal{H}_{\mathsf{P}}^{\mathsf{ma}} \otimes \mathcal{H}_{\mathsf{P}}^{\mathsf{am}}$ (which by Proposition 7 is a chain-complete poset), it has a least fixpoint which we can take as the *Kripke-Kleene fixpoint of* T_{P}. Finally, as in the case of first-order logic programs, the set of all fixpoints of $\mathcal{C}_{T_{\mathsf{P}}}$ is the set of *stable fixpoints of* T_{P}, and can be taken as the 3-valued stable models of P (by Theorem 6 in Appendix C, every fixpoint of $\mathcal{C}_{T_{\mathsf{P}}}$ is also a fixpoint of T_{P} and therefore a model of P).

In contrast to the above 3-valued semantics, the definition of *2-valued stable models* for higher-order logic programs seems less direct to obtain. In the case of first-order logic programs, the 2-valued stable models are those fixpoints of $\mathcal{C}_{T_{\mathsf{P}}}$ that are *exact* (Denecker et al. 2000; Denecker et al. 2004), i.e., that are of the form (I, I). In the higher-order case however, things are not that simple. Consider for example the positive higher-order logic program consisting only of the rule `p(R) ← R`, where `p` is of type $o \to o$. Since this is a positive program, it is reasonable to assume that it has a unique 2-valued stable model which assigns to `p` the identity relation over the set of classical two truth values. The meaning of this program under the semantics proposed in the present paper is captured by the pair of interpretations (I, J) where: $I(\texttt{p})(\mathit{false}, \mathit{false}) = \mathit{false}$, $I(\texttt{p})(\mathit{true}, \mathit{true}) = \mathit{true}$, $I(\texttt{p})(\mathit{false}, \mathit{true}) = \mathit{false}$, and $J(\texttt{p})(\mathit{false}, \mathit{false}) = \mathit{false}$, $J(\texttt{p})(\mathit{true}, \mathit{true}) = \mathit{true}$, $J(\texttt{p})(\mathit{false}, \mathit{true}) = \mathit{true}$. Notice that $I \neq J$ and this is due to the fact that I and J are 3-valued interpretations and not 2-valued ones as in the first-order case. In other words, under our semantics there does not exist an exact pair of interpretations that is a fixpoint of $\mathcal{C}_{T_{\mathsf{P}}}$ which we could take as the 2-valued stable semantics of the program. What needs to be done here is to generalize the notion of "*exact pair of interpretations*". Informally speaking, a pair (I, J) of Herbrand interpretations of P will be called exact if for every predicate constant p of the program, $I(\mathsf{p})$ coincides with $J(\mathsf{p})$ when they are applied to arguments that are *essentially* 2*-valued* (we need to define inductively for all types what it means for a relation to be essentially 2-valued). Notice that $I(\mathsf{p})$ agrees with $J(\mathsf{p})$ when applied to 2-valued arguments, i.e., when applied to $(\mathit{true}, \mathit{true})$ and $(\mathit{false}, \mathit{false})$. We believe that the approach sketched above leads to a characterization of the 2-valued stable models, but the details need to be carefully examined and specified.

In this paper we have claimed that the proposed approach is an appealing formulation for capturing the well-founded semantics for higher-order logic programs with negation. We have substantiated our claim by demonstrating that the proposed semantics generalizes the well-founded one for propositional programs. As suggested by one of the reviewers, this claim would be stronger if one could define alternative semantics that lead to the same model. One such approach would be to extend the original definition of the well-founded semantics (Gelder et al. 1991) which was based on the notion of *unfounded sets*. Another promising direction would be to derive an extension of Przymusinski's *iterated least fixpoint construction* (Przymusinski 1989) to the higher-order case. Both of these directions seem quite fruitful and non-trivial, and certainly require further investigation.

References

Bezem, M. 1999. Extensionality of simply typed logic programs. In *Logic Programming: The 1999 International Conference, Las Cruces, New Mexico, USA, November 29 - December 4, 1999*, D. D. Schreye, Ed. MIT Press, 395–410.

Bloom, S. L. and Ésik, Z. 1993. *Iteration Theories - The Equational Logic of Iterative Processes.* EATCS Monographs on Theoretical Computer Science. Springer.

Carayol, A. and Ésik, Z. 2016. An analysis of the equational properties of the well-founded fixed point. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016.*, C. Baral, J. P. Delgrande, and F. Wolter, Eds. AAAI Press, 533–536.

Charalambidis, A., Ésik, Z., and Rondogiannis, P. 2014. Minimum model semantics for extensional higher-order logic programming with negation. *TPLP 14,* 4-5, 725–737.

Charalambidis, A., Handjopoulos, K., Rondogiannis, P., and Wadge, W. W. 2013. Extensional higher-order logic programming. *ACM Trans. Comput. Log. 14,* 3, 21.

Charalambidis, A., Rondogiannis, P., and Symeonidou, I. 2017. Equivalence of two fixed-point semantics for definitional higher-order logic programs. *Theor. Comput. Sci. 668*, 27–42.

Davey, B. A. and Priestley, H. A. 2002. *Introduction to Lattices and Order.* Cambridge University Press.

Denecker, M., Bruynooghe, M., and Vennekens, J. 2012. Approximation fixpoint theory and the semantics of logic and answers set programs. In *Correct Reasoning - Essays on Logic-Based AI in Honour of Vladimir Lifschitz*, E. Erdem, J. Lee, Y. Lierler, and D. Pearce, Eds. Lecture Notes in Computer Science, vol. 7265. Springer, 178–194.

Denecker, M., Marek, V., and Truszczyński, M. 2000. *Approximations, Stable Operators, Well-Founded Fixpoints and Applications in Nonmonotonic Reasoning.* In: Logic-Based Artificial Intelligence. The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, Boston, MA, 127–144.

Denecker, M., Marek, V. W., and Truszczynski, M. 2004. Ultimate approximation and its application in nonmonotonic knowledge representation systems. *Inf. Comput. 192,* 1, 84–121.

ÉSIK, Z. 2015. Equational properties of stratified least fixed points (extended abstract). In *Logic, Language, Information, and Computation - 22nd International Workshop, WoLLIC 2015, Bloomington, IN, USA, July 20-23, 2015, Proceedings*, V. de Paiva, R. J. G. B. de Queiroz, L. S. Moss, D. Leivant, and A. G. de Oliveira, Eds. Lecture Notes in Computer Science, vol. 9160. Springer, 174–188.

ÉSIK, Z. AND RONDOGIANNIS, P. 2015. A fixed point theorem for non-monotonic functions. *Theoretical Computer Science 574*, 18–38.

FITTING, M. 2002. Fixpoint semantics for logic programming: a survey. *Theor. Comput. Sci. 278,* 1-2, 25–51.

GELDER, A. V., ROSS, K. A., AND SCHLIPF, J. S. 1991. The well-founded semantics for general logic programs. *J. ACM 38,* 3, 620–650.

PRZYMUSINSKI, T. C. 1989. Every logic program has a natural stratification and an iterated least fixed point model. In *Proceedings of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, March 29-31, 1989, Philadelphia, Pennsylvania, USA*. 11–21.

RONDOGIANNIS, P. AND SYMEONIDOU, I. 2016. Extensional semantics for higher-order logic programs with negation. In *Logics in Artificial Intelligence - 15th European Conference, JELIA 2016, Larnaca, Cyprus, November 9-11, 2016, Proceedings*, L. Michael and A. C. Kakas, Eds. Lecture Notes in Computer Science, vol. 10021. 447–462.

RONDOGIANNIS, P. AND SYMEONIDOU, I. 2017. The intricacies of three-valued extensional semantics for higher-order logic programs. *TPLP 17,* 5-6, 974–991.

TENNENT, R. D. 1991. *Semantics of programming languages.* Prentice Hall International Series in Computer Science. Prentice Hall.

VAN EMDEN, M. H. AND KOWALSKI, R. A. 1976. The semantics of predicate logic as a programming language. *J. ACM 23,* 4, 733–742.

WADGE, W. W. 1991. Higher-order horn logic programming. In *Logic Programming, Proceedings of the 1991 International Symposium, San Diego, California, USA, Oct. 28 - Nov 1, 1991*, V. A. Saraswat and K. Ueda, Eds. MIT Press, 289–303.

Appendix A: Mathematical Preliminaries and Proofs of Section 4

A partially ordered set (or poset) $(L, \leq)$ is called a *lattice* if for all $x, y \in L$ there exists a least upper bound and a greatest lower bound. A lattice $(L, \leq)$ is called *complete* if for all $S \subseteq L$, there exists a least upper bound and a greatest lower bound, denoted by $\bigvee S$ and $\bigwedge S$ respectively. Every complete lattice has a least element and a greatest element, denoted by $\perp$ and $\top$ respectively. We will use the following two convenient equivalent definitions of complete lattices (Davey and Priestley 2002, Theorem 2.31, page 47):

Theorem 4
A partially ordered set $(L, \leq)$ is a complete lattice if L has a least element and every non-empty subset $S \subseteq L$ has a least upper bound in L. Alternatively, $(L, \leq)$ is a complete lattice if L has a greatest element and every non-empty subset $S \subseteq L$ has a greatest lower bound in L.

Given a partially ordered set $(L, \leq)$, every linearly ordered subset S of L will be called a *chain*. A partially ordered set is *chain-complete* if it has a least element $\perp$ and every chain $S \subseteq L$ has a least upper bound.

Proposition 1
Let D be a nonempty set. For every predicate type π, $(\llbracket\pi\rrbracket_D, \leq_\pi)$ is a complete lattice and $(\llbracket\pi\rrbracket_D, \preceq_\pi)$ is a chain complete poset.

Proof
Consider the first statement and let π be an arbitrary predicate type. Recall that $\perp_{\leq_\pi}$ exists; it suffices to show that for every non-empty subset S of $\llbracket\pi\rrbracket_D$, the least upper bound of S exists and belongs to $\llbracket\pi\rrbracket_D$.

The least upper bound can be defined inductively on the structure of predicate types. If $\pi = o$, then $\bigvee_{\leq_o} S$ is defined in the obvious way. For $\pi = \iota \to \pi_1$, we define for all $d \in D$, $(\bigvee_{\leq_{\iota\to\pi_1}} S)(d) = \bigvee_{\leq_{\pi_1}} \{f(d) \mid f \in S\}$. Finally, if $\pi = \pi_1 \to \pi_2$, we define for all $d \in \llbracket\pi_1\rrbracket_D$, $(\bigvee_{\leq_{\pi_1\to\pi_2}} S)(d) = \bigvee_{\leq_{\pi_2}} \{f(d) \mid f \in S\}$. We need to verify that for type $\pi_1 \to \pi_2$ the least upper bound is a Fitting-monotonic function. This is a consequence of the following auxiliary statement, which we need to establish for every predicate type π:

Auxiliary statement: Let I be a non-empty index-set and let $d_i, d'_i \in \llbracket\pi\rrbracket_D$, $i \in I$. If for all $i \in I$, $d_i \preceq_\pi d'_i$, then $\bigvee_{\leq_\pi} \{d_i \mid i \in I\} \preceq_\pi \bigvee_{\leq_\pi} \{d'_i \mid i \in I\}$.

The proof of the auxiliary statement is by a simple induction on the structure of π. For type $\pi = o$ the statement follows by a case analysis on the value of $\bigvee_{\leq_\pi} \{d_i \mid i \in I\}$. For types $\iota \to \pi_1$ and $\pi_1 \to \pi_2$, the statement follows directly by the induction hypothesis. The auxiliary statement implies that $(\bigvee_{\leq_{\pi_1\to\pi_2}} S)$ is a Fitting-monotonic function. More specifically, for all $d, d' \in \llbracket\pi_1\rrbracket_D$ with $d \preceq_{\pi_1} d'$, it holds $f(d) \preceq_{\pi_2} f(d')$ for every $f \in S$ (because the members of S are Fitting-monotonic functions). Then, the auxiliary statement implies that $\bigvee_{\leq_{\pi_2}} \{f(d) \mid f \in S\} \preceq_{\pi_2} \bigvee_{\leq_{\pi_2}} \{f(d) \mid f \in S\}$ which is equivalent to $(\bigvee_{\leq_{\pi_1\to\pi_2}} S)(d) \preceq_{\pi_2} (\bigvee_{\leq_{\pi_1\to\pi_2}} S)(d')$, which means that $(\bigvee_{\leq_{\pi_1\to\pi_2}} S)$ is Fitting-monotonic.

Consider now the second statement. Notice that $(\llbracket\pi\rrbracket_D, \preceq_\pi)$ is not a complete lattice (for example, the set $\{\mathit{false}, \mathit{true}\}$ does not have a least upper bound with respect to $\preceq_o$). However, it is a chain complete poset. For every type π, $\perp_{\preceq_\pi}$ exists. Moreover, given a chain S of elements of $[\![\pi]\!]_D$, it suffices to verify that $\bigvee_{\preceq_\pi} S$ exists and belongs to $[\![\pi]\!]_D$. The proof is by induction on the structure of π. For type $\pi = o$ it is obvious. For $\pi = \iota \to \pi_1$, define $(\bigvee_{\preceq_{\iota\to\pi_1}} S)(d) = \bigvee_{\preceq_{\pi_1}}\{f(d) \mid f \in S\}$. For $\pi = \pi_1 \to \pi_2$ define $(\bigvee_{\preceq_{\pi_1\to\pi_2}} S)(d) = \bigvee_{\preceq_{\pi_2}}\{f(d) \mid f \in S\}$. We need to verify that $(\bigvee_{\preceq_{\pi_1\to\pi_2}} S)$ is a Fitting-monotonic function, i.e., that for all $d, d' \in [\![\pi_1]\!]_D$ with $d \preceq_{\pi_1} d'$, it holds $(\bigvee_{\preceq_{\pi_1\to\pi_2}} S)(d) \preceq_{\pi_2} (\bigvee_{\preceq_{\pi_1\to\pi_2}} S)(d')$, or equivalently that $\bigvee_{\preceq_{\pi_2}}\{f(d) \mid f \in S\} \preceq_{\pi_2} \bigvee_{\preceq_{\pi_2}}\{f(d') \mid f \in S\}$, which holds because for every $f \in S$, $f(d) \preceq_{\pi_2} f(d')$. □

The proof of the above lemma has as a direct consequence the following corollary:

Corollary 1
Let D be a nonempty set and π a predicate type. Let I be a non-empty index-set and let $d_i, d'_i \in [\![\pi]\!]_D$, $i \in I$. If for all $i \in I$, $d_i \preceq_\pi d'_i$, then $\bigvee_{\preceq_\pi}\{d_i \mid i \in I\} \preceq_\pi \bigvee_{\preceq_\pi}\{d'_i \mid i \in I\}$.

Appendix B: Proofs of Section 5

Proposition 2
Let D be a nonempty set. For every predicate type π, $([\![\pi]\!]_D^{\mathsf{ma}}, \leq_\pi)$ and $([\![\pi]\!]_D^{\mathsf{am}}, \leq_\pi)$ are complete lattices.

Proof
We give the proof for the case $([\![\pi]\!]_D^{\mathsf{ma}}, \leq_\pi)$; the case $([\![\pi]\!]_D^{\mathsf{am}}, \leq_\pi)$ is symmetrical and omitted. The proof is by induction on the structure of π. For $\pi = o$ the result is immediate. We show the result for types $\iota \to \pi$ and $\pi_1 \to \pi_2$, assuming it holds for π, π_1 and π_2.

Consider first the set $[\![\iota \to \pi]\!]_D^{\mathsf{ma}} = D \to [\![\pi]\!]^{\mathsf{ma}}$. This set has a least element, namely the function that assigns to each $d \in D$ the bottom element of type π. Let $S \subseteq D \to [\![\pi]\!]^{\mathsf{ma}}$ be a nonempty set. For every $d \in D$ we define $(\bigvee_{\leq_{\iota\to\pi}} S)(d) = \bigvee_{\leq_\pi}\{f(d) \mid f \in S\}$, which by the induction hypothesis exists and belongs to $[\![\pi]\!]_D^{\mathsf{ma}}$.

Consider now the set $[\![\pi_1 \to \pi_2]\!]_D^{\mathsf{ma}} = [([\![\pi_1]\!]_D^{\mathsf{ma}} \otimes [\![\pi_1]\!]_D^{\mathsf{am}}) \overset{\mathsf{ma}}{\to} [\![\pi_2]\!]_D^{\mathsf{ma}}]$. This set has a least element, namely the function that assigns to each pair $(x, y) \in ([\![\pi_1]\!]_D^{\mathsf{ma}} \otimes [\![\pi_1]\!]_D^{\mathsf{am}})$ the bottom element of type $\perp_{\pi_2}$; this function is constant and therefore obviously monotone-antimonotone. Let $S \subseteq [([\![\pi_1]\!]_D^{\mathsf{ma}} \otimes [\![\pi_1]\!]_D^{\mathsf{am}}) \overset{\mathsf{ma}}{\to} [\![\pi_2]\!]_D^{\mathsf{ma}}]$ be a nonempty set. For every $(x, y) \in ([\![\pi_1]\!]_D^{\mathsf{ma}} \otimes [\![\pi_1]\!]_D^{\mathsf{am}})$ we define $(\bigvee_{\leq_{\pi_1\to\pi_2}} S)(x, y) = \bigvee_{\leq_{\pi_2}}\{f(x, y) \mid f \in S\}$, which by the induction hypothesis exists and belongs to $[\![\pi_2]\!]_D^{\mathsf{ma}}$. It remains to show that $\bigvee S$ is monotone-antimonotone. Consider $(x, y), (x', y') \in ([\![\pi_1]\!]_D^{\mathsf{ma}} \otimes [\![\pi_1]\!]_D^{\mathsf{am}})$ and assume that $x \leq x'$ and $y \geq y'$. It suffices to show that $(\bigvee_{\leq_{\pi_1\to\pi_2}} S)(x, y) \leq_{\pi_2} (\bigvee_{\leq_{\pi_1\to\pi_2}} S)(x', y')$. Since every element of S is monotone-antimonotone, for every $f \in S$ it holds $f(x, y) \leq_{\pi_2} f(x', y')$. Therefore, $\bigvee_{\leq_{\pi_2}}\{f(x, y) \mid f \in S\} \leq_{\pi_2} \bigvee_{\leq_{\pi_2}}\{f(x', y') \mid f \in S\}$, and thus $(\bigvee S_{\leq_{\pi_1\to\pi_2}})(x, y) \leq_{\pi_2} (\bigvee S_{\leq_{\pi_1\to\pi_2}})(x', y')$. □

The proof of Proposition 3 requires the following lemma which can be established by induction on the structure of π:

Lemma 6
Let D be a nonempty set and let π be a predicate type. Let $S \subseteq [\![\pi]\!]_D^{\mathsf{ma}}$ and $g \in [\![\pi]\!]_D^{\mathsf{am}}$.

- If for all $f \in S$, $f \leq g$, then $\bigvee S \leq g$.
- If for all $f \in S$, $f \geq g$, then $\bigwedge S \geq g$.

Proposition 3
Let D be a nonempty set. For each predicate type π, $[\![\pi]\!]_D^{\mathsf{ma}} \otimes [\![\pi]\!]_D^{\mathsf{am}}$ is a complete lattice with respect to $\leq_\pi$ and a chain-complete poset with respect to $\preceq_\pi$.

Proof
For every π it is straightforward to define the bottom elements of the partially ordered sets $([\![\pi]\!]_D^{\mathsf{ma}} \otimes [\![\pi]\!]_D^{\mathsf{am}}, \leq_\pi)$ and $([\![\pi]\!]_D^{\mathsf{ma}} \otimes [\![\pi]\!]_D^{\mathsf{am}}, \preceq_\pi)$.

Given $S \subseteq [\![\pi]\!]_D^{\mathsf{ma}} \otimes [\![\pi]\!]_D^{\mathsf{am}}$, we define $\bigvee_{\leq_\pi} S = (\bigvee_{\leq_\pi}\{f \mid (f,g) \in S\}, \bigvee_{\leq_\pi}\{g \mid (f,g) \in S\})$. It can be easily seen that $\bigvee_{\leq_\pi} S \in [\![\pi]\!]_D^{\mathsf{ma}} \otimes [\![\pi]\!]_D^{\mathsf{am}}$ due to Proposition 2, Lemma 6 and the fact that for every pair $(f,g) \in S$, $f \leq_\pi g$.

On the other hand, let $S \subseteq [\![\pi]\!]_D^{\mathsf{ma}} \otimes [\![\pi]\!]_D^{\mathsf{am}}$ be a chain. We define $\bigvee_{\preceq_\pi} S = (\bigvee_{\leq_\pi}\{f \mid (f,g) \in S\}, \bigwedge_{\leq_\pi}\{g \mid (f,g) \in S\})$. It is straightforward to show that $\bigvee_{\preceq_\pi} S$ is the $\preceq_\pi$-least upper bound of the chain. Moreover, $(\bigvee_{\preceq_\pi} S) \in [\![\pi]\!]_D^{\mathsf{ma}} \otimes [\![\pi]\!]_D^{\mathsf{am}}$ because $\bigvee_{\leq_\pi}\{f \mid (f,g) \in S\} \leq_\pi \bigwedge_{\leq_\pi}\{g \mid (f,g) \in S\}$ (this can easily be shown using basic properties of lubs and glbs, Lemma 6, and the fact that S is a chain; see also Proposition 2.3 in (Denecker et al. 2004)). $\square$

Proposition 4
Let D be a nonempty set and let π be a predicate type. Then, for every $f, g \in [\![\pi]\!]_D$ and for every $(f_1, f_2), (g_1, g_2) \in [\![\pi]\!]_D^{\mathsf{ma}} \otimes [\![\pi]\!]_D^{\mathsf{am}}$, the following statements hold:

1. $\tau_\pi(f) \in ([\![\pi]\!]_D^{\mathsf{ma}} \otimes [\![\pi]\!]_D^{\mathsf{am}})$ and $\tau_\pi^{-1}(f_1, f_2) \in [\![\pi]\!]_D$.
2. If $f \preceq_\pi g$ then $\tau_\pi(f) \preceq_\pi \tau_\pi(g)$.
3. If $f \leq_\pi g$ then $\tau_\pi(f) \leq_\pi \tau_\pi(g)$.
4. If $(f_1, f_2) \preceq_\pi (g_1, g_2)$ then $\tau_\pi^{-1}(f_1, f_2) \preceq_\pi \tau_\pi^{-1}(g_1, g_2)$.
5. If $(f_1, f_2) \leq_\pi (g_1, g_2)$ then $\tau_\pi^{-1}(f_1, f_2) \leq_\pi \tau_\pi^{-1}(g_1, g_2)$.

Proof
The five statements are shown by a simultaneous induction on the structure of π. We give the proofs for Statement 1, Statement 2 (the proof of Statement 3 is analogous and omitted) and Statement 4 (the proof of Statement 5 is similar and omitted).

The basis case is for $\pi = o$ and is straightforward for all statements. We assume the statements hold for π, π_1 and π_2. We demonstrate that they hold for $\iota \to \pi$ and for $\pi_1 \to \pi_2$.

Statement 1: Consider first the case of $\iota \to \pi$. It suffices to show that $\tau_{\iota\to\pi}(f) \in ([\![\iota \to \pi]\!]_D^{\mathsf{ma}} \otimes [\![\iota \to \pi]\!]_D^{\mathsf{am}})$. By the induction hypothesis, $\tau_\pi(f(d)) \in ([\![\pi]\!]_D^{\mathsf{ma}} \otimes [\![\pi]\!]_D^{\mathsf{am}})$. Therefore, $[\tau_\pi(f(d))]_1 \leq [\tau_\pi(f(d))]_2$, and consequently $(\lambda d.[\tau_\pi(f(d))]_1, \lambda d.[\tau_\pi(f(d))]_2) \in ([\![\iota \to \pi]\!]_D^{\mathsf{ma}} \otimes [\![\iota \to \pi]\!]_D^{\mathsf{am}})$. We next show that $\tau_\pi^{-1}(f_1, f_2) \in [\![\iota \to \pi]\!]_D$. Since $(f_1, f_2) \in ([\![\iota \to \pi]\!]_D^{\mathsf{ma}} \otimes [\![\iota \to \pi]\!]_D^{\mathsf{am}})$, $f_1 \leq f_2$ and $(f_1(d), f_2(d)) \in ([\![\pi]\!]_D^{\mathsf{ma}} \otimes [\![\pi]\!]_D^{\mathsf{am}})$. By the induction hypothesis, $\tau_\pi^{-1}(f_1(d), f_2(d)) \in [\![\pi]\!]_D$ and $\lambda d.\tau_\pi^{-1}(f_1(d), f_2(d)) \in [\![\iota \to \pi]\!]_D$.

Consider the case $\pi_1 \to \pi_2$. We show that $\tau_{\pi_1\to\pi_2}(f) \in ([\![\pi_1 \to \pi_2]\!]_D^{\mathsf{ma}} \otimes [\![\pi_1 \to \pi_2]\!]_D^{\mathsf{am}})$. Let $(d_1, d_2) \in ([\![\pi_1]\!]_D^{\mathsf{ma}} \otimes [\![\pi_1]\!]_D^{\mathsf{am}})$. By the induction hypothesis $\tau_{\pi_1}^{-1}(d_1, d_2) \in [\![\pi_1]\!]_D$, $f(\tau_{\pi_1}^{-1}(d_1, d_2)) \in [\![\pi_2]\!]_D$, and $\tau_{\pi_2}(f(\tau_{\pi_1}^{-1}(d_1, d_2))) \in ([\![\pi_1]\!]_D^{\mathsf{ma}} \otimes [\![\pi_1]\!]_D^{\mathsf{am}})$, which has as a direct consequence that $[\tau_{\pi_2}(f(\tau_{\pi_1}^{-1}(d_1, d_2)))]_1 \leq [\tau_{\pi_2}(f(\tau_{\pi_1}^{-1}(d_1, d_2)))]_2$. Therefore, $\lambda(d_1, d_2).[\tau_{\pi_2}(f(\tau_{\pi_1}^{-1}(d_1, d_2)))]_1 \leq \lambda(d_1, d_2).[\tau_{\pi_2}(f(\tau_{\pi_1}^{-1}(d_1, d_2)))]_2$. It remains to show that the function $\lambda(d_1, d_2).[\tau_{\pi_2}(f(\tau_{\pi_1}^{-1}(d_1, d_2)))]_1$ is monotone-antimonotone and the function

$\lambda(d_1,d_2).[\tau_{\pi_2}(f(\tau_{\pi_1}^{-1}(d_1,d_2)))]_2$ is antimonotone-monotone. This follows by using the induction hypothesis for Statement 4, the Fitting-monotonicity of f, and the induction hypothesis of Statement 2. The fact that $\tau_{\pi_1\to\pi_2}^{-1}(f_1,f_2) \in [\![\pi_1 \to \pi_2]\!]_D$ follows using similar arguments as above.

Statement 2: Consider first the case of $\iota \to \pi$. It suffices to show that:

$$(\lambda d.[\tau_\pi(f(d))]_1, \lambda d.[\tau_\pi(f(d))]_2) \preceq (\lambda d.[\tau_\pi(g(d))]_1, \lambda d.[\tau_\pi(g(d))]_2)$$

or equivalently that $\lambda d.[\tau_\pi(f(d))]_1 \leq \lambda d.[\tau_\pi(g(d))]_1$ and $\lambda d.[\tau_\pi(f(d))]_2 \geq \lambda d.[\tau_\pi(g(d))]_2$, or equivalently that for every d, $[\tau_\pi(f(d))]_1 \leq [\tau_\pi(g(d))]_1$ and $[\tau_\pi(f(d))]_2 \geq [\tau_\pi(g(d))]_2$. This holds because, since $f \preceq g$, it holds $f(d) \preceq g(d)$ and by the induction hypothesis, $\tau_\pi(f(d)) \preceq \tau_\pi(g(d))$. Consider now the case of $\pi_1 \to \pi_2$. It suffices to show that:

$$\begin{aligned}&(\lambda(d_1,d_2).[\tau_{\pi_2}(f(\tau_{\pi_1}^{-1}(d_1,d_2)))]_1, \lambda(d_1,d_2).[\tau_{\pi_2}(f(\tau_{\pi_1}^{-1}(d_1,d_2)))]_2) \preceq\\ &(\lambda(d_1,d_2).[\tau_{\pi_2}(g(\tau_{\pi_1}^{-1}(d_1,d_2)))]_1, \lambda(d_1,d_2).[\tau_{\pi_2}(g(\tau_{\pi_1}^{-1}(d_1,d_2)))]_2)\end{aligned}$$

or equivalently that $\lambda(d_1,d_2).[\tau_{\pi_2}(f(\tau_{\pi_1}^{-1}(d_1,d_2)))]_1 \leq \lambda(d_1,d_2).[\tau_{\pi_2}(g(\tau_{\pi_1}^{-1}(d_1,d_2)))]_1$ and $\lambda(d_1,d_2).[\tau_{\pi_2}(f(\tau_{\pi_1}^{-1}(d_1,d_2)))]_2 \geq \lambda(d_1,d_2).[\tau_{\pi_2}(g(\tau_{\pi_1}^{-1}(d_1,d_2)))]_2$, or equivalently that for all d_1,d_2, $[\tau_{\pi_2}(f(\tau_{\pi_1}^{-1}(d_1,d_2)))]_1 \leq [\tau_{\pi_2}(g(\tau_{\pi_1}^{-1}(d_1,d_2)))]_1$ and $[\tau_{\pi_2}(f(\tau_{\pi_1}^{-1}(d_1,d_2)))]_2 \geq [\tau_{\pi_2}(g(\tau_{\pi_1}^{-1}(d_1,d_2)))]_2$. Since $f \preceq g$, it holds that $f(\tau_{\pi_1}^{-1}(d_1,d_2))) \preceq g(\tau_{\pi_1}^{-1}(d_1,d_2)))$, and by the induction hypothesis $\tau_{\pi_2}(f(\tau_{\pi_1}^{-1}(d_1,d_2))) \preceq \tau_{\pi_2}(g(\tau_{\pi_1}^{-1}(d_1,d_2)))$, which is the desired result.

Statement 4: Consider first the case of $\iota \to \pi$. It suffices to show that:

$$\lambda d.\tau_\pi^{-1}(f_1(d), f_2(d)) \preceq \lambda d.\tau_\pi^{-1}(g_1(d), g_2(d))$$

or equivalently that for every d, $\tau_\pi^{-1}(f_1(d), f_2(d)) \preceq \tau_\pi^{-1}(g_1(d), g_2(d))$. Since $(f_1,f_2) \preceq (g_1,g_2)$, it holds $(f_1(d), f_2(d)) \preceq (g_1(d), g_2(d))$, and the result follows from the induction hypothesis. Consider now the case of $\pi_1 \to \pi_2$. It suffices to show that:

$$\lambda d.\tau_{\pi_2}^{-1}(f_1(\tau_{\pi_1}(d)), f_2(\tau_{\pi_1}(d))) \preceq \lambda d.\tau_{\pi_2}^{-1}(g_1(\tau_{\pi_1}(d)), g_2(\tau_{\pi_1}(d)))$$

or equivalently that for every d, $\tau_{\pi_2}^{-1}(f_1(\tau_{\pi_1}(d)), f_2(\tau_{\pi_1}(d))) \preceq \tau_{\pi_2}^{-1}(g_1(\tau_{\pi_1}(d)), g_2(\tau_{\pi_1}(d)))$. Since $(f_1,f_2) \preceq (g_1,g_2)$, it holds $(f_1(\tau_{\pi_1}(d)), f_2(\tau_{\pi_1}(d))) \preceq (g_1(\tau_{\pi_1}(d)), g_2(\tau_{\pi_1}(d)))$, and the result follows from the induction hypothesis. □

Proposition 5

Let D be a nonempty set and let π be a predicate type. Then, for every $f \in [\![\pi]\!]_D$, $\tau_\pi^{-1}(\tau_\pi(f)) = f$, and for every $(f_1,f_2) \in [\![\pi]\!]_D^{\mathsf{ma}} \otimes [\![\pi]\!]_D^{\mathsf{am}}$, $\tau_\pi(\tau_\pi^{-1}(f_1,f_2)) = (f_1,f_2)$.

Proof

The proof of the two statements is by a simultaneous induction on the structure of π. The case $\pi = o$ is immediate. Assume the two statements hold for π, π_1 and π_2. We demonstrate that they hold for $\iota \to \pi$ and for $\pi_1 \to \pi_2$.

We have:

$$
\begin{aligned}
& \tau_{\iota\to\pi}^{-1}(\tau_{\iota\to\pi}(f)) \; = \\
= \; & \tau_{\iota\to\pi}^{-1}(\lambda d.[\tau_\pi(f(d))]_1, \lambda d.[\tau_\pi(f(d))]_2) \\
& \text{(Definition of } \tau_{\iota\to\pi}) \\
= \; & \lambda d.\tau_\pi^{-1}([\tau_\pi(f(d))]_1, [\tau_\pi(f(d))]_2) \\
& \text{(Definition of } \tau_{\iota\to\pi}^{-1}) \\
= \; & \lambda d.\tau_\pi^{-1}(\tau_\pi(f(d))) \\
& \text{(Definition of } [\cdot]_1 \text{ and } [\cdot]_2) \\
= \; & \lambda d.f(d) \\
& \text{(Induction Hypothesis)} \\
= \; & f
\end{aligned}
$$

Also:

$$
\begin{aligned}
& \tau_{\iota\to\pi}(\tau_{\iota\to\pi}^{-1}(f_1, f_2)) \; = \\
= \; & \tau_{\iota\to\pi}(\lambda d.\tau_\pi^{-1}(f_1(d), f_2(d))) \\
& \text{(Definition of } \tau_{\iota\to\pi}^{-1}) \\
= \; & (\lambda d.[\tau_\pi(\tau_\pi^{-1}(f_1(d), f_2(d)))]_1, \lambda d.[\tau_\pi(\tau_\pi^{-1}(f_1(d), f_2(d)))]_2) \\
& \text{(Definition of } \tau_{\iota\to\pi}) \\
= \; & (\lambda d.[(f_1(d), f_2(d))]_1, \lambda d.[(f_1(d), f_2(d))]_2) \\
& \text{(Induction Hypothesis)} \\
= \; & (\lambda d.f_1(d), \lambda d.f_2(d)) \\
& \text{(Definition of } [\cdot]_1 \text{ and } [\cdot]_2) \\
= \; & (f_1, f_2)
\end{aligned}
$$

Consider now the case of $\pi_1 \to \pi_2$. We have:

$$
\begin{aligned}
& \tau_{\pi_1\to\pi_2}^{-1}(\tau_{\pi_1\to\pi_2}(f)) \; = \\
= \; & \tau_{\pi_1\to\pi_2}^{-1}(\lambda(d_1, d_2).[\tau_{\pi_2}(f(\tau_{\pi_1}^{-1}(d_1, d_2)))]_1, \lambda(d_1, d_2).[\tau_{\pi_2}(f(\tau_{\pi_1}^{-1}(d_1, d_2)))]_2) \\
& \text{(Definition of } \tau_{\pi_1\to\pi_2}) \\
= \; & \lambda d.\tau_{\pi_2}^{-1}([\tau_{\pi_2}(f(\tau_{\pi_1}^{-1}(\tau_{\pi_1}(d))))]_1, [\tau_{\pi_2}(f(\tau_{\pi_1}^{-1}(\tau_{\pi_1}(d))))]_2) \\
& \text{(Definition of } \tau_{\pi_1\to\pi_2}^{-1}) \\
= \; & \lambda d.\tau_{\pi_2}^{-1}([\tau_{\pi_2}(f(d))]_1, [\tau_{\pi_2}(f(d))]_2) \\
& \text{(Induction Hypothesis)} \\
= \; & \lambda d.\tau_{\pi_2}^{-1}(\tau_{\pi_2}(f(d))) \\
& \text{(Definition of } [\cdot]_1 \text{ and } [\cdot]_2) \\
= \; & \lambda d.f(d) \\
& \text{(Induction Hypothesis)} \\
= \; & f
\end{aligned}
$$

Also:

$$
\begin{array}{rl}
& \tau_{\pi_1\to\pi_2}(\tau^{-1}_{\pi_1\to\pi_2}(f_1,f_2)) \;= \\
= & \tau_{\pi_1\to\pi_2}(\lambda d.\tau^{-1}_{\pi_2}(f_1(\tau_{\pi_1}(d)), f_2(\tau_{\pi_1}(d))) \\
& \text{(Definition of } \tau^{-1}_{\pi_1\to\pi_2}) \\
= & (\lambda(d_1,d_2).[\tau_{\pi_2}(\tau^{-1}_{\pi_2}(f_1(\tau_{\pi_1}(\tau^{-1}_{\pi_1}(d_1,d_2))), f_2(\tau_{\pi_1}(\tau^{-1}_{\pi_1}(d_1,d_2)))))]_1, \\
& \;\lambda(d_1,d_2).[\tau_{\pi_2}(\tau^{-1}_{\pi_2}(f_1(\tau_{\pi_1}(\tau^{-1}_{\pi_1}(d_1,d_2))), f_2(\tau_{\pi_1}(\tau^{-1}_{\pi_1}(d_1,d_2)))))]_2) \\
& \text{(Definition of } \tau_{\pi_1\to\pi_2}) \\
= & (\lambda(d_1,d_2).[f_1(d_1,d_2), f_2(d_1,d_2)]_1, \lambda(d_1,d_2).[f_1(d_1,d_2), f_2(d_1,d_2)]_2) \\
& \text{(Induction Hypothesis)} \\
= & (\lambda(d_1,d_2).f_1(d_1,d_2), \lambda(d_1,d_2).f_2(d_1,d_2)) \\
& \text{(Definition of } [\cdot]_1 \text{ and } [\cdot]_2) \\
= & (f_1,f_2)
\end{array}
$$

The above completes the proof of the proposition. □

Appendix C: An Extension of Consistent Approximation Fixpoint Theory

In this appendix we propose a mild extension of the theory of consistent approximating operators developed in (Denecker et al. 2004). We briefly highlight the main idea behind the work in (Denecker et al. 2004) and then justify the necessity for our extension.

Let $(L, \leq)$ be a complete lattice. The authors in (Denecker et al. 2004) consider the set $L^c = \{(x, y) \in L \times L \mid x \leq y\}$. Intuitively speaking, a pair $(x, y) \in L^c$ can be viewed as an approximation to all elements $z \in L$ such that $x \leq z \leq y$. An operator $A : L^c \to L^c$ is called in (Denecker et al. 2004) a *consistent approximating operator* if it is $\preceq$-monotone (see below) and for every $x \in L$, $A(x, x)_1 = A(x, x)_2$ (the subscripts 1 and 2 denote projection to the first and second elements respectively of the pair returned by A). In Section 3 of (Denecker et al. 2004), an elegant theory is developed whose purpose is to demonstrate how, under specific conditions, one can characterize the *well-founded fixpoint* of a given consistent approximating operator A. Since approximating operators emerge in many non-monotonic formalisms, the theory developed in (Denecker et al. 2004) provides a useful tool for the study of the semantics of such formalisms.

In our work, the immediate consequence operator T_{P} is not an approximating operator in the sense of (Denecker et al. 2004). More specifically, T_{P} is a function in $(\mathcal{H}^{\mathsf{ma}}_{\mathsf{P}} \otimes \mathcal{H}^{\mathsf{am}}_{\mathsf{P}}) \to (\mathcal{H}^{\mathsf{ma}}_{\mathsf{P}} \otimes \mathcal{H}^{\mathsf{am}}_{\mathsf{P}})$. In other words, there is not just a single lattice L involved in the definition of T_{P}, but instead two lattices, namely $\mathcal{H}^{\mathsf{ma}}_{\mathsf{P}}$ and $\mathcal{H}^{\mathsf{am}}_{\mathsf{P}}$. Moreover, the condition "for every $x \in L$, $A(x, x)_1 = A(x, x)_2$" required in (Denecker et al. 2004), does not hold in our case, because the two arguments of T_{P} range over two different sets (namely $\mathcal{H}^{\mathsf{ma}}_{\mathsf{P}}$ and $\mathcal{H}^{\mathsf{am}}_{\mathsf{P}}$). We therefore need to define an extension of the material in Section 3 of (Denecker et al. 2004), that suits our purposes.

In the following, we develop the above mentioned extension following closely the statements and proofs of (Denecker et al. 2004). The material is presented in an abstract form (as in (Denecker et al. 2004)), with the purpose of having a wider applicability than the present paper. In order to retrieve the connections with the present paper, one can take $A = T_{\mathsf{P}}$, $L_1 = \mathcal{H}^{\mathsf{ma}}_{\mathsf{P}}$ and $L_2 = \mathcal{H}^{\mathsf{am}}_{\mathsf{P}}$.

Let $(L, \leq)$ be a partially ordered set and assume that L contains a least element $\perp$ and a greatest element $\top$ with respect to $\leq$. Let $L_1, L_2 \subseteq L$ be non-empty sets such that $L_1 \cup L_2 = L$ and $(L_1, \leq)$ and $(L_2, \leq)$ are complete lattices that both contain the elements $\perp$ and $\top$. We will denote the least upper bound operations in the two lattices by lub_{L_1} and lub_{L_2} respectively (we will also use $\bigvee_{L_1}$ and $\bigvee_{L_2}$). We denote the greatest lower bound operations by glb_{L_1} and glb_{L_2} (also denoted by $\bigwedge_{L_1}$ and $\bigwedge_{L_2}$). We assume that our lattices satisfy the following two properties:

1. **Interlattice Lub Property:** Let $b \in L_2$ and $S \subseteq L_1$ such that for every $x \in S$, $x \leq b$. Then, $\bigvee_{L_1} S \leq b$.
2. **Interlattice Glb Property:** Let $a \in L_1$ and $S \subseteq L_2$ such that for every $x \in S$, $x \geq a$. Then, $\bigwedge_{L_2} S \geq a$.

Remark: It can be easily verified (see Lemma 6 in Appendix B) that both the Interlattice Lub Property and the Interlattice Glb Property hold when we take $L_1 = \mathcal{H}_{\mathsf{P}}^{\mathsf{ma}}$ and $L_2 = \mathcal{H}_{\mathsf{P}}^{\mathsf{am}}$.

Given $(x, y), (x', y') \in L_1 \times L_2$, we will write $(x, y) \preceq (x', y')$ if $x \leq x'$ and $y' \leq y$. We will write:

$$L_1 \otimes L_2 = \{(x, y) \mid x \in L_1, y \in L_2, x \leq y\}$$

The above set is non-empty since $(\perp, \top) \in L_1 \otimes L_2$.

Definition 22
A function $A : L_1 \otimes L_2 \to L_1 \otimes L_2$ is called a *consistent approximating operator* if it is $\preceq$-monotonic.

We will write $Appx(L_1 \otimes L_2)$ for the set of all consistent approximating operators over $L_1 \otimes L_2$. In the following results we assume we work with a given consistent approximating operator A (and therefore the symbol A will appear free in most definitions and results).

Definition 23
The pair $(a, b) \in L_1 \otimes L_2$ will be called A-reliable if $(a, b) \preceq A(a, b)$.

Given $a \in L_1$ and $b \in L_2$, we write $[a, b]_{L_1} = \{x \in L_1 \mid a \leq x \leq b\}$. Symmetrically, $[a, b]_{L_2} = \{x \in L_2 \mid a \leq x \leq b\}$.

Proposition 8
For all $a \in L_1$ and $b \in L_2$, the sets $[\perp, b]_{L_1}$ and $[a, \top]_{L_2}$ are complete lattices.

Proof
We use Theorem 4 of Appendix A. Consider first the set $[\perp, b]_{L_1}$ which obviously has a least element (since $\perp$ is the least element of both L_1 and L_2 and therefore $\perp \in [\perp, b]_{L_1}$). Let S be a non-empty subset of $[\perp, b]_{L_1}$. Since L_1 is a complete lattice, $\bigvee_{L_1} S \in L_1$. It suffices to show that $\bigvee_{L_1} S \in [\perp, b]_{L_1}$. Since $S \subseteq [\perp, b]_{L_1}$, for every $x \in S$ it holds $x \leq b$. By the Interlattice Lub Property, $\bigvee_{L_1} S \leq b$, and therefore $\bigvee_{L_1} S \in [\perp, b]_{L_1}$.

The proof for the case of $[a, \top]_{L_2}$ is symmetrical and uses the Interlattice Glb Property instead. $\square$

The following proposition corresponds to Proposition 3.3 in (Denecker et al. 2004):

Proposition 9
Let $(a, b) \in L_1 \otimes L_2$ and assume that (a, b) is A-reliable. Then, for every $x \in [\perp, b]_{L_1}$, it holds $\perp \leq A(x, b)_1 \leq b$. Moreover, for every $x \in [a, \top]_{L_2}$, it holds $a \leq A(a, x)_2 \leq \top$.

Proof
Define $a^* = lub_{L_1}\{y \in L_1 \mid y \leq b\}$. By the fact that $a \leq b$ and the definition of a^*, we get that $a \leq a^*$. By the Interlattice Lub Property we get that $a^* \leq b$ and therefore $(a^*, b) \in L_1 \otimes L_2$. Moreover, $(x, b) \preceq (a^*, b)$. Due to the $\preceq$-monotonicity of A we have $A(x, b) \preceq A(a^*, b)$, and therefore $A(x, b)_1 \leq A(a^*, b)_1$. Then:

$$\begin{array}{rcll} A(a^*, b)_1 & \leq & A(a^*, b)_2 & \text{(Consistency of } A) \\ & \leq & A(a, b)_2 & (a \leq a^* \text{ and } A \text{ is } \preceq\text{-monotone}) \\ & \leq & b & (A\text{-reliability}) \end{array}$$

For the second part of the proof, define $b^* = glb_{L_2}\{y \in L_2 \mid y \geq a\}$. By the fact that $b \geq a$ and the definition of b^*, we get that $b^* \leq b$. By the Interlattice Glb Property we get that $b^* \geq a$ and therefore $(a, b^*) \in L_1 \otimes L_2$. Moreover, $(a, x) \preceq (a, b^*)$. Due to the $\preceq$-monotonicity of A we have $A(a, x) \preceq A(a, b^*)$, and therefore $A(a, x)_2 \geq A(a, b^*)_2$. Then:

$$\begin{array}{rcll} A(a, b^*)_2 & \geq & A(a, b^*)_1 & \text{(Consistency of } A) \\ & \geq & A(a, b)_1 & (b^* \leq b \text{ and } A \text{ is } \preceq\text{-monotone}) \\ & \geq & a & (A\text{-reliability}) \end{array}$$

This completes the proof of the proposition. $\square$

The above proposition implies that for every A-reliable pair (a, b), the restriction of $A(., b)_1$ to $[\bot, b]_{L_1}$ and the restriction of $A(a, .)_2$ to $[a, \top]_{L_2}$ are in fact operators (namely functions $[\bot, b]_{L_1} \to [\bot, b]_{L_1}$ and $[a, \top]_{L_2} \to [a, \top]_{L_2}$) on these intervals. Since by Proposition 8 we know that $([\bot, b]_{L_1}, \leq)$ and $([a, \top]_{L_2}, \leq)$ are complete lattices, the operators $A(\cdot, b)_1$ and $A(a, \cdot)_2$ have least fixpoints in the corresponding lattices. We define:

$$b^{\downarrow} = lfp(A(\cdot, b)_1)$$

and

$$a^{\uparrow} = lfp(A(a, \cdot)_2)$$

In the following, we will call the function mapping the A-reliable pair (a, b) to $(b^{\downarrow}, a^{\uparrow})$, the *stable revision operator* for the approximating operator A. We will denote this mapping by $\mathcal{C}_A$, namely:

$$\mathcal{C}_A(x, y) = (y^{\downarrow}, x^{\uparrow}) = (lfp(A(\cdot, y)_1), lfp(A(x, \cdot)_2))$$

We have the following proposition, which corresponds to Proposition 3.6 of (Denecker et al. 2004):

Proposition 10
Let $A \in Appx(L_1 \otimes L_2)$. For every A-reliable pair (a, b), $b^{\downarrow} \leq b$, $a \leq a^{\uparrow} \leq b$, and $(b^{\downarrow}, a^{\uparrow}) \in L_1 \otimes L_2$.

Proof
The inequalities $b^{\downarrow} \leq b$ and $a \leq a^{\uparrow}$ follow from the definition of the stable revision operator. By the A-reliability of (a, b) we have $A(a, b)_2 \leq b$ and therefore b is a pre-fixpoint of $A(a, \cdot)_2$. Since $a^{\uparrow}$ is the least pre-fixpoint of $A(a, \cdot)_2$, we conclude that $a^{\uparrow} \leq b$.

Let $a^* = lub_{L_1}\{x \in L_1 \mid x \leq a^{\uparrow}\}$. Since $a \in \{x \in L_1 \mid x \leq a^{\uparrow}\}$ and since a^* is the *lub* of this set, it holds $a \leq a^*$. Moreover, notice that a^* is in the domain of $A(\cdot, b)_1$ because (by the Interlattice Lub Property) $a^* \leq a^\uparrow$, and since $a^\uparrow \leq b$ we get $a^* \leq b$. We have:

$$\begin{array}{rcll} A(a^*, b)_1 & \leq & A(a^*, a^\uparrow)_1 & (A \text{ is } \preceq\text{-monotonic}) \\ & \leq & A(a^*, a^\uparrow)_2 & (A \text{ is consistent}) \\ & \leq & A(a, a^\uparrow)_2 & (A \text{ is } \preceq\text{-monotonic}) \\ & = & a^\uparrow & (a^\uparrow \text{ fixpoint of } A(a, \cdot)_2) \end{array}$$

Consequently, $A(a^*, b)_1 \leq a^\uparrow$ and therefore $A(a^*, b)_1 \in \{x \in L_1 \mid x \leq a^\uparrow\}$. But $a^* = lub_{L_1}\{x \in L_1 \mid x \leq a^\uparrow\}$ and therefore $A(a^*, b)_1 \leq a^*$. It follows that a^* is a pre-fixpoint of the operator $A(\cdot, b)_1$. Thus, $b^\downarrow = \mathit{lfp}(A(\cdot, b)_1) \leq a^* \leq a^\uparrow$. □

Definition 24
An A-reliable approximation (a, b) is A-prudent if $a \leq b^\downarrow$.

Proposition 11
Let $A \in \mathit{Appx}(L_1 \otimes L_2)$ and let $(a, b) \in L_1 \otimes L_2$ be A-prudent. Then, $(a, b) \preceq (b^\downarrow, a^\uparrow)$ and $(b^\downarrow, a^\uparrow)$ is A-prudent.

Proof
By Proposition 10, it holds $b^\downarrow \leq b$, $a \leq a^\uparrow$ and $a^\uparrow \leq b$. Since (a, b) is A-prudent, we get $(a, b) \preceq (b^\downarrow, a^\uparrow)$.

Notice now that by the $\preceq$ monotonicity of A we get that $b^\downarrow = A(b^\downarrow, b)_1 \leq A(b^\downarrow, a^\uparrow)_1$ and $a^\uparrow = A(a, a^\uparrow)_2 \geq A(b^\downarrow, a^\uparrow)_2$. This implies that $(b^\downarrow, a^\uparrow)$ is A-reliable.

Observe now that since $a^\uparrow \leq b$ and A is $\preceq$-monotonic, it holds that for every $x \in [\perp, a^\uparrow]_{L_1}$, $A(x, b)_1 \leq A(x, a^\uparrow)_1$. Therefore, each pre-fixpoint of $A(\cdot, a^\uparrow)_1$ is a pre-fixpoint of $A(\cdot, b)_1$. By the proof of Proposition 10 we have that $A(a^*, a^\uparrow)_1 \leq a^\uparrow$, and by the definition of a^* in that same proof, it follows that $A(a^*, a^\uparrow)_1 \leq a^*$. Therefore the set of pre-fixpoints of $A(\cdot, a^\uparrow)_1$ is non-empty. Consequently, $b^\downarrow = \mathit{lfp}(A(\cdot, b)_1) \leq \mathit{lfp}(A(\cdot, a^\uparrow)_1) = (a^\uparrow)^\downarrow$, and therefore $(b^\downarrow, a^\uparrow)$ is A-prudent. □

The following proposition (corresponding to Proposition 2.3 in (Denecker et al. 2004)) now requires in its proof the Interlattice Lub Property.

Proposition 12
Let $\{(a_\kappa, b_\kappa)\}_{\kappa<\lambda}$, where λ is an ordinal, be a chain in $L_1 \otimes L_2$ ordered by the relation $\preceq$. Then:

1. $\bigvee_{L_1}\{a_\kappa \mid \kappa < \lambda\} \leq \bigwedge_{L_2}\{b_\kappa \mid \kappa < \lambda\}$.
2. The least upper bound of the chain with respect to $\preceq$ exists, and is equal to $(\bigvee_{L_1}\{a_\kappa \mid \kappa < \lambda\}, \bigwedge_{L_2}\{b_\kappa \mid \kappa < \lambda\})$.

Proof
We demonstrate the first statement; the proof of the second part is easy and omitted. For the proof of the first part, notice that since the chain is ordered by $\preceq$, $\bigwedge_{L_2}\{b_\kappa \mid \kappa < \lambda\} = b_0$. Moreover, for every $\kappa < \lambda$ it holds $a_\kappa \leq b_\kappa$ because $(a_\kappa, b_\kappa) \in L_1 \otimes L_2$; since $b_\kappa \leq b_0$, it is $a_\kappa \leq b_0$ for all $\kappa < \lambda$. By the Interlattice Lub Property, we get $\bigvee_{L_1}\{a_\kappa \mid \kappa < \lambda\} \leq b_0 = \bigwedge_{L_2}\{b_\kappa \mid \kappa < \lambda\}$. □

The following proposition (corresponding to Proposition 3.10 in (Denecker et al. 2004)) and the subsequent theorem (corresponding to Theorem 3.11 in (Denecker et al. 2004)) have identical proofs to the ones given in (Denecker et al. 2004) (the only difference being that our underlying domain is $L_1 \otimes L_2$):

Proposition 13
Let $A \in \mathit{Appx}(L_1 \otimes L_2)$ and let $\{(a_\kappa, b_\kappa)\}_{\kappa<\lambda}$, where λ is an ordinal, be a chain of A-prudent pairs from $L_1 \otimes L_2$. Then, $\bigvee_{\preceq}\{(a_\kappa, b_\kappa)\}_{\kappa<\lambda}$, is A-prudent.

Theorem 5
Let $A \in \mathit{Appx}(L_1 \otimes L_2)$. The set of A-prudent elements of $L_1 \otimes L_2$ is a chain-complete poset under $\preceq$ with least element $(\bot, \top)$. The stable revision operator is a well-defined, increasing and monotone operator in this poset, and therefore it has a least fixpoint which is A-prudent and can be obtained as the limit of the following sequence:

$$\begin{array}{lcl} (a_0, b_0) & = & (\bot, \top) \\ (a_{\lambda+1}, b_{\lambda+1}) & = & \mathcal{C}_A(a_\lambda, b_\lambda) \\ (a_\lambda, b_\lambda) & = & \bigvee_{\preceq}\{(a_\kappa, b_\kappa) : \kappa < \lambda\} \quad \text{for limit ordinals } \lambda \end{array}$$

The proof of the following theorem is also a straightforward generalization of the proof of Theorem 19 in (Denecker et al. 2000):

Theorem 6
Every fixpoint of the stable revision operator $\mathcal{C}_A$ is a $\leq$-minimal pre-fixpoint of A.

Appendix D: Proofs of Section 6

Before providing the proofs of the results of Section 6, we notice that Proposition 4 extends to the case of Herbrand interpretations as follows:

Proposition 14
Let P be a program. Then, for every $\mathcal{I}, \mathcal{J} \in \mathcal{H}_\mathsf{P}$ and for every $(I_1, J_1), (I_2, J_2) \in (\mathcal{H}_\mathsf{P}^{\mathsf{ma}} \otimes \mathcal{H}_\mathsf{P}^{\mathsf{am}})$, the following statements hold:

1. $\tau(\mathcal{I}) \in (\mathcal{H}_\mathsf{P}^{\mathsf{ma}} \otimes \mathcal{H}_\mathsf{P}^{\mathsf{am}})$ and $\tau^{-1}(I_1, J_1) \in \mathcal{H}_\mathsf{P}$.
2. If $\mathcal{I} \preceq \mathcal{J}$ then $\tau(\mathcal{I}) \preceq \tau(\mathcal{J})$.
3. If $\mathcal{I} \leq \mathcal{J}$ then $\tau(\mathcal{I}) \leq \tau(\mathcal{J})$.
4. If $(I_1, J_1) \preceq (I_2, J_2)$ then $\tau^{-1}(I_1, J_1) \preceq \tau^{-1}(I_2, J_2)$.
5. If $(I_1, J_1) \leq (I_2, J_2)$ then $\tau^{-1}(I_1, J_1) \leq \tau^{-1}(I_2, J_2)$.

Lemma 3
Let P be a program and let $(I_1, J_1), (I_2, J_2) \in \mathcal{H}_\mathsf{P}^{\mathsf{ma}} \otimes \mathcal{H}_\mathsf{P}^{\mathsf{am}}$. If $(I_1, J_1) \preceq (I_2, J_2)$ then $T_\mathsf{P}(I_1, J_1) \preceq T_\mathsf{P}(I_2, J_2)$.

Proof
It follows directly from the definition of Ψ_P together with Lemma 2 and Corollary 1 in Appendix A that Ψ_P is $\preceq$-monotonic. It follows from Proposition 14 that $\tau^{-1}(I_1, J_1) \preceq \tau^{-1}(I_2, J_2)$. Since Ψ_P is $\preceq$-monotonic we get $\Psi_\mathsf{P}(\tau^{-1}(I_1, J_1)) \preceq \Psi_\mathsf{P}(\tau^{-1}(I_2, J_2))$. By applying again Proposition 4 we have that $T_\mathsf{P}(I_1, J_1) \preceq T_\mathsf{P}(I_2, J_2)$ that concludes the proof. $\square$

Lemma 4
Let P be a program. If $(I, J) \in \mathcal{H}_\mathsf{P}^{\mathsf{ma}} \otimes \mathcal{H}_\mathsf{P}^{\mathsf{am}}$ is a pre-fixpoint of T_P then $\tau^{-1}(I, J)$ is a model of P.

Proof
From the definition of T_{P} and using the fact that (I,J) is a pre-fixpoint of T_{P}, it follows that $\tau(\Psi_{\mathsf{P}}(\tau^{-1}(I,J))) = T_{\mathsf{P}}(I,J) \leq (I,J)$. By applying τ^{-1} to both sides of the statement and using Proposition 14 we get that $\tau^{-1}(\tau(\Psi_{\mathsf{P}}(\tau^{-1}(I,J)))) \leq \tau^{-1}(I,J)$ which gives $\Psi_{\mathsf{P}}(\tau^{-1}(I,J)) \leq \tau^{-1}(I,J)$. From the definition of Ψ_{P} and the definition of model, it follows that $\tau^{-1}(I,J)$ is model of P. $\square$

Lemma 5
Let $\mathcal{M} \in \mathcal{H}_{\mathsf{P}}$ be a model of P. Then, $\tau(\mathcal{M})$ is a pre-fixpoint of T_{P}.

Proof
By the definition of Ψ_{P} we have that for every predicate constant p in P, $\Psi_{\mathsf{P}}(\mathcal{M})(\mathsf{p}) = \bigvee_{\leq}\{[\![\mathsf{E}]\!](\mathcal{M}) \mid (\mathsf{p} \leftarrow \mathsf{E}) \in \mathsf{P}\}$. Since $\mathcal{M}$ is a model of P it follows that $[\![\mathsf{E}]\!](\mathcal{M}) \leq \mathcal{M}(\mathsf{p})$ for every clause $\mathsf{p} \leftarrow \mathsf{E}$ in P, i.e., $\mathcal{M}(\mathsf{p})$ is an upper bound of the set $\{[\![\mathsf{E}]\!](\mathcal{M}) \mid (\mathsf{p} \leftarrow \mathsf{E}) \in \mathsf{P}\}$. Therefore, $\bigvee_{\leq}\{[\![\mathsf{E}]\!](\mathcal{M}) \mid (\mathsf{p} \leftarrow \mathsf{E}) \in \mathsf{P}\} \leq \mathcal{M}(\mathsf{p})$, which implies that $\Psi_{\mathsf{P}}(\mathcal{M}) \leq \mathcal{M}$. By Proposition 14 it follows that $\tau(\Psi_{\mathsf{P}}(\mathcal{M})) \leq \tau(\mathcal{M})$. Moreover, by the definition of T_{P} and Proposition 14 we have that $T_{\mathsf{P}}(\tau(\mathcal{M})) = \tau(\Psi_{\mathsf{P}}(\tau^{-1}(\tau(\mathcal{M})))) = \tau(\Psi_{\mathsf{P}}(M)) \leq \tau(\mathcal{M})$, and therefore $\tau(\mathcal{M})$ is a pre-fixpoint of T_{P}. $\square$

In order to establish Theorem 2 that follows, we need the following lemma:

Lemma 7
Let P be a program. If $(I,J) \in \mathcal{H}_{\mathsf{P}}^{\mathsf{ma}} \otimes \mathcal{H}_{\mathsf{P}}^{\mathsf{am}}$ is a minimal pre-fixpoint of T_{P} then $\tau^{-1}(I,J)$ is a minimal model of P.

Proof
Let $\mathcal{M} = \tau^{-1}(I,J)$. By Lemma 4, $\mathcal{M}$ is a model of P. Assume there exists a model $\mathcal{N} \in \mathcal{H}_{\mathsf{P}}$ of P such that $\mathcal{N} \leq \mathcal{M}$. Applying τ to both sides and using Proposition 14 we get that $\tau(\mathcal{N}) \leq \tau(\mathcal{M})$. By Lemma 5, $\tau(\mathcal{N})$ is a pre-fixpoint of T_{P} and since $\tau(\mathcal{M}) = (I,J)$ is a minimal pre-fixpoint of T_{P}, we get that $\tau(\mathcal{N}) = \tau(\mathcal{M})$. Applying τ^{-1} to both sides, we get $\mathcal{N} = \mathcal{M}$. $\square$

Theorem 2
Let P be a program. Then, $\mathcal{M}_{\mathsf{P}}$ is a $\leq$-minimal model of P.

Proof
By Theorem 6 (see Appendix C) every fixpoint of $\mathcal{C}_{T_{\mathsf{P}}}$ is a *minimal pre-fixpoint* of T_{P}. Since by Theorem 1 $(I_\delta, J_\delta) = \tau(\mathcal{M}_{\mathsf{P}})$ is a fixpoint of $\mathcal{C}_{T_{\mathsf{P}}}$, $\tau(\mathcal{M}_{\mathsf{P}})$ is a minimal pre-fixpoint of T_{P}. By Lemma 7, $\tau^{-1}(\tau(\mathcal{M}_{\mathsf{P}})) = \mathcal{M}_{\mathsf{P}}$ is a minimal model of P. $\square$

Theorem 3
For every propositional program P, $\mathcal{M}_{\mathsf{P}}$ coincides with the well-founded model of P.

Proof
In (Denecker et al. 2004)[Section 6, pages 107-108], the well-founded semantics of propositional logic programs (allowing arbitrary nesting of conjunction, disjunction and negation in clause bodies) is derived. By a careful inspection of the steps used in the above reference, it can be seen that the construction given therein is a special case of the technique used in the present paper. $\square$

Appendix E: The Model $\mathcal{M}_{\mathsf{P}}$ for an Example Program

Consider the following program P which is a simplified non-recursive version of a program taken from (Rondogiannis and Symeonidou 2017). Initially we use a Prolog-like syntax:

$$\begin{array}{l}\texttt{s(Q,V)} \leftarrow \texttt{Q(V)}\\ \texttt{p(R)} \leftarrow \texttt{R}\\ \texttt{q(R)} \leftarrow \sim \texttt{w(R)}\\ \texttt{w(R)} \leftarrow \sim \texttt{R}\end{array}$$

In the above example, the type of `p`, `q` and `w` is $o \to o$, and the type of `s` is $(o \to o) \to o \to o$. In $\mathcal{HOL}$ notation the program can be written as follows:

$$\begin{array}{l}\texttt{s} \leftarrow \lambda\texttt{Q}.\lambda\texttt{V}.\texttt{(Q V)}\\ \texttt{p} \leftarrow \lambda\texttt{R}.\texttt{R}\\ \texttt{q} \leftarrow \lambda\texttt{R}.\sim \texttt{(w R)}\\ \texttt{w} \leftarrow \lambda\texttt{R}.(\sim \texttt{R})\end{array}$$

Notice now that the bodies of the clauses of `s`, `q` and `w` do not involve other predicate constants, and therefore the calculation of their meaning can be performed in a more direct way. On the other hand, the body of the clause concerning `q` involves the predicate constant `w`, and therefore the calculation of the meaning of `q` is more involved.

The first approximation to the well-founded model of P is the pair $(I_0, J_0) = (\bot, \top)$ (see Theorem 1). Consider now (I_1, J_1). We have:

$$I_1 = \mathit{lfp}([T_{\mathsf{P}}(\cdot, \top)]_1) = \mathit{lfp}([\tau(\Psi_{\mathsf{P}}(\tau^{-1}(\cdot, \top)))]_1)$$

and

$$J_1 = \mathit{lfp}([T_{\mathsf{P}}(\bot, \cdot)]_2) = \mathit{lfp}([\tau(\Psi_{\mathsf{P}}(\tau^{-1}(\bot, \cdot)))]_2)$$

where, as discussed in Appendix C, the *lfp* in the case of I_1 is the least upper bound of the sequence $I_1^0, I_1^1, \ldots$, defined as follows:

$$\begin{array}{rcl} I_1^0 & = & [\tau(\Psi_{\mathsf{P}}(\tau^{-1}(\bot, \top)))]_1\\ I_1^1 & = & [\tau(\Psi_{\mathsf{P}}(\tau^{-1}(I_1^0, \top)))]_1\\ & \cdots & \\ I_1^{\alpha+1} & = & [\tau(\Psi_{\mathsf{P}}(\tau^{-1}(I_1^\alpha, \top)))]_1\\ & \cdots & \end{array}$$

and the *lfp* in the case of J_1 is the least upper bound of the sequence $J_1^0, J_1^1, \ldots$, defined as follows:

$$\begin{array}{rcl} J_1^0 & = & [\tau(\Psi_{\mathsf{P}}(\tau^{-1}(\bot, \bot)))]_2\\ J_1^1 & = & [\tau(\Psi_{\mathsf{P}}(\tau^{-1}(\bot, J_1^0)))]_2\\ & \cdots & \\ J_1^{\alpha+1} & = & [\tau(\Psi_{\mathsf{P}}(\tau^{-1}(\bot, J_1^\alpha)))]_2\\ & \cdots & \end{array}$$

For the predicate constant w we have:

$$
\begin{array}{l}
I_1^0(\mathtt{w}) = [\tau(\Psi_{\mathsf{P}}(\tau^{-1}(\bot,\top)))]_1(\mathtt{w}) = [\tau([\![\lambda\mathtt{R}.\sim \mathtt{R}]\!](\tau^{-1}(\bot,\top)))]_1 = [\tau(\lambda v.v^{-1})]_1\\
I_1^1(\mathtt{w}) = [\tau(\Psi_{\mathsf{P}}(\tau^{-1}(I_1^0,\top)))]_1(\mathtt{w}) = [\tau([\![\lambda\mathtt{R}.\sim \mathtt{R}]\!](\tau^{-1}(I_1^0,\top)))]_1 = [\tau(\lambda v.v^{-1})]_1\\
\qquad\qquad\cdots\\
I_1^{\alpha+1}(\mathtt{w}) = [\tau(\Psi_{\mathsf{P}}(\tau^{-1}(I_1^\alpha,\top)))]_1(\mathtt{w}) = [\tau([\![\lambda\mathtt{R}.\sim \mathtt{R}]\!](\tau^{-1}(I_1^\alpha,\top)))]_1 = [\tau(\lambda v.v^{-1})]_1\\
\qquad\qquad\cdots
\end{array}
$$

Similarly, we can show that for every ordinal α, $J_1^\alpha(\mathtt{w}) = [\tau(\lambda v.v^{-1})]_2$. The above imply that $\mathcal{M}_{\mathsf{P}}(\mathtt{w}) = \lambda v.v^{-1}$. In other words, the denotation of w is the *not* function over our 3-valued truth domain. In a similar way, it follows that $\mathcal{M}_{\mathsf{P}}(\mathtt{p}) = \lambda v.v$. In other words, the denotation of p is the identity function over our 3-valued domain.

Consider now the predicate constant q. We have:

$$
\begin{array}{l}
I_1^0(\mathtt{q}) = [\tau(\Psi_{\mathsf{P}}(\tau^{-1}(\bot,\top)))]_1(\mathtt{q}) = [\tau([\![\lambda\mathtt{R}.\sim\!(\mathtt{w\ R})]\!](\tau^{-1}(\bot,\top)))]_1 = [\tau(\lambda v.\mathit{undef})]_1\\
I_1^1(\mathtt{q}) = [\tau(\Psi_{\mathsf{P}}(\tau^{-1}(I_1^0,\top)))]_1(\mathtt{q}) = [\tau([\![\lambda\mathtt{R}.\sim\!(\mathtt{w\ R})]\!](\tau^{-1}(I_1^0,\top)))]_1 = [\tau(f)]_1\\
\qquad\qquad\cdots\\
I_1^{\alpha+1}(\mathtt{q}) = [\tau(\Psi_{\mathsf{P}}(\tau^{-1}(I_1^\alpha,\top)))]_1(\mathtt{q}) = [\tau([\![\lambda\mathtt{R}.\sim\!(\mathtt{w\ R})]\!](\tau^{-1}(I_1^\alpha,\top)))]_1 = [\tau(f)]_1\\
\qquad\qquad\cdots
\end{array}
$$

where f is the function such that $f(\mathit{true}) = f(\mathit{undef}) = \mathit{undef}$ and $f(\mathit{false}) = \mathit{false}$. Similarly, we have:

$$
\begin{array}{l}
J_1^0(\mathtt{q}) = [\tau(\Psi_{\mathsf{P}}(\tau^{-1}(\bot,\bot)))]_2(\mathtt{q}) = [\tau([\![\lambda\mathtt{R}.\sim\!(\mathtt{w\ R})]\!](\tau^{-1}(\bot,\bot)))]_2 = [\tau(\lambda v.\mathit{true})]_2\\
J_1^1(\mathtt{q}) = [\tau(\Psi_{\mathsf{P}}(\tau^{-1}(\bot,J_1^0)))]_2(\mathtt{q}) = [\tau([\![\lambda\mathtt{R}.\sim\!(\mathtt{w\ R})]\!](\tau^{-1}(\bot,J_1^0)))]_2 = [\tau(g)]_2\\
\qquad\qquad\cdots\\
J_1^{\alpha+1}(\mathtt{q}) = [\tau(\Psi_{\mathsf{P}}(\tau^{-1}(\bot,J_1^\alpha)))]_2(\mathtt{q}) = [\tau([\![\lambda\mathtt{R}.\sim\!(\mathtt{w\ R})]\!](\tau^{-1}(\bot,J_1^\alpha)))]_2 = [\tau(g)]_2\\
\qquad\qquad\cdots
\end{array}
$$

where g is the function such that $g(\mathit{false}) = g(\mathit{undef}) = \mathit{undef}$ and $g(\mathit{true}) = \mathit{true}$.

Consider now (I_2, J_2). We have:

$$I_2 = \mathit{lfp}([T_{\mathsf{P}}(\cdot, J_1)]_1) = \mathit{lfp}([\tau(\Psi_{\mathsf{P}}(\tau^{-1}(\cdot, J_1)))]_1)$$

and

$$J_2 = \mathit{lfp}([T_{\mathsf{P}}(I_1, \cdot)]_2) = \mathit{lfp}([\tau(\Psi_{\mathsf{P}}(\tau^{-1}(I_1, \cdot)))]_2)$$

where the *lfp* in the case of I_2 is the least upper bound of the sequence $I_2^0, I_2^1, \ldots$ defined as follows:

$$
\begin{array}{lcl}
I_2^0 & = & [\tau(\Psi_{\mathsf{P}}(\tau^{-1}(\bot, J_1)))]_1\\
I_2^1 & = & [\tau(\Psi_{\mathsf{P}}(\tau^{-1}(I_2^0, J_1)))]_1\\
 & \cdots & \\
I_2^{\alpha+1} & = & [\tau(\Psi_{\mathsf{P}}(\tau^{-1}(I_2^\alpha, J_1)))]_1\\
 & \cdots &
\end{array}
$$

and the *lfp* in the case of J_2 is the least upper bound of the sequence $J_2^0, J_2^1, \ldots$ defined as follows:

$$
\begin{array}{lcl}
J_2^0 & = & [\tau(\Psi_{\mathsf{P}}(\tau^{-1}(I_1, I_1^*)))]_2\\
J_2^1 & = & [\tau(\Psi_{\mathsf{P}}(\tau^{-1}(I_1, J_2^0)))]_2\\
 & \cdots & \\
J_2^{\alpha+1} & = & [\tau(\Psi_{\mathsf{P}}(\tau^{-1}(I_1, J_2^\alpha)))]_2\\
 & \cdots &
\end{array}
$$

where I_1^* is the least interpretation in $\mathcal{H}_{\mathsf{P}}^{\mathsf{am}}$ such that $I_1 \leq I_1^*$ (namely, the bottom antimonotone-monotone element of the interval $[I_1, \perp]$, see the construction in Appendix C).

Consider again the predicate constant $\mathtt{q}$. We have:

$$
\begin{aligned}
I_2^0(\mathtt{q}) &= [\tau(\Psi_{\mathsf{P}}(\tau^{-1}(\perp, J_1)))]_1(\mathtt{q}) = [\tau([\![\lambda\mathtt{R}.\sim\!(\mathtt{w}\ \mathtt{R})]\!](\tau^{-1}(\perp, J_1)))]_1 \\
I_2^1(\mathtt{q}) &= [\tau(\Psi_{\mathsf{P}}(\tau^{-1}(I_2^0, J_1)))]_1(\mathtt{q}) = [\tau([\![\lambda\mathtt{R}.\sim\!(\mathtt{w}\ \mathtt{R})]\!](\tau^{-1}(I_2^0, J_1)))]_1 = [\tau(\lambda v.v)]_1 \\
&\dots \\
I_2^{\alpha+1}(\mathtt{q}) &= [\tau(\Psi_{\mathsf{P}}(\tau^{-1}(I_2^\alpha, J_1)))]_1(\mathtt{q}) = [\tau([\![\lambda\mathtt{R}.\sim\!(\mathtt{w}\ \mathtt{R})]\!](\tau^{-1}(I_2^\alpha, J_1)))]_1 = [\tau(\lambda v.v)]_1 \\
&\dots
\end{aligned}
$$

because for all ordinals α, $I_2^\alpha(\mathtt{w}) = [\tau(\lambda v.v^{-1})]_1$ and $J_1(\mathtt{w}) = [\tau(\lambda v.v^{-1})]_2$. Similarly, we have:

$$
\begin{aligned}
J_2^0(\mathtt{q}) &= [\tau(\Psi_{\mathsf{P}}(\tau^{-1}(I_1, I_1^*)))]_2(\mathtt{q}) = [\tau([\![\lambda\mathtt{R}.\sim\!(\mathtt{w}\ \mathtt{R})]\!](\tau^{-1}(I_1, I_1^*)))]_2 \\
J_2^1(\mathtt{q}) &= [\tau(\Psi_{\mathsf{P}}(\tau^{-1}(I_1, J_2^0)))]_2(\mathtt{q}) = [\tau([\![\lambda\mathtt{R}.\sim\!(\mathtt{w}\ \mathtt{R})]\!](\tau^{-1}(I_1, J_2^0)))]_2 = [\tau(\lambda v.v)]_2 \\
&\dots \\
J_2^{\alpha+1}(\mathtt{q}) &= [\tau(\Psi_{\mathsf{P}}(\tau^{-1}(I_1, J_2^\alpha)))]_2(\mathtt{q}) = [\tau([\![\lambda\mathtt{R}.\sim\!(\mathtt{w}\ \mathtt{R})]\!](\tau^{-1}(I_1, J_2^\alpha)))]_2 = [\tau(\lambda v.v)]_2 \\
&\dots
\end{aligned}
$$

because $I_1(\mathtt{w}) = [\tau(\lambda v.v^{-1})]_1$ and for all ordinals α, $J_2^\alpha(\mathtt{w}) = [\tau(\lambda v.v^{-1})]_2$. The above imply that $\mathcal{M}_{\mathsf{P}}(\mathtt{q}) = \lambda v.v$. In other words, the denotation of $\mathtt{q}$ is the identity function over our 3-valued truth domain. Notice that despite their different definitions, $\mathtt{p}$ and $\mathtt{q}$ denote the same 3-valued relation (in some sense, the two negations in the definition of $\mathtt{q}$ cancel each other).

Finally, consider the predicate constant $\mathtt{s}$. We have:

$$
\begin{aligned}
I_1^0(\mathtt{s}) &= [\tau([\![\lambda\mathtt{Q}.\lambda\mathtt{V}.(\mathtt{Q}\ \mathtt{V})]\!](\tau^{-1}(\perp, J_1)))]_1 = [\tau(\lambda q.\lambda v.(q\,v))]_1 \\
I_1^1(\mathtt{s}) &= [\tau([\![\lambda\mathtt{Q}.\lambda\mathtt{V}.(\mathtt{Q}\ \mathtt{V})]\!](\tau^{-1}(I_1^0, \top)))]_1 = [\tau(\lambda q.\lambda v.(q\,v))]_1 \\
&\dots \\
I_1^{\alpha+1}(\mathtt{s}) &= [\tau([\![\lambda\mathtt{Q}.\lambda\mathtt{V}.(\mathtt{Q}\ \mathtt{V})]\!](\tau^{-1}(I_1^\alpha, \top)))]_1 = [\tau(\lambda q.\lambda v.(q\,v))]_1 \\
&\dots
\end{aligned}
$$

and also:

$$
\begin{aligned}
J_1^0(\mathtt{s}) &= [\tau([\![\lambda\mathtt{Q}.\lambda\mathtt{V}.(\mathtt{Q}\ \mathtt{V})]\!](\tau^{-1}(I_1, \perp)))]_2 = [\tau(\lambda q.\lambda v.(q\,v))]_2 \\
J_1^1(\mathtt{s}) &= [\tau([\![\lambda\mathtt{Q}.\lambda\mathtt{V}.(\mathtt{Q}\ \mathtt{V})]\!](\tau^{-1}(\perp, J_1^0)))]_2 = [\tau(\lambda q.\lambda v.(q\,v))]_2 \\
&\dots \\
J_1^{\alpha+1}(\mathtt{s}) &= [\tau([\![\lambda\mathtt{Q}.\lambda\mathtt{V}.(\mathtt{Q}\ \mathtt{V})]\!](\tau^{-1}(\perp, J_1^\alpha)))]_2 = [\tau(\lambda q.\lambda v.(q\,v))]_2 \\
&\dots
\end{aligned}
$$

The above imply that $\mathcal{M}_{\mathsf{P}}(\mathtt{s}) = \lambda q.\lambda v.(q\,v)$.