5-1992

General Model Theoretic Semantics for Higher-Order Horn Logic Programming

Mino Bai

Howard A. Blair
Syracuse University, School of Computer and Information Science, blair@top.cis.syr.edu

SU-CIS-92-09

General Model Theoretic Semantics for Higher-Order Horn Logic Programming

Mino Bai and Howard A. Blair

May 1992

School of Computer and Information Science
Syracuse University
Suite 4-116, Center for Science and Technology
Syracuse, NY 13244-4100

General Model Theoretic Semantics for Higher-Order Horn Logic Programming*

Mino Bai** Howard A. Blair***

School of Computer and Information Science
Syracuse University, Syracuse, NY 13244, USA

Abstract. We introduce model-theoretic semantics [6] for Higher-Order Horn logic programming language. One advantage of logic programs over conventional non-logic programs has been that the least fixpoint is equal to the least model, therefore it is associated to logical consequence and has a meaningful declarative interpretation. In simple theory of types [9] on which Higher-Order Horn logic programming language is based, domain is dependent on interpretation [10]. To define $T_{\mathcal{P}}$ operator for a logic program $\mathcal{P}$, we need a fixed domain without regard to interpretation which is usually taken to be a set of atomic propositions. We build a semantics where we can fix a domain while changing interpretations. We also develop a fixpoint semantics based on our model, and show that we can get the least fixpoint which is the least model. Using this fixpoint we prove the completeness of the interpreter of our language in [14].

1 Introduction

Many extended versions of Prolog are developed which incorporate higher-order features in logic programming languages to make programs more versatile and expressive [16, 8, 1]. In this paper, we build a model-theoretic semantics for a higher-order logic programming language which is suitable for describing declaratively operations of such programming language.

Church [9] introduced a simple theory of types as a system of higher-order logic. This system incorporated λ-notation in its particularly simple syntax which actually be viewed as a version of simply typed λ-calculus. Henkin first gave a semantics for Church's system based on general models. Domain members of a general model are truth values, individuals, and functions. Church's system was proved to be complete with respect to Henkin's semantics [10]. Andrews studied general models further in [3, 4, 5], and built a non-extensional model which is suitable under settings of resolution theorem proving [2]. The proof theory for this system is shown to have a close resemblance to that of first-order logic: there is, for example, a generalization to Herbrand theorem that holds for a variant of this system [12, 13].

λProlog [16] was the first language to show that higher-order logic could be used as the basis of a practical programming language. λProlog is based on typed λ-calculi which have their ultimate origin in Russel's method of stratifying sets to avoid the set theoretic paradoxes. One advantage of logic programs over coventional non-logic programs has been that they have simple declarative model-theoretic semantics. That is, in logic programs the least fixpoint is equal to least model,

* To appear in the *Proceedings of the Third Russian Conference on Logic Programming and Automated Reasoning*, July 1992, edited by A. Voronkov, Lecture Notes in Artificial Intelligence, Springer-Verlag

** *Address correspondence to* authors, School of Computer and Information Science, Center for Science and Technology/Fourth Floor, Syracuse University, Syracuse, New York 13244-4100, USA, *Telephone number of* Mino Bai, 315-443-5812, *Email address of* Mino Bai, mbai@top.cis.syr.edu

*** *Email address of* Howard Blair, blair@top.cis.syr.edu, *Telephone number of* Howard Blair 315-443-2368

therefore it is associated to logical consequences and has a meaninful declarative interpretation. In higher-order logic on which λProlog is based, compared to first-order case, it is extremely difficult to build an effective model-theoretic semantics. One of these difficulties is that the definition of satisfaction of formulas is mutually recursive with the process of evaluation of terms (see [10, 2, 3, 4, 5]). In first-order case, the model-theory is two level [11]. First we define a domain of individuals, and then define satisfaction wrt this domain. As a result of this in higher-order logic it is difficult to define $T_{\mathcal{P}}$ operator for a logic program $\mathcal{P}$: In a definition of $T_{\mathcal{P}}$ operator for a logic program $\mathcal{P}$, we consider a set of atomic propositions as an interpretation, and need a fixed domain without regard to interpretations. The second reason is that since higher-order logic programming languages are usually formulated in *non-extensional* form, we need a non-extensional model to describe properly such languages.

Henkin's general model semantics is extensional: i.e., if two objects in a model have the same extension, then they must be equal. Extensional models are very difficult to deal with, and unsuitable to describe a higher-order logic programming language like λProlog which contain a propositional type in its primitive set of types. For example, we can define a program $\mathcal{P}_1 = \{p(a) \leftarrow \top, q(a) \leftarrow \top, r(p(a)) \leftarrow \top\}$ in λProlog. Given program $\mathcal{P}_1$, the goal $r(p(a))$ will succeed in λProlog, but the goal $r(q(a))$ will fail, since the unification of $r(q(a))$ and $r(p(a))$ will simply fail. For any extensional model $\mathcal{M}$ for $\mathcal{P}_1$, $\mathcal{M}$ will assign the value **T** for $p(a)$ and $q(a)$. So $p(a) = q(a)$ is a logical consequence of $\mathcal{P}_1$. $\mathcal{M}$ will also assign the value **T** to $r(p(a))$, so the extension of the predicate which $\mathcal{M}$ will assign to r contains **T**. Therefore $r(q(a))$ is a logical consequence of the program $\mathcal{P}_1$. Note that for this program the valuation of terms is mutually recursive with the satisfaction of formulas, since a formula can occur as an argument of predicate or functional symbols.

As shown above extensional models are difficult to define and unsuitable for higher-order logic programming. In this paper, we develop a non-extensional model where domain is independent from interpretations and build a fixed point semantics, and we prove the completeness of the interpreter in [14].

2 Higher-Order Horn Logic Programming Language

In this section we describe a higher-order logic programming language for which we build models in the later sections. For the exposition of our logic programming language $\mathcal{L}$ we will follow closely those in [16, 15].

The set $\mathcal{T}$ of types contains a collection $\mathcal{T}_0$ of primitive types and is closed under the formation of functional types: i.e., if $\alpha, \beta \in \mathcal{T}$, then $(\alpha \rightarrow \beta) \in \mathcal{T}$. The type constructor $\rightarrow$ associates to the right. The type $(\alpha \rightarrow \beta)$ is that of a function from objects of type α to objects of type β.

We introduce a very convenient notation from [17]. For each type symbol α, and each set S containing objects or expressions, we write S_α to denote the *set of things in S which are of type α*. We sometimes write $\{S_\alpha\}_\alpha$ to denote S. We can also define a *type assignment mapping* τ on the set S such that $\tau : S \rightarrow \mathcal{T}$ and for all $s \in S$, $\tau(s) = \alpha$ if $s \in S_\alpha$.

Let S, T, T_1, T_2 be sets. Given a mapping $f : S \rightarrow T$, $a \in S$, and $b \in T$, let $f[b/a]$ be that mapping $f' : S \rightarrow T$ such that for $f'a = b$ and $f'c = fc$ for all $c \neq a$. Let b be an element in $T_1 \times T_2$, then b^1 and b^2 are the first and second components of b, so $b = \langle b^1, b^2 \rangle$. If f is a mapping whose values are in $T_1 \times T_2$, let f^1 and f^2 be mappings with the same domain as f defined so that for any argument t, $f^i t = (ft)^i$ for $i = 1, 2$. Thus $ft = \langle f^1 t, f^2 t \rangle$. If $f : S \rightarrow T$ is a mapping, then we say that f is *type consistent* if for all $s \in S$, $\tau(f(s)) = \tau(s)$. If $f : S \rightarrow T_1 \times T_2$, then we say that f is *type consistent* if f^1 and f^2 are type consistent. For each integer $n \in \omega$, we write $[n]$ for the set $\{1, \cdots, n\}$.

We assume that there are denumerably many variables and constants of each type. Let the set of variables and constants be Δ and Σ, respectively. *Simply typed λ-terms* are built up in the usual fashion from these typed constants and variables via abstraction and application. Our *well formed terms* (wfts) are simply typed λ-terms. We, as usual, can define the set $T(\Sigma)$ of all wfts by giving the definition of the set $T(\Sigma)_\alpha$ of wfts of type α by induction.

It is assumed that the reader is familiar with most of basic notions and definitions such as bound, free variables, closed terms (c-terms), substitution and λ-conversion for this language; only a few are reviewed here. Letters $f_\alpha, s_\alpha, t_\alpha, \cdots$, will be used as syntactical variables of wfts of type α. Type subscript symbols may be omitted when context indicates what they should be or irrelevant to discussion. By Church-Rosser theorem [7], a λ-normal wfts of a wft is unique upto a renaming of variables. For most part we shall be satisfied with any of these normal forms corresponding to a wft t, and we shall write $\lambda norm(t)$ to denote such a form. In certain situations we shall need to talk about a unique normal form and, in such cases, we shall use $\rho(t)$ to designate what we shall call the *principal normal* or *ρ-normal* form of t; i.e. ρ is a mapping from wfts to λ-normal terms. There are several schemes that may be used to pick a representative of the α-equivalence classes of λ-normal terms and the one implicitly assumed here is that of [2].

So far we have introduced λ-term structures and operations on λ-terms. We can introduce logic into λ-term structures by including o, a type for propositions, amongst the set of primitive types $\mathcal{T}_0$, and requiring that the collection Σ of constants contain the following *logical* constants: $\wedge$ and $\vee$ of type $o \rightarrow o \rightarrow o$; $\top$ of type o; and for every type α, $\exists_\alpha$ of type $(\alpha \rightarrow o) \rightarrow o$. The constants in Σ other than $\wedge, \vee, \exists$ and $\top$ are called as *non-logical* constants. A type will be called a *predicate type* if it is a type of the form $\alpha_1 \rightarrow \cdots \alpha_n \rightarrow o$, or a *non-predicate type* otherwise. We let $\Pi \subseteq \Sigma$ be the *set of predicate constants.* Expression of the form $\exists(\lambda x G)$ will be abbreviated by $\exists x G$.

Terms of type o are referred to as *goal formula.* The λ-normal form of a goal formula consists, at the outermost level, of a sequence of applications, and the leftmost symbol in this sequence is called its *top level symbol.* We shall have use for the structure of λ-normal formulas that is described below. A goal formula is said to be an *atom* (*atomic*) if its leftmost symbol that is not a bracket is either a predicate variable or constant. A λ-normal goal formula G, then, has the following inductive characterization: (a) it is $\top$, (b) it is an atom, (c) it is $G_1 \wedge G_2$ or $G_1 \vee G_2$, where G_1 and G_2 are λ-normal goal formulas, or (d) it is $\exists x G$, where G is a λ-normal goal formula.

Now we identify the formulas that we call higher-order definite clauses, goal formula, and equations. Let $\mathcal{G}$ be the collection of all λ-normal goal formulas. An *atom* is an atomic goal formula A. A *rigid atom* is an atom A_r that has a predicate constant as its head. An atom is thus a formula of the form $pt_1 \cdots t_n$ where $\gamma = \alpha_1, \cdots, \alpha_n \rightarrow o$, p is a predicate constant$_\gamma$ or variable$_\gamma$, and, for each $i \in [n]$, t_i is a λ-normal term$_{\alpha_i}$, it is a rigid atom just in case p is a constant. Sometimes we write $p(t_1, \cdots, t_n)$ or $p(\overline{t})$ for the above atom. Let G be an arbitrary goal formula and A_r be any rigid atom. Let a formula C be of the form $A_r \leftarrow G$. Then C is a *(higher-order) definite clause.* Let $s_\alpha, t_\alpha \in T(\Sigma)$. Then, as usual, an *equation* e is of the form $s_\alpha = t_\alpha$, and an *extensional equation* is of the form $s_\alpha \equiv t_\alpha$. Let $\mathcal{D}ef$ be the set of all definite clauses. Then given the collection Σ of constants, our *logic programming language* $\mathcal{L} = \mathcal{L}(\Sigma)$ is completely determined as the triple $\langle T(\Sigma), \mathcal{G}, \mathcal{D}ef \rangle$. A formula F in a language $\mathcal{L}$ is a goal formula, or a definite clause, or an equation. We refer a set $\mathcal{P}$ of formulas from $\mathcal{D}ef$ as a *higher-order definite logic program.* As usual, variables in definite clauses are implicitly universally quantified. Note that in the above definition all wfts in $T(\Sigma)$ do not contain such symbols as $=, \equiv, \leftarrow$, hence a goal formula G and s_α and t_α in an equation $s_\alpha = t_\alpha$ do not contain those symbols.

We say that a predicate symbol p *occurs extensionally in* a goal formula G if (a) G is $p(\overline{t})$, or (b) G is $G_1 \wedge G_2$ or $G_1 \vee G_2$, or and p occurs extensionally in G_1 or G_2, or (c) G is $\exists x G_1$, and p occurs extensionally in G_1. In following sections, we will define semantics for λProlog. We will take

advantage of the following situation: Since logic programs compute extensions of predicates, and relations between arguments of predicate symbols constitute extensions of predicates, we don't need extensions of terms until we meet extensional occurrences of predicate symbols in the definition of satisfaction of formulas.

3 General Model Theoretic Semantics

In this Section we build model-theoretic semantics for the language $\mathcal{L}$. As introduced in Section 1 we need a non-extensional model to prove that a resolution system in type theory is complete. The model in [2] is in a sense non-extensional. But it doesn't provide an adequate notion of "general" non-extensional model for our purpose: Domain is defined by indexing extension of the element in it by wfts. The indexed entity like $\langle t, p \rangle$ is called a V-complexe where V is a truth value evaluation of formulas. So only one kind of domain is used in [2], since the set of all wfts is predetermined given a language $\mathcal{L}$. In [2], in order to define the domain of interpretation we need a semivaluation function V, as above, which evaluates proposional formulas to **T** or **F**. The definition of domain or the evaluation of terms is mutually recursive with the definition of evaluation of formulas.

Now we generalize Andrews model to a model where we index the extension by an element from a general domain which we call frame. From this model we build a model where the definition of domain is independent from the definition of satisfaction. These two models will be shown to be isomorphic and elementarily equivalent in the sense that the sets of valid sentences in each semantics are same. Since our language $\mathcal{L}$ is based on λ-calculus and application is a basic operation of the λ-calculus, any model of $\mathcal{L}$ should be an applicative structure which is a λ-model.

Definition Let A be a set and $\cdot$ a binary operation over A such that for all $\alpha, \beta \in \mathcal{T}$, for all $a \in A_{\alpha\to\beta}, b \in A_\alpha$, $a \cdot b$ is an element in A_β. Then $\mathcal{A} = \langle A, \cdot \rangle$ is said to be an *applicative structure*. An *assignment into* a set A is a type consistent mapping $\varphi : \Delta \to A$. A *λ-model* is a triple $\langle A, \cdot, \|\cdot\| \rangle$ such that $\langle A, \cdot \rangle$ is an applicative structure and $\|\cdot\|$ a binary function such that for each assignment φ into A and term t_α, $\|t_\alpha\|_\varphi \in A_\alpha$ and for all terms $f \in T(\Sigma)_{\alpha\to\beta}$ and $t \in T(\Sigma)_\alpha$, $\|ft\|_\varphi = \|f\|_\varphi \cdot \|t\|_\varphi$. We call the function $\|\cdot\|$ a *valuation function* in A. □

Note that in the usual definition of extensional λ-model, we need one more condition, which can be expressed as for all term t, variable x_α, $\|\lambda x_\alpha t\|_\varphi = \lambda a \in A_\alpha \cdot \|t\|_{\varphi[a/x_\alpha]}$. This condition is equivalent to extensionality. Since we want non-extensional model, we do not include this condition for λ-model.

A *frame* is a nonempty set D of objects each of which is assigned a type symbol from the set $\mathcal{T}$ in such a way that every object in $D_{\alpha\to\beta}$ is a function from D_α to D_β for all type symbols α and β. A *pre-interpretation* $\mathcal{F}$ of the language $\mathcal{L}$ is a pair $\langle D, J \rangle$ where D is a frame, and J is a type consistent mapping in $\Sigma \to D$. An assignment into a pre-interpretation is an assignment into the frame of the pre-interpretation. Note that $D_{\alpha\to\beta}$ is some collection of functions mapping D_α into D_β, i.e. $D_{\alpha\to\beta} \subseteq D_\alpha \to D_\beta$. A pre-interpretation $\mathcal{F} = \langle D, J \rangle$ is said to be *general* iff there is a binary function $\mathsf{V}^{\mathcal{F}} = \mathsf{V}$ such that for each assignment φ and term t_α, $\mathsf{V}_\varphi t_\alpha \in D_\alpha$, and the following conditions are satisfied for each assignment φ and all terms: (a) if $x \in \Delta$, then $\mathsf{V}_\varphi x = \varphi x$. (b) if $c \in \Sigma$, then $\mathsf{V}_\varphi c = Jc$. (c) $\mathsf{V}_\varphi(ft) = (\mathsf{V}_\varphi f)\mathsf{V}_\varphi t$ (the value of the function $\mathsf{V}_\varphi f$ at the argument $\mathsf{V}_\varphi t$). (d) $\mathsf{V}_\varphi(\lambda x_\alpha t_\beta) = \lambda d \in D_\alpha \cdot \mathsf{V}_{\varphi[d/x]} t_\beta$ i.e. that function from D_α into D_β whose value for each argument $d \in D_\alpha$ is $\mathsf{V}_{\varphi[d/x]} t_\beta$.

If a pre-interpretation $\mathcal{F}$ is general, the function $\mathsf{V}^{\mathcal{F}}$ is uniquely determined. We can prove this by induction on the definition of terms. We call the unique function $\mathsf{V}^{\mathcal{F}}$ the *connotational valuation function* of terms in the pre-interpretation $\mathcal{F}$. $\mathsf{V}^{\mathcal{F}}_\varphi t$ is called the *connotation* of t in $\mathcal{F}$ wrt φ. We sometimes write $\mathsf{V}^{\mathcal{F}}_\varphi$ as V_φ, as $\mathsf{V}^{\mathcal{F}}$, or as V, when pre-interpretation or assignment is clear from

context, or irrelevant. It is clear that if t is a c-term, then $\mathsf{V}^{\mathcal{F}}t$ may be considered meaningful without regard to any assignment. In this case, $\mathsf{V}^{\mathcal{F}}t$ is called the *connotation* of t in $\mathcal{F}$ and written as t'. Obviously for a general frame D, $\langle D, \cdot, \mathsf{V}\rangle$ where $\cdot$ is interpreted as a functional application is a λ-model, but in a pre-interpretation logic symbols such as logical operators and predicate constants are not fully interpreted. So we call it a pre-interpretation.

Now we will give interpretations to logical symbols, after discussing a few constructions of posets. Any non-empty set A can be considered a poset under the identity relation where $x \subseteq_A y$ iff $x = y$. We call this type of poset *discrete*. Let P_1 and P_2 be disjoint posets. $P_1 \cup P_2$ is a poset $P = P_1 \cup P_2$ such that for all $x, y \in P$, $x \subseteq_P y$ if $x \subseteq_{P_1} y$ or $x \subseteq_{P_2} y$. $P_1 \times P_2$ is a poset $P = P_1 \times P_2$ where for all $x, y \in P$, $x \subseteq_P y$ if $x^1 \subseteq_{P_1} y^1$ and $x^2 \subseteq_{P_2} y^2$. Let S be a set, and P a poset. $S \to P$ is a poset F such that for all $f, g \in F$, $f \subseteq_F g$ if for all $s \in S$, $f(s) \subseteq_P g(s)$. Let $\mathcal{B}$ be the set of boolean values $\mathbf{T}$ and $\mathbf{F}$ where $\mathbf{F} \subseteq_{\mathcal{B}} \mathbf{T}$. We shall write $\vee$ and $\wedge$ for $\sqcup_{\mathcal{B}}$ and $\sqcap_{\mathcal{B}}$, respectively. Let A be a set. We can consider A a discrete poset. A *predicate* P *over* A *of type* $\alpha_1, \cdots, \alpha_n \to o$ is a mapping in $A_{\alpha_1} \times \cdots \times A_{\alpha_n} \to \mathcal{B}$, or equivalently a subset of $A_{\alpha_1} \times \cdots \times A_{\alpha_n}$. And we consider truth values $\mathbf{T}$ and $\mathbf{F}$ as *null-ary* predicates over A of type $() \to o$ such that $\mathbf{T}() \equiv \mathbf{T}$ and $\mathbf{F}() \equiv \mathbf{F}$, respectively. More generally, we define predicates $\mathbf{T}^A_{\alpha_1,\cdots,\alpha_n}$ for each list $\alpha_1, \cdots, \alpha_n$ of types where $n \geq 0$ as $A_{\alpha_1} \times \cdots \times A_{\alpha_n}$. We write $\Phi(A)$ for the *set of all predicates over* A. Given two predicates $P, Q \in \Phi(A)$, it is obvious that $P \subseteq Q$ if P and Q are of same type and P is a subset of Q.

Definition Let D be a frame. A *semivaluation* of D is a function V with domain D_o and range the set $\mathcal{B}$ of truth values such that the following properties hold: for all $c_o, d_o, f_{\alpha\to o} \in D$, (a) $V(\top') = \mathbf{T}$. (b) $V(\vee' c_o d_o) = V(c_o) \vee V(d_o)$. (c) $V(\wedge' c_o d_o) = V(c_o) \wedge V(d_o)$. (d) $V(\exists'_\alpha f_{\alpha\to o}) = \mathbf{T}$ iff there is some $e \in D_\alpha$ such that $V(f_{\alpha\to o} e) = \mathbf{T}$. Given a frame D and a semivaluation V of D, we define the *set* $\mathcal{D}$ *of* V*-complexes based on* D as follows: For each type γ we define the set $\mathcal{D}_\gamma$ of V-complexes$_\gamma$ and one-one onto mapping $\kappa_\gamma : D_\gamma \to \mathcal{D}_\gamma$ as follows by induction on γ: (a) $\mathcal{D}_o = \{\langle d, Vd\rangle : d \in D_o\}$. For $d \in D_o$, $\kappa_o d = \langle d, Vd\rangle$. (b) When $\alpha \in \mathcal{T}_0 - \{o\}$, $\mathcal{D}_\alpha = \{\langle d, d\rangle : d \in D_\alpha\}$. For $d \in D_\alpha$, $\kappa_\alpha d = \langle d, d\rangle$. (c) $\mathcal{D}_{\alpha\to\beta} = \{\langle f, \kappa_\alpha^{-1} \circ f \circ \kappa_\beta\rangle : f \in D_{\alpha\to\beta}\}$. For $f \in D_{\alpha\to\beta}$, $\kappa_{\alpha\to\beta} f = \langle f, \kappa_\alpha^{-1} \circ f \circ \kappa_\beta\rangle$. We say that $\mathcal{D}$ is the set of V-complexes *based on* D. We can also introduce one-one onto mapping $\kappa : D \to \mathcal{D}$ such that for $\alpha \in \mathcal{T}, d \in D_\alpha, \kappa d = \kappa_\alpha d$, and function v whose domain is D such that for $d \in D_\alpha$, $v(d) = (\kappa d)^2$. □

Now it is easy to see that (a) if $f \in D_{\alpha\to\beta}$, then $v(f) : \mathcal{D}_\alpha \to \mathcal{D}_\beta$, (b) for $a \in \mathcal{D}_\alpha$, $v(f)a = \langle fa^1, v(fa^1)\rangle$, and (c) $\mathcal{D} = \{\langle d, v(d)\rangle : d \in D\}$. And for any $a \in \mathcal{D}$, $\kappa a^1 = a$, and for any mapping χ whose values are in $\mathcal{D}$, $\chi^1 \circ \kappa = \chi$. Let $\mathcal{D}$ be a set of V-complexes. Then we define the *applicative operation* $\star$ of type $(\alpha \to \beta), \alpha \to \beta$: For $a \in \mathcal{D}_{\alpha\to\beta}$ and $b \in \mathcal{D}_\alpha$, $a \star b$ is defined to be $a^2 b$. The operation $\star$ is left associative. Let $a \in \mathcal{D}_{\alpha_1,\cdots,\alpha_n\to\beta}$ and $b_i \in \mathcal{D}_{\alpha_i}$ for $i \in [n]$. Then by definition of $\mathcal{D}$ it is easy to see that $a \star b_1 \star \cdots \star b_n \in \mathcal{D}_\beta$. Moreover, $\langle \mathcal{D}, \star\rangle$ is an applicative structure and for all $f \in D_{\alpha\to\beta}, d \in D_\alpha$, $(\kappa f) \star (\kappa d) = \kappa(fd)$.

Definition Let $\mathcal{D}$ be a set of V-complexes. We can define a binary mapping $\mathcal{V}$ such that for all assignment φ into $\mathcal{D}$, $\mathcal{V}_\varphi : T(\Sigma) \to \mathcal{D}$, and for all $t \in T(\Sigma)$, $\mathcal{V}^1_\varphi t = \mathsf{V}_{\varphi^1} t$. □

Let φ be an assignment into D. Then for all term t, $\kappa \mathsf{V}_\varphi t = \mathcal{V}_{\varphi\circ\kappa} t$. If φ is an assignment into $\mathcal{D}$, then for $a \in \mathcal{D}_\alpha$, $\mathcal{V}_\varphi(\lambda x_\alpha t) \star a = \mathcal{V}_{\varphi[a/x_\alpha]} t$. If D be a general frame and $\mathcal{D}$ a set of V-complexes, then there is the unique $\mathcal{V}$ satisfying that for all $t_\gamma \in T(\Sigma)$ and assignment φ into $\mathcal{D}$, $\mathcal{V}_\varphi t_\gamma \in \mathcal{D}_\gamma$, since the function V is unique. Therefore $\langle \mathcal{D}, \star, \mathcal{V}\rangle$ is a λ-model.

Definition Given a frame D, we define a *primitive extensional domain* E_α for $\alpha \in \mathcal{T}_0$: (a) $E_o = \mathcal{B}$. (b) $E_\alpha = D_\alpha$ for $\alpha \in \mathcal{T}_0 - \{o\}$. Given an $a \in \mathcal{D}_{\alpha_1,\cdots,\alpha_n\to\beta}$ where $n \geq 0$ and $\beta \in \mathcal{T}_0$, we define a mapping $a^\odot$ in $D_{\alpha_1} \to \cdots \to D_{\alpha_n} \to E_\beta$ by induction on n: (a) When $n = 0$, $a^\odot = a^2$. (b) When $n > 0$, $a^\odot = \lambda d_1 \in D_{\alpha_1} \cdot (a \star \kappa d_1)^\odot$. □

Let $a \in \mathcal{D}_{\alpha_1,\cdots,\alpha_n\to\beta}$ where $n > 0$ and $\beta \in \mathcal{T}_0$. Then (a) for all $d_i \in D_{\alpha_i}$, $i \in [n]$, $a^\odot d_1 \cdots d_n = (a \star \kappa d_1 \star \cdots \kappa d_n)^2$, (b) If $\beta \in \mathcal{T}_0 - \{o\}$, then $a^\odot = a^1$. We can show this by induction on n.

Definition [4] Let $\mathcal{F} = \langle D, J\rangle$ be a general pre-interpretation and V a semivaluation of D. An *$\mathcal{L}$-structure* $\mathcal{A}$ is a pair $\langle \mathcal{D}, J\rangle$ such that $\mathcal{D}$ is a set of V-complexes based on D. We say that $\mathcal{A}$ is *based on* $\mathcal{F}$ or *on* D. An *assignment φ into $\mathcal{A}$* is an assignment into $\mathcal{D}$. When F is a formula in $\mathcal{L}$, we write $\mathcal{A}\models F[\varphi]$ to say that $\mathcal{A}$ *satisfies* F wrt φ. (a) When $s_\alpha, t_\alpha \in T(\Sigma)$, $\mathcal{A}\models s_\alpha = t_\alpha[\varphi]$ iff $\mathcal{V}_\varphi s_\alpha = \mathcal{V}_\varphi t_\alpha$, $\mathcal{A}\models s_\alpha \equiv t_\alpha[\varphi]$ iff $(\mathcal{V}_\varphi s_\alpha)^\odot = (\mathcal{V}_\varphi t_\alpha)^\odot$. (b) When G is a goal formula, $\mathcal{A}\models G[\varphi]$ iff $\mathcal{V}^2_\varphi G = \mathbf{T}$. (c) When $A \leftarrow G$ is a definite clause, $\mathcal{A}\models A \leftarrow G[\varphi]$ iff $\mathcal{A}\models A[\varphi]$ whenever $\mathcal{A}\models G[\varphi]$. We write $\mathcal{A}\models F$ to say that a formula F is *valid* in $\mathcal{A}$ if $\mathcal{A}\models F[\varphi]$ for all assignments φ into $\mathcal{A}$. Given a set of definite clause $\mathcal{P}$, we say that $\mathcal{A}$ is a *model* or *D-model* for $\mathcal{P}$, and write $\mathcal{A}\models\mathcal{P}$, if each definite clause in $\mathcal{P}$ is valid in $\mathcal{A}$. Given a closed goal formula G, we say that G is a *logical consequence* of $\mathcal{P}$, and write $\mathcal{P}\models G$ if G is valid in all models of $\mathcal{P}$. □

Definition Let D be a general frame and S a subset of D_o. Then S is *upward saturated* if a) $\mathsf{T}' \in S$, b) $c \in S$ implies $\vee' cd, \vee' dc \in S$ for $d \in D_o$, c) $c, d \in S$ implies $\wedge' cd \in S$, and d) $f_{\alpha\to o} d_\alpha \in S$ implies $\exists'_\alpha f_{\alpha\to o} \in S$. □

Let $S \subseteq D_o$. Then there is a smallest upward saturated set extending S. Let $\mathcal{C}$ be the collection of upward saturated set extending S. $\mathcal{C}$ is not empty, since $D_o \in \mathcal{C}$. So $\cap\mathcal{C}$ exists. It is easy to check that it is upward saturated. It fulfills the other considerations, by definition. The smallest upward saturated set extending S is called the *upward saturated closure* of S, and is denoted as S^U.

If $S \subseteq D_o$ we can always find by the above method an extension of S which is saturated. The above definition is certainly simple, but it is unsatisfactory on several grounds. For example, it does not make explicit how the elements of the closure of S are generated from the elements of S. From this reason we give a more constructive definition, involving restricted set-theoretic methods.

Definition Let $S \subseteq D_o$. An *elementary S-derivation* is a sequence $c^1, \cdots, c^m$, $m \geq 1$, of elements from D_o, where for each $i \in [m]$, at least one of the following conditions is satisfied: (a) $c^i = \mathsf{T}'$. (b) $c^i \in S$. (c) There is a $j < i$ such that c^i is either $\vee' c^j d$ or $\vee' d c^j$ for some $d \in D_o$. (d) There are $j, k < i$ such that $c^i = \wedge' c^j c^k$. (e) There are $j < i$ and $f \in D_{\alpha\to o}$ such that $c^j = fd$ for some $d \in D_\alpha$ and $c^i = \exists'_\alpha f$. □

Note that if $c^1, \cdots, c^m$ and $d^1, \cdots, d^n$ are two elementary S-derivations, then the concatenation $c^1, \cdots, c^m, d^1, \cdots, d^n$ is also an elementary S-derivation. Furthermore, a nonempty initial segment of an elementary S-derivation is again an elementary S-derivation. An element $d \in D_o$ is *elementary S-derivable* if there is an elementary S-derivation $c^1, \cdots, c^m$ where $c^m = d$. This is equivalent to requiring that d be an element (not necessarily the last) in some elementary S-derivation. The set of all $d \in D_o$ that are elementary S-derivable is denoted by $\mathsf{E}(S)$. We shall show that $\mathsf{E}(S)$ is the upward closure of S referred to above.

Theorem 1. *Let $S \subseteq D_o$. Then: (a) $S \subseteq \mathsf{E}(S)$. (b) $\mathsf{E}(S)$ is upward saturated. (c) If $S \subseteq S'$ and S' is upward saturated, then $\mathsf{E}(S) \subseteq S'$. (d) $S^U = \mathsf{E}(S)$.*

Proof The proofs of (a) and (b) are obvious. (c) Let $S \subseteq S'$ and S' be upward saturated. We prove by induction on m that whenever $c^1, \cdots, c^m$ is an elementary S-derivation then $c^i \in S'$ for $i \in [m]$. When $m = 1$. it is clear. If the property is true for m, and $c^1, \cdots, c^m, c^{m+1}$ is an elmentary S-derivation, then by IH we have that $c^i \in S'$ for $i \in [m]$. Furthermore c^{m+1} is T', or a $c \in S \subseteq S'$, or it is obtained by one of the defining rules from the elements in S'. In all cases it is easy to see, by IH and definition of upward saturatedness, that $c^{m+1} \in S'$. □

Definition Let $\langle D, J\rangle$ be a general pre-interpretation. Then we write $\Pi(D)$ for the *D-base* which is defined to be the set $\{p(a_1, \cdots, a_n) : p \in \Pi_{\alpha_1,\cdots,\alpha_n\to o}$ and $a_i \in D_{\alpha_i}$ for all $i \in [n]\}$. □

[4] Note that in this definition the symbol for satisfaction in $\mathcal{A}$ is the small $\models$. The normal size $\models$ is used for another definition of satisfaction which is defined later in this paper.

A subset $\mathcal{K}$ of $\Pi(D)$ induces a unique mapping $I_{\mathcal{K}}$ in $\Pi \to \Phi(D)$ as follows: for all $\overline{d} \in D$, $\langle \overline{d} \rangle \in I_{\mathcal{K}}(p)$ iff $p(\overline{d}) \in \mathcal{K}$. Let $\mathcal{K}_1 \subseteq \mathcal{K}_2 \subseteq \Pi(D)$, then it is easy to see that $I_{\mathcal{K}_1} \subseteq_{\Pi \to \Phi(D)} I_{\mathcal{K}_2}$. Sometimes given $\mathcal{K} \subseteq \Pi(D)$, we write simply $\mathcal{K}$ to mean the mapping $I_{\mathcal{K}}$.

Given $I \subseteq \Pi(D)$, we can introduce set S_I such that $S_I = \{p'\overline{d} : p\overline{d} \in I\}$. We define a function $V_I : D_o \to \mathcal{B}$ as follows: for each $d \in D_o$, $V_I d = \mathbf{T}$ if $d \in S_I^U$, $\mathbf{F}$ otherwise. And V_I is obviously a semivaluation of D. And for all $d \in D_o$, $d \in S_I^U$ only if there is an S_I-derivation for d. This follows from Theorem 1.

Theorem 2. *Let $I \subseteq \Pi(D)$. $d \in S_I^U$ only if there is a finite $I' \subseteq I$ such that $d \in S_{I'}^U$.*
Proof Assume $d \in S_I^U$. Then by the fact that a derivation sequence is finite, it is clear that there is a finite $I' \subseteq I$ such that there is a finite elementary $S_{I'}$-derivation sequence. □

Definition Let $I \subseteq \Pi(D)$. Then I induces the set $\mathcal{D}^I$ of V_I-complexes based on D and that one-one onto function $\kappa_I : D \to \mathcal{D}^I$ given by the definition of V_I-complexes, and the following functions whose domain is D: the function $\boldsymbol{v}_I$ such that for each $d \in D$, $\boldsymbol{v}_I(d) = (\kappa_I d)^2$, and the function $\boldsymbol{e}_I$ such that for $d \in D$, $\boldsymbol{e}_I d = (\kappa_I d)^{\odot}$. Let $d \in D_{\alpha \to \beta}$. Then for all $d_1 \in D_\alpha$, $\boldsymbol{e}_I(d)d_1 = \boldsymbol{e}_I(dd_1)$. □

Lemma 3. *Let $I_1 \subseteq I_2 \subseteq \Pi(D)$. Then (a) $V_{I_1} \subseteq V_{I_2}$. (b) $\boldsymbol{v}_{I_1} \subseteq \boldsymbol{v}_{I_2}$. (c) $\boldsymbol{e}_{I_1} \subseteq \boldsymbol{e}_{I_2}$.* □

Definition Let $\langle D, J \rangle$ be a general pre-interpretation. An *interpretation* $\mathcal{M}$ is a pair $\langle D, I \rangle$ where I is a type consistent mapping in $\Pi \to \Phi(D)$. We call $\mathcal{M}$ a D-interpretation. An *assignment φ into* $\mathcal{M}$ is a type consistent mapping $\varphi : \Delta \to D$. When F is a formula in $\mathcal{L}$, we write $\mathcal{M} \models F[\varphi]$ to say that $\mathcal{M}$ satisfies F wrt φ. For all goal formulas G, G_1, G_2, for each rigid atom A, (a) When $s_\alpha, t_\alpha \in T(\Sigma)_\alpha$, $\mathcal{M} \models s_\alpha = t_\alpha[\varphi]$ iff $\mathsf{V}_\varphi s_\alpha = \mathsf{V}_\varphi t_\alpha$, $\mathcal{M} \models s_\alpha \equiv t_\alpha[\varphi]$ iff $\boldsymbol{e}_I(\mathsf{V}_\varphi s_\alpha) = \boldsymbol{e}_I(\mathsf{V}_\varphi t_\alpha)$. (b) $\mathcal{M} \models \top[\varphi]$. (c) $\mathcal{M} \models p(t_1, \cdots, t_n)[\varphi]$ iff $\langle \mathsf{V}_\varphi t_1, \cdots, \mathsf{V}_\varphi t_n \rangle \in Ip$ if p is a constant, or $\langle \mathsf{V}_\varphi t_1, \cdots, \mathsf{V}_\varphi t_n \rangle \in \varphi \circ \boldsymbol{e}_I(p)$ if p is a variable. (d) $\mathcal{M} \models G_1 \vee G_2[\varphi]$ iff $\mathcal{M} \models G_1[\varphi]$ or $\mathcal{M} \models G_2[\varphi]$. (e) $\mathcal{M} \models G_1 \wedge G_2[\varphi]$ iff $\mathcal{M} \models G_1[\varphi]$ and $\mathcal{M} \models G_2[\varphi]$. (f) $\mathcal{M} \models \exists x_\alpha G$ iff there is a $d \in D_\alpha$ such that $\mathcal{M} \models G[\varphi[d/x_\alpha]]$. (g) $\mathcal{M} \models A \leftarrow G[\varphi]$ iff $\mathcal{M} \models A[\varphi]$ if $\mathcal{M} \models G[\varphi]$.
We write $\mathcal{M} \models F$ to say that a formula F is *valid* in $\mathcal{M}$ if $\mathcal{M} \models F[\varphi]$ for all assignments φ into $\mathcal{M}$. Given a definite program $\mathcal{P}$, we say that $\mathcal{M}$ is a *model* or *D-model* for $\mathcal{P}$, and write $\mathcal{M} \models \mathcal{P}$, if each definite clause in $\mathcal{P}$ is valid in $\mathcal{M}$. Given a closed goal formula G, we say that G is a *logical consequence* of $\mathcal{P}$, and write $\mathcal{P} \models G$ if G is valid in all models of $\mathcal{P}$. □
Definition Let $\mathcal{F} = \langle D, J \rangle$ be a general pre-interpretation, V a semivaluation of D, and $\mathcal{D}$ be the set of V-complexes based on D. Given an $\mathcal{L}$-strucure $\mathcal{A} = \langle \mathcal{D}, J \rangle$ based on $\mathcal{F}$, the *D-interpretation $\mathcal{A}^{\odot}$ induced by $\mathcal{A}$* is defined to be $\langle D, I \rangle$ where $I = J \circ \kappa \circ (\cdot)^{\odot} \uparrow \Pi$. Conversely, given a D-interpretation $\mathcal{M} = \langle D, I \rangle$ based on $\mathcal{F}$, we can get the set $D^{\oplus}$ of V_I-complexes based on D. Then $\mathcal{M}^{\oplus}$ is an *$\mathcal{L}$-structure $\langle D^{\oplus}, J \rangle$ induced by $\mathcal{M}$*. □

Using the above facts and since assignments into D and $\mathcal{D}$ have one-one correspondence between them, we can show that the two semantics are elemetarily equivalent in the following sense.

Theorem 4. *(a) For all formula F in $\mathcal{L}$, $\models F$ iff $\models F$. (b) If $\mathcal{P}$ be a definite program and G a closed goal, then $\mathcal{P} \models G$ iff $\mathcal{P} \models G$.* □

Theorem 5. *The extensionality is not valid.*
Proof Take an extensionality formula $p_o \equiv q_o \to p_o = q_o$. It is obvious that $\mathcal{V}_\varphi^2 p_o = \mathcal{V}_\varphi^2 q_o$ does not imply that $\mathcal{V}_\varphi p_o = \mathcal{V}_\varphi q_o$. For the extensionality formula $(\forall x_\alpha \cdot fx \equiv gx) \to f = g$, we take $\alpha \in \mathcal{T}_0$ and $\beta = o$ and D-interpretation I such that $If = Ig = \mathbf{T}_\alpha^D$. Then $f \equiv g$ but not always $f = g$. □

Let $\mathcal{M} = \langle D, I \rangle$ be an interpretation based on $\mathcal{F} = \langle D, J \rangle$, we can identify $\mathcal{M}$ with the subset I of $\Pi(D)$. And every subset I of $\Pi(D)$ is a D-interpretation. Obviously the set of all D-interpretation is a complete lattice with the usual set inclusion ordering between D-interpretations.

Theorem 6. *Let $I_1 \subseteq I_2 \subseteq \Pi(D)$. If $I_1 \models G[\varphi]$, then $I_2 \models G[\varphi]$.*
Proof By induction on G. When G is $\top$, it is obvious. When G is a rigid atom $p(t_1, \cdots, t_n)$, since $I_1 p \subseteq I_2 p$, $I_2 \models G[\varphi]$. When G is $p(t_1, \cdots, t_n)$ where p is a variable. Since $e_{I_1} \subseteq e_{I_2}$, $I_2 \models p(t_1, \cdots, t_n)[\varphi]$. When G is $G_1 \wedge G_2$. $I_1 \models G_1[\varphi]$ and $I_1 \models G_2[\varphi]$. By IH $I_2 \models G_1[\varphi]$ and $I_2 \models G_2[\varphi]$. So $I_2 \models G[\varphi]$. When G is $G_1 \vee G_2$. Assume, wlog, $I_1 \models G_1[\varphi]$. By IH $I_2 \models G_1[\varphi]$. When G is $\exists x_\alpha G_1$. There exists a $d \in D_\alpha$ such that $I_1 \models G_1[\varphi[d/x_\alpha]]$. By IH $I_2 \models G_1[\varphi[d/x_\alpha]]$. So $I_2 \models G[\varphi]$. □

Let $\mathcal{F} = \langle D, J \rangle$ be a general pre-interpretation. We can define a mapping $\mathsf{T}_\mathcal{P}^D$ from the lattice of D-interpretations to itself. Let $\mathcal{F}$ be a pre-interpretation $\langle D, J \rangle$ of a definite program $\mathcal{P}$ and I a D-interpretation. Then $\mathsf{T}_\mathcal{P}^D(I) = \{p(d_1, \cdots, d_n) \in \Pi(D) :$ there exist an assignment φ into D and a clause $p(t_1, \cdots, t_n) \leftarrow G \in \mathcal{P}$ such that $d_i = \mathsf{V}_\varphi t_i$ for each $i \in [n]$ and $I \models G[\varphi]\}$

Lemma 7. *$\mathsf{T}_\mathcal{P}^D$ is monotonic, i.e. given $I_1 \subseteq I_2 \subseteq \Pi(D)$, $\mathsf{T}_\mathcal{P}^D(I_1) \subseteq \mathsf{T}_\mathcal{P}^D(I_2)$.*
Proof Assume $p(d_1, \cdots, d_n) \in \mathsf{T}_\mathcal{P}^\mathcal{F}(I_1)$ for $p(d_1, \cdots, d_n) \in \Pi(D)$. Then there are an assignment φ into I_1 and a clause $p(t_1, \cdots, t_n) \leftarrow G \in \mathcal{P}$ such that $\mathsf{V}_\varphi t_i = d_i$ for all $i \in [n]$ and $I_1 \models G[\varphi]$. By Theorem 6, $I_2 \models G[\varphi]$. □

So $\mathsf{T}_\mathcal{P}^\mathcal{F}$ is a monotonic transformation on the set of all D-interpretations.

Lemma 8. *Let $I \subseteq \Pi(D)$. Then $I \models \mathcal{P}$ iff $\mathsf{T}_\mathcal{P}^D(I) \subseteq I$.*
Proof $\Rightarrow$) Assume $p(d_1, \cdots, d_n) \in \mathsf{T}_\mathcal{P}(I)$ for some $p(d_1, \cdots, d_n) \in \Pi(D)$. Then there are an assignment φ into D and a clause $p(t_1, \cdots, t_n) \leftarrow G \in \mathcal{P}$ such that $\mathsf{V}_\varphi t_i = d_i$ for all $i \in [n]$ and $I \models G[\varphi]$. Then since $I \models \mathcal{P}$, $I \models p(t_1, \cdots, t_n)[\varphi]$. Therefore $p(d_1, \cdots, d_n) \in I$.
$\Leftarrow$) Similarly. □

Lemma 9. *Let I_1 and I_2 be D-models of $\mathcal{P}$. Then $I_1 \cap I_2$ is also D-model of $\mathcal{P}$.*
Proof Since $\mathsf{T}_\mathcal{P}(I_1) \subseteq I_1$ and $\mathsf{T}_\mathcal{P}(I_2) \subseteq I_2$, by monotonicity of $\mathsf{T}_\mathcal{P}$ operator, $\mathsf{T}_\mathcal{P}(I_1 \cap I_2) \subseteq \mathsf{T}_\mathcal{P}(I_1) \subseteq I_1$ and $\mathsf{T}_\mathcal{P}(I_1 \cap I_2) \subseteq \mathsf{T}_\mathcal{P}(I_2) \subseteq I_2$. So $\mathsf{T}_\mathcal{P}(I_1 \cap I_2) \subseteq I_1 \cap I_2$. □

But the set of all D-models is not closed under join operation, i.e. $I_1 \cup I_2$ is not necessarily a D-model, whenever I_1 and I_2 are D-models. Take for example the definite program $\mathcal{P}_2 = \{p \leftarrow q, r\}$. Then $\Pi(D) = \{p, q, r\}$. $\{q\}$ and $\{r\}$ are D-models for $\mathcal{P}_2$, but $\{q, r\}$ is not a D-model.

Lemma 10. *Let $\langle I_n \rangle_{n \in \omega}$ be ω-chain of D-interpretations. Then for each goal G and assignment φ into D, $\sqcup_{n \in \omega} I_n \models G[\varphi]$ only if there is an $n \in \omega$ such that $I_n \models G[\varphi]$.*
Proof Let $I = \sqcup_{n \in \omega} I_n$. Then $I \models G[\varphi]$ only if $V_I(\mathsf{V}_\varphi G) = \mathbf{T}$. So there is a finite $I' \subseteq I$such that $V_{I'}(\mathsf{V}_\varphi G) = \mathbf{T}$. Therefore there is an $n \in \omega$ such that $I' \subseteq I_n$. By monotonicity $I_n \models G[\varphi]$. □

Lemma 11. *$\mathsf{T}_\mathcal{P}^D$ is continuous.*
Proof Let $\langle I_n \rangle_{n \in \omega}$ be a ω-chain of D-interpretations. We need to show: $\mathsf{T}_\mathcal{P}(\sqcup_{n \in \omega} I_n) = \sqcup_{n \in \omega} \mathsf{T}_\mathcal{P}(I_n)$. The monotonicity of $\mathsf{T}_\mathcal{P}$ implies that $\sqcup_{n \in \omega} \mathsf{T}_\mathcal{P}(I_n) \subseteq \mathsf{T}_\mathcal{P}(\sqcup_{n \in \omega} I_n)$. Now we need to show that $\mathsf{T}_\mathcal{P}(\sqcup_{n \in \omega} I_n) \subseteq \sqcup_{n \in \omega} \mathsf{T}_\mathcal{P}(I_n)$. Let $d_1, \cdots, d_n \in D$, and $p(d_1, \cdots, d_n) \in \Pi(D)$. Assume $p(d_1, \cdots, d_n) \in \mathsf{T}_\mathcal{P}(\sqcup_{n \in \omega} I_n)$, to show $p(d_1, \cdots, d_n) \in \sqcup_{n \in \omega} \mathsf{T}_\mathcal{P}(I_n)$. There are $p(t_1, \cdots, t_n) \leftarrow G \in \mathcal{P}$ and an assignment φ into H such that $\varphi t_i = d_i$ for all $i \in [n]$ and $\sqcup_{n \in \omega} I_n \models G[\varphi]$. So there is $n \in \omega$ such that $I_n \models G[\varphi]$. Therefore there is $n \in \omega$ such that $p(d_1, \cdots, d_n) \in \mathsf{T}_\mathcal{P}(I_n)$. □

So we can show that every definite program has the least D-model as follows:

Theorem 12. *$(\mathsf{T}_\mathcal{P}^D)^\omega(\phi)$ is the least fixpoint of $\mathsf{T}_\mathcal{P}^D$.* □

Theorem 13. *Let $\mathsf{M}_\mathcal{P}^D = \bigcap\{I \subseteq \Pi(D) : I \models P\}$, then $\mathsf{M}_\mathcal{P}^D$ is the least D-model of $\mathcal{P}$ and $\mathsf{M}_\mathcal{P}^D = \mathsf{T}_\mathcal{P}^\omega(\phi)$.*
Proof By Lemmas 8,9,7 and Theorem 12. □

4 Herbrand Models

In order to determine validity or logical consequences, we need to consider all interpretations of the language $\mathcal{L}$. In this section we shall show that we can restrict our attention to Herbrand models. That is, we show that if A is true in all Herbrand (that is symbolic) models it follows that A is true in all models and a fortiori in the model intended by the person who wrote the program.

Definition The *Herbrand frame* H is a set such that (a) H is the set of all ρ-normal c-terms. (b) Let $f \in H_{\alpha\to\beta}$, then for all $t \in H_\alpha$, $f(t) = \rho(ft)$. □

It is obvious that the Herbrand frame H is countable.

Definition The *Herbrand pre-interpretation* $\mathcal{HF}$ is a pre-interpretation $\langle H, J\rangle$ such that H is the Herbrand frame and J satisfies the following: (a) If c_α is a constant such that α is a primitive type, then $Jc_\alpha = c_\alpha$. (b) If $d_{\alpha\to\beta}$ is a constant of type $\alpha \to \beta$, then for all $t_\alpha \in H_\alpha$, $(Jd_{\alpha\to\beta})(t_\alpha) = d_{\alpha\to\beta}t_\alpha$. □

Lemma 14. *The Herbrand pre-interpretation is general.* □

Definition An *Herbrand interpretation* $\mathcal{M}$ is an interpretation $\langle H, I\rangle$ based on the Herbrand pre-interpretation. The *Herbrand base* $\mathcal{HB}$ is the set $\Pi(H)$. □

Let $I \subseteq \Pi(H)$ be an Herbrand interpretation and φ an assignment into I. Then we can consider φ as the generalized substitution σ such that for each term $t \in T(\Sigma)$, $\sigma t = (\varphi \upharpoonright FV(t))t$. It is easy to see that for every term t, φt is a c-term and $\mathsf{V}_\varphi t = \varphi t$, for each goal formula G, φG a closed goal formula, and for each definite clause C, φC a closed definite clause.

Let I be a D-interpretation based on $\mathcal{F}$. The *Herbrand interpretation* I^* *induced by* I is an Herbrand interpretation such that for every $A \in \Pi(H)$, $A \in I^*$ iff $I \models A$. Let φ and φ' be assignments into H and D, respectively. Then we say that φ' is *induced by* φ if $\varphi' = \varphi \circ \mathsf{V}^{\mathcal{F}}$. The mapping $\mathsf{V}^{\mathcal{F}} : H \to D$ is a *homomorhism from* I^* *into* I, since for $p \in \Pi_{\alpha_1,\cdots,\alpha_n\to o}$, $h_i \in H_{\alpha_i}, i \in [n]$, if $\langle h_1, \cdots, h_n\rangle \in I^*p$, then $\langle \mathsf{V}^{\mathcal{F}}h_1, \cdots, \mathsf{V}^{\mathcal{F}}h_n\rangle \in Ip$. Let $h \in H_{\alpha_1,\cdots,\alpha_n\to o}$. Then for all $h_i \in H_{\alpha_i}, i \in [n]$, $\langle h_1, \cdots, h_n\rangle \in e_{I^*}(h)$ implies $\langle \mathsf{V}^{\mathcal{F}}h_1, \cdots, \mathsf{V}^{\mathcal{F}}h_n\rangle \in e_I(\mathsf{V}^F h)$.

Lemma 15. *Let* $I, I^*, \varphi', \varphi$ *be as above. Then (a) If* t *is a term, then* $\mathsf{V}^{\mathcal{F}}_{\varphi'}(\varphi t) = \mathsf{V}^{\mathcal{F}}_{\varphi'}t$, *(b) If* A *is a rigid atom then* $I^* \models A[\varphi]$ *iff* $I \models A[\varphi']$, *(c) If* G *is a goal formula such that* $I^* \models G[\varphi]$, *then* $I \models G[\varphi']$, *(d) If* C *is a definite clause such that* $I \models C[\varphi']$, *then* $I^* \models C[\varphi]$, *(e) Then if* $I \models \mathcal{P}$, *then* $I^* \models \mathcal{P}$. □

Let $\mathcal{F}$ be a general pre-interpretation. Then $\models_{\mathcal{F}}$ denotes logical implication in the context of fixed domains and functional assignment. Specifically $\models_{\mathcal{HF}}$ denotes logical implication in the context of Herbrand frame and functional assignment.

Let G be a goal formula. We write $\exists(G)$ to denote the existential closure of free variables in G.

Theorem 16. *Let* $\mathcal{P}$ *be a definite program and* G *a goal formula. Then* $\mathcal{P} \models \exists(G)$ *iff* $\mathcal{P} \models_{\mathcal{HF}} \exists(G)$.
Proof $\Longleftarrow$) Let an Herbrand interpretation induced by the given interpretation I be I^*. Assume $I \models \mathcal{P}$. Then $I^* \models \mathcal{P}$, so $I^* \models \exists(G)$. Then there is an assignment φ into I^* such that $I^* \models G[\varphi]$. Let the assignment φ' into I be induced by φ. Then $I \models G[\varphi']$ by Lemma 15 (c). So $I \models \exists(G)$. □

If φ is a substitution, then φ_{-x_α} is that substitution σ such that $\sigma = \varphi \upharpoonright (\Delta - \{x_\alpha\})$.

Lemma 17. *Let* $I \subseteq \Pi(H)$. *Then for all closed substitution* σ, *assignment* φ *into* H, *and goal formula* G, $I \models \sigma G[\varphi]$ *iff* $I \models \varphi\sigma G$.
Proof We prove by induction on G. When G is $\top$ or a rigid atom, it is obvious. When G is

$p(t_1, \cdots, t_n)$ where $p \in \Delta$. $I \models \sigma G[\varphi]$ iff $\langle \varphi\sigma t_1, \cdots, \varphi\sigma t_n \rangle \in e_I(\varphi\sigma p)$ iff $\langle \varphi'\varphi\sigma t_1, \cdots, \varphi'\varphi\sigma t_n \rangle \in e_I(\varphi'[\varphi\sigma p/p]p)$ for all assignment φ' into H iff $I \models \varphi\sigma G[\varphi']$ for all assignment φ' into H iff $I \models \varphi\sigma G$.

When G is $\exists x_\alpha G_1$. $I \models \sigma G[\varphi]$ iff $I \models \exists x_\alpha \sigma_{-x_\alpha} G_1[\varphi]$ iff there is an $h \in H_\alpha$ such that $I \models \sigma_{-x_\alpha} G_1[\varphi[h/x_\alpha]]$ iff there is an $h \in H_\alpha$ such that $I \models \varphi[h/x_\alpha]\sigma_{-x_\alpha} G_1$ by IH iff for all assignment φ' into H, $I \models \varphi'\varphi[h/x_\alpha]\sigma_{-x_\alpha} G_1$, since $\varphi[h/x_\alpha]\sigma_{-x_\alpha} G_1$ is a closed goal. iff $I \models (\varphi'[h/x_\alpha])\varphi_{-x_\alpha}\sigma_{-x_\alpha} G_1$ iff $I \models \varphi_{-x_\alpha}\sigma_{-x_\alpha} G_1[\varphi'[h/x_\alpha]]$ iff $I \models \exists x_\alpha \varphi_{-x_\alpha}\sigma_{-x_\alpha} G_1[\varphi']$ iff $I \models \varphi\sigma G$. □

Corollary 18. *For all assignment φ into H, goal formula G, $I \models G[\varphi]$ iff $I \models \varphi G$.* □

Theorem 19. *For all closed substitution σ and goal formula G such that $\sigma\exists x_\alpha G$ is closed, $I \models \sigma\exists x_\alpha G$ iff there is an $h \in H_\alpha$ such that $I \models \sigma[h/x_\alpha]G$.*
Proof Let φ be an assignment into H. $I \models \sigma\exists x_\alpha G[\varphi]$ iff $I \models \exists x_\alpha \sigma_{-x_\alpha} G[\varphi]$ iff there is an $h \in H_\alpha$ such that $I \models \sigma_{-x_\alpha} G[\varphi[h/x_\alpha]]$ iff there is an $h \in H_\alpha$ such that $I \models \varphi[h/x_\alpha]\sigma_{-x_\alpha} G$ by Corollary 18 iff $I \models \sigma[h/x_\alpha]G[\varphi]$ by Corollary 18, since $\varphi[h/x_\alpha]\sigma_{-x_\alpha} = \varphi\sigma[h/x_\alpha]$. □

Corollary 20. *Let $\mathsf{M}_\mathcal{P}^H = \bigcap\{I \subseteq \Pi(H) : I \models \mathcal{P}\}$. Then $\mathsf{M}_\mathcal{P}^H \models \mathcal{P}$.*
Proof Follows from Theorem 13. □

Theorem 21. *$(\mathsf{T}_\mathcal{P}^H)^\omega(\phi)$ is the least fixed point of $\mathsf{T}_\mathcal{P}^H$ and $\mathsf{M}_\mathcal{P}^H = (\mathsf{T}_\mathcal{P}^H)^\omega(\phi)$.*
Proof Follows from Lemma 11. □

Theorem 22. *Let $A \in \Pi(H)$. Then $\mathcal{P} \models A$ iff $\mathsf{M}_\mathcal{P}^H \models A$.*
Proof $\mathcal{P} \models A$ iff $\mathcal{P} \models_{\mathcal{HF}} A$ iff for all H-interpretation I such that $I \models \mathcal{P}$, $A \in I$ iff $A \in \mathsf{M}_\mathcal{P}^H$. □

For the definite program $\mathcal{P}_1$ introduced in section 1, it is easy to see that

$$\mathsf{T}_{\mathcal{P}_1}^\omega(\phi) = \{p(a), q(a), r(p(a))\}$$

So $r(p(a))$ is a logical consequence of $\mathcal{P}_1$, while $r(q(a))$ is not.

The program $\mathcal{P}_1$ is non-extensional in the sense that extensional identity of arguments of the predicate r does not imply extensional identity of proposition $r(\cdot)$. In [18] Wadge defined a fragment of higher-order logic programming language (in fact it's a pure subset of HiLog [8]) where every program behaves extensionally.

Example We can define the following higher-order logic program $\mathcal{P}_3$ in the language of [18]: Let MAP be predicate constant of type $(int \to o), list \to o$ and $\cdot$ be an infix functional constant of type $int, list \to list$ and p and q predicate constants of type $int \to o$ and $\mathcal{P}_3$ include the following definite clauses.
$MAP(z, x \cdot l) \leftarrow zx \wedge MAP(z, l)$.
$MAP(z, nil) \leftarrow \top$.

Assume that the above clauses are the only clauses that defines the predicate MAP. Let I be a fixpoint of $\mathsf{T}_{\mathcal{P}_3}$. We shall show that $p \equiv q \to MAPp \equiv MAPq$ is valid under I. Let $p \equiv q$ valid under I. Then for all $a \in H_{int}$, $pa \in I$ iff $qa \in I$. Moreover the set H_{list} has the following inductive characteriztion. (a) $nil \in H_{list}$. (b) For $a \in H_{int}$, $a \cdot l \in H_{list}$ if $l \in H_{list}$. To prove $MAPp \equiv MAPq$ is valid in I, it's enough to show that for all $l \in H_{list}$, $MAP(p, l) \in I$ iff $MAP(q, l) \in I$. We prove this by induction on l. Obviously $MAP(p, nil), MAP(q, nil) \in I$. Let $a \cdot l \in H_{list}$. Assume $MAP(p, a \cdot l) \in I$ to show $MAP(q, a \cdot l) \in I$. Then $pa, MAP(p, l) \in I$. So by IH, $MAP(q, l) \in I$. Therefore $MAP(q, a \cdot l) \in I$. □

5 Completeness

In this section we prove completeness of interpreter in [14]. Our actual interpreter is that of [14] plus backchaining when atomic goals need to be solved. The definition of this non-deterministic interpreter can be given by describing how a theorem prover for programs and goals should function. This interpreter, given the pair $\langle \mathcal{P}, G\rangle$ in its initial state, should either succeed or fail. We shall use the notation $\mathcal{P} \vdash G$ to indicate the meta proposition that the interpreter succeeds if started in the state $\langle \mathcal{P}, G\rangle$. The search related semantics which we want to attribute to the logical constants can be specified as follows: (a) $\mathcal{P} \vdash \top$. (b) $\mathcal{P} \vdash G_1 \vee G_2$ only if $\mathcal{P} \vdash G_1$ or $\mathcal{P} \vdash G_2$. (c) $\mathcal{P} \vdash G_1 \wedge G_2$ only if $\mathcal{P} \vdash G_1$ and $\mathcal{P} \vdash G_2$. (d) $\mathcal{P} \vdash \exists x_\alpha G_1$ only if there is some term $t \in T(\Sigma)_\alpha$ such that $\mathcal{P} \vdash [t/x_\alpha]G_1$. (e) $\mathcal{P} \vdash A$ only if there are a definite clause $A_1 \leftarrow G_1 \in \mathcal{P}$ and a substitution σ such that $A = \sigma A_1$ and $\mathcal{P} \vdash \sigma G_1$.

Let F be a formula of $\mathcal{L}$. Then $|F|$ denotes the set $\{\varphi F : \varphi \text{ is an assignment into } H\}$. It is easy to see that if F is a goal formula, $|F|$ is a set of closed goal formulas, and if F is a definite clause, then $|F|$ is a set of closed definite clauses. This notation can be extended to set Γ of formulas of $\mathcal{L}$: $|\Gamma| = \bigcup\{|F| : F \in \Gamma\}$.

Definition Let Γ be a set of formulas that are either closed atoms or definite clauses, and let G be a closed goal formula. Then a *Γ-derivation sequence for G* is a finite sequence $G^1, G^2, \cdots, G^n$ of closed goal formulas such that G^n is G, and for each $i \in [n]$, (a) if G^i is a closed atom, then i) G^i is $\top$, or ii) $G^i \in \Gamma$, or iii) there is a definite clause $G^i \leftarrow G^j \in |\Gamma|$ such that $j < i$, (b) if G^i is $G_1 \vee G_2$, then for some $j < i$, G^j is either G_1 or G_2, (c) if G^i is $G_1 \wedge G_2$, then for some $j, k < i$, $G^j = G_1$ and $G^k = G_2$, (d) if G^i is $\exists x_\alpha G_1$, then there is a $t \in H_\alpha$ and $j < i$ such that $[t/x_\alpha]G_1 = G^j$. □

Theorem 23. *Let $I \subseteq \Pi(H)$. Then for all closed goal formula G, $I \models G$ iff there is an I-derivation sequence for G.*

Proof $\Leftarrow$) Let $G^1, \cdots, G^n$ be an I-derivation sequence. We prove by induction on i: for all $i \in [n]$, $I \models G^i$. When $i = 1$, then it is obvious. When $i > 1$. If $G^i = G_1 \wedge G_2$, then by IH, $I \models G_1$ and $I \models G_2$. So $I \models G_1 \wedge G_2$. If $G^i = \exists x_\alpha G_1$, then by IH, there is a $t \in H_\alpha$ such that $I \models [t/x_\alpha]G_1$. So $I \models \exists x_\alpha G_1$ by Theorem 19.

$\Rightarrow$) Follows from Theorem 1, since for a Herbrand interpretation I, we can identify I with S_I. □

Lemma 24. *Let G be a closed goal formula. Then $\mathcal{P} \vdash G$ iff there is a $\mathcal{P}$-derivation for G.*

Proof See [16]. □

Theorem 25. *Let G be a closed goal formula. Then $\mathcal{P} \vdash G$ iff $\mathcal{P} \models G$.*

Proof By Theorems 22,21, $\mathcal{P} \models G$ iff $T^\omega_{\mathcal{P}}(\phi) \models G$. Let $I_n = T^n_{\mathcal{P}}(\phi)$ for $n \in \omega$. Now we need to prove that there is a $\mathcal{P}$-derivation $G^1, \cdots, G^l$ for G iff there is an $n \in \omega$ such that $I_n \models G$.

$\Rightarrow$) By induction on l. When G is $\top$, $I_0 \models \top$. When G is $G_1 \wedge G_2$, then there are $\mathcal{P}$-derivations for G_1 and G_2 whose lengths are less than l. So by IH, there are $n_1, n_2 \in \omega$ such that $I_{n_1} \models G_1$ and $I_{n_2} \models G_2$. Assume, wlog, $n_1 < n_2$. Then $I_{n_2} \models G_2$, so $I_{n_2} \models G_1 \wedge G_2$. When G is $\exists x_\alpha G_1$. Then there are a term $t \in H_\alpha$ and a $\mathcal{P}$-derivation for $[t/x_\alpha]G_1$ whose length is less than l. So by IH, there is an $n \in \omega$ such that $I_n \models [t/x_\alpha]G_1$. Therefore $I_n \models \exists x_\alpha G_1$ by Theorem 19. When G is a rigid atom A. Then there are a number $j < l$ and a definite clause $A \leftarrow G^j \in |\mathcal{P}|$. By IH, $I_n \models G^j$. Therefore $I_{n+1} \models A$.

$\Leftarrow$) We prove the claim by induction on n. First assume the claim true if $I_n \models G$. To prove the claim for $n+1$ assume $I_{n+1} \models G$. Then there is an I_{n+1}-derivation $G^1, \cdots, G^m$ for G by Theorem 23. Now we prove, by induction on i, that there is a $\mathcal{P}$-derivation for G^i, for each $i \in [m]$. If G^i is $\top$, it is immediate. If G^i is a rigid atom A, then since $A \in I_{n+1}$, there is a definite clause $A \leftarrow G_1 \in |\mathcal{P}|$ such that $I_n \models G_1$. Then by our first assumption, there is a $\mathcal{P}$-derivation for G_1. We now get a $\mathcal{P}$

derivation for A by appending A to this sequence. When G^i is $G_1 \wedge G_2$. Then by our second IH, there are $\mathcal{P}$-derivations for G_1 and G_2. Now we get a $\mathcal{P}$-derivation for G^i by appending G^i to the end of concatenation these sequences. When G^i is $\exists x_\alpha G_1$. By second IH, there is a term $t \in H_\alpha$ such that there is a $\mathcal{P}$-derivation for $[t/x_\alpha]G_1$, to which we attach G^i to get $\mathcal{P}$-derivation for G^i. □

6 Conclusion

We have built a general model theoretic semantics for Higher-Order Horn logic programming language and established the least model and least fixed point semantics. We also showed soundness and completeness of those interpreters developed in [16, 14] by establishing equivalence between the fixed point semantics and the operational semantics of those interpreters based on $\mathcal{P}$-derivations.

7 Acknowledgements

We wish to thank Prof. Sanchis for pointing out a serious error in the previous version of this paper. The first author also would like to thank Prof. Dale Miller for the encouragements and helpful suggestions.

References

1. James H. Andrews. Predicates as parameters in logic programming: A set-theoretic basis. In P. Schroeder-Heister, editor, *Extensions of Logic Programming*, pages 31–47, 1989.
2. Peter B. Andrews. Resolution in type theory. *The Journal of Symbolic Logic*, 36(3):414–432, 1971.
3. Peter B. Andrews. General models and extensionality. *The Journal of Symbolic Logic*, 37(2):395–397, 1972.
4. Peter B. Andrews. General models, descriptions, and choice in type theory. *The Journal of Symbolic Logic*, 37(2):385–394, 1972.
5. Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth through Proof.* Academic Press, 1986.
6. Mino Bai. A declarative foundation of λProlog with equality. Technical Report SU-CIS-92-03, Syracuse University, 1992.
7. H. P. Barendregt. *The Lambda Calculus.* North-Holland, 1984.
8. Weidong Chen, Kichael Kifer, and David S. Warren. Hilog: A first-order semantics for higher-order logic programming constructs. In Ewing L. Lusk and Ross A. Overbeek, editors, *Logic Programming Proceedings of North American Conference*, pages 1090–1114, 1989.
9. Alonzo Church. A formulation of simple theory of types. *The Journal of Symbolic Logic*, 5:56–68, 1940.
10. Leon Henkin. Completeness of the theory of types. *The Journal of Symbolic Logic*, 15:81–91, 1950.
11. John W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1987.
12. Dale A. Miller. *Proofs in higher-order logic.* PhD thesis, Carnegie-Mellon University, 1983.
13. Dale A. Miller. A compact representation of proofs. *Studia Logica*, 46(4):347–370, 1987.
14. Dale A. Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. Technical report, University of Pennsylvania, 1989.
15. Dale A. Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.
16. Gopalan Nadathur. *A higher-order logic a the basis for logic programming.* PhD thesis, University of Pennsylvania, 1986.
17. J. A. Robinson. Mechanizing higher-order logic. *Machine Intelligence*, 4:150–170, 1969.
18. William W. Wadge. Higher-order horn logic programming. In U. Saraswat and K. Ueda, editors, *Proceedings of International Logic Programming Symposium*, pages 289–303, 1991.