Uniform Algebras:

A Complete semantics for

Higher Order Logic programming with HOHH Formulae.

Mary DeMarco
Wesleyan University
mdemarco@wesleyan.edu

James Lipton
Wesleyan University
jlipton@wesleyan.edu

January 24, 2002

Abstract

This paper introduces a model theory for resolution on Higher Order Hereditarily Harrop formulae (HOHH), the logic underlying the lambda-Prolog programming language. Based on an intuitionistic sequent presentation of Church's theory of types we call ICTT, the HOHH formalism was introduced by Miller and Nadathur in the late 1980's and shown to be an instance of their notion of an *abstract logic programming language* in [17].

The classical Higher-order Horn fragment was shown sound and complete with respect to a semantics introduced by Wolfram in [22], and independently by Blair and Bai in [3], but none has ever been supplied for the full HOHH logic, with intuitionistic implication. HOHH with uniform proofs (a sequent calculus extension of resolution) is here shown sound and and complete with respect to *Uniform Algebras*, an indexed version of typed Omega-set semantics similar to that introduced by Fourman and Scott in [10]. We develop a bottom-up semantics based on these structures producing a least term model as the least fixed point of an appropriate continuous operator.

1 Introduction

The Lambda Prolog language was first developed in the late 1980's. It is based on work of Miller, Nadathur, Pfenning, Scedrov, Felty and others aimed at adding to the expressive power of Horn Clause logic programming without sacrificing its declarative transparency by extending the original first-order core to include new connectives and proof rules from well-behaved fragments of higher-order *intuitionistic* type theory.

Since the classical and intuitionistic theories of both the first and higher-order Horn fragment coincide, the difference in focus between classical and intuitionistic logic has no practical consequences for these logics. The distinction, however, becomes essential when logical extensions that permit implication in goals and clauses are considered.

Intuitionistic formalisms are certainly a plausible candidate for logic programming extensions for a number if reasons. A logical formalism can hardly be considered declarative without the *existence* (EP) and *disjunction* (DP) properties: existential goals should not be logical consequences of a program unless some instance is, and backtracking through disjunctive alternatives should not sacrifice completeness. Both properties *fail*, in general, in classical extensions of Horn Logic, but hold in many intuitionistic extensions.

As it turns out, global satisfaction of EP and DP is not sufficient to guarantee completeness of a suitably operational backwards proof search: each intermediate state of proof search must also satisfy them as well, hence the *hereditary* or local Harrop requirement of lambda-Prolog programs.

Such operational considerations led Miller et. al. to formulate the general proof-theoretic principle of *uniformity* in [17], where a cut-free sequent calculus fragment is termed an *abstract logic programming language* if restriction to a certain class of immediately decomposable deductions, called *uniform* proofs, does not sacrifice theorems. A class of sequent proofs is uniform if every occurrence, in a deduction, of a sequent

$$\mathsf{Program} \vdash \mathsf{Goal}$$

with a nonatomic goal may be assumed, without loss of generality, to result from an introduction of the principal goal connective. This guarantees that a backchaining proof search may be restricted to destruction of connectives in the goals until they are atomic without sacrificing completeness. A uniform system is said to admit a *search-theoretic interpretation* of connectives.[1]

In HOHH the following new resolution steps arise in addition to back-chaining.

AUGMENT To infer $\mathcal{P} \vdash D \supset G$ seek a proof of $\mathcal{P} \cup \{D\} \vdash G$

GENERIC To infer $\mathcal{P} \vdash \forall x.G$ seek a proof of $\mathcal{P} \vdash G[c/x]$ for a fresh constant c,

where the logic is simply typed, contains λ-abstraction and $\beta\eta$ reduction as well as suitably restricted higher-order formulae. Thus, implications are now permitted in goals (and in the tails of programs) in such a way as to preserve the $\mathsf{Program} \vdash \mathsf{Goal}$ syntax after application of the AUGMENT step, and so is explicit universal quantification.

Computationally, the gains are considerable. The AUGMENT rule permits dynamic loading of modules and a local *assert*, declaratively. The inclusion of higher-order logic permits declarative definition of higher-order predicates, and greatly facilitates, in conjunction with abstraction and reduction, declarative metaprogramming. These and many other gains are discussed extensively in [14].

Semantics: Declarative and Operational Over the past decade a series of papers have provided semantics for certain classical or intuitionistic fragments of HOHH (see [22, 3]) based on work by Henkin [13] and Andrews [1] on the semantics of Church's classical Type theory [6]. Nadathur's dissertation [20] provides a notion of term model for the HOHH fragment. Miller defines a Kripke-like bottom-up semantics for a first-order fragment [15] in which the syntax of programs is built into the notion of model.

In "A semantics for λProlog", D.A. Wolfram [22] provided semantics for the classical higher-order Horn clause fragment of λProlog, a result which was independently established in Bai and Blair [3] and was built on work in Nadathur's dissertation [20] and Andrews' V-complexes [2]. However, the larger hereditarily Harrop fragment considered here requires intuitionistic logic and, consequently, an intuitionistic model theory. Until recently (see [8, 9]) no such semantics had been developed even for the intuitionistic Church's type theory of which HOHH is an executable fragment.

Operationality: bottom-up semantics In [15], Miller produced an indexed variant of Kripke style models for a first-order fragment of lambda-Prolog in which the conventional definition of truth in Kripke semantics at a possible world p in an arbitrary poset:

$$p \Vdash A \supset B \qquad \textit{iff} \quad \textit{for every} \quad q \geq p, \qquad q \Vdash A \Rightarrow q \Vdash B \tag{1}$$

[1] The notion of an abstract logic programming language developed by Miller et. al. ended up having a broader scope than we are considering here: uniform proof calculi may exist for carefully chosen presentations of linear and certain classical logics as well, see e.g. FORUM [5, 16].

is replaced by:

$$P \Vdash D \supset G \qquad \textit{iff} \qquad P, D \Vdash G \tag{2}$$

where P and D must be program formulae and G a goal. Since the set of possible worlds is fixed in advance, and consists of formulae, this semantics –as far as programs are concerned– contains only term models. This paper addresses this problem by supplying arbitrary carriers for all types, including that of program formulae. We thus generalize Miller's semantics in [15], and extend it to higher order logic with universal quantification in goals as well.

The reader should note that definition (2) essentially mimics resolution, for which reason we have termed it operational semantics. It is easily shown sound and complete for resolution, and admits a minimal Herbrand-style model, definable as a suitable fixed point.

What is harder is to show that it agrees with full semantic logical consequence. In other words, that an equivalent model of type (1) above exists. The formal difference above between (1) and (2) reflects, in the semantics, the difference between full provability and uniform provability. In our treatment below we also define two notions of model. The former actually defines what is meant by truth in ICTT, the latter provides an operational semantics, in the same sense, for resolution, and also admits a least Herbrand-style model. Since our aim here is to model logic programming, the general semantics for ICTT is briefly sketched. Details can be found in [8, 9].

1.1 The main obstacles

If it is to capture the declarative content of programs, a model theory for HOHH must interpret an intuitionistic fragment of Church's type theory. In particular, it must be a model for the simply typed lambda-calculus, sound for $\beta\eta$ (and hence extensional on functionals). It must also supply a carrier for predicates and interpret the logical constants. Since quantifiers and connectives are higher-order constants in the language, the interpretation of terms cannot be easily divorced from the meaning of propositions. This is one of the reasons we construct a model in which *the object of truth values, a (parametrically) complete Heyting Algebra, is distinct from the carriers of the types of propositions*.

There are, in addition, a series of requirements peculiar to logic programming that must be met to supply a semantics with respect to which resolution deductions on HOHH are sound and complete. We summarize them here.

Breaking truth-functionality of higher-order predicates: In lambda-prolog we observe the following behavior.

Program:

```
module ntf.
type p o.                          p :- q.
type q o.                          q :- p.
type f o -> o.                    f(p).
```

Queries:

```
?- f(p).
solved

?- f(q).
no

?- f(p & p).
no
```

The logical equivalence of p and q, or of p and $p \wedge p$ plays no role in the search for a proof of $f(q)$ or of $f(p \wedge p)$. In other words, the following inference fails in the underlying logic of lambda-Prolog, and in ICTT, when A, B are propositions and the type of p is $o \longrightarrow o$:

$$\frac{A \iff B}{p(A) \longrightarrow p(B)}$$

Thus, If HOHH-resolution is to be sound and complete with respect to our model theory, the meaning of the formula $f(p \wedge p)$ must not be identified with $f(p)$. Once again, this is achieved by separating the cHa object of truth values Ω from the carrier D_o of propositions. Although p and $p \wedge p$ will have the same truth value in Ω, they will not be identified in D_o, making it possible to consistently assign different truth values to $f(p \wedge p)$ and $f(p)$.

The logical intensionality problem: Models must be extensional on lambda terms: if f and g are terms, say, of type $\alpha \to \beta$, with α not the type o of truth-values, and if for all individuals u in the model's carrier of type α

$$[\![f]\!] u = [\![g]\!] u$$

then

$$[\![f]\!] = [\![g]\!]$$

However, we need to enforce a certain kind of meta-level failure of *logical extensionality*. Wolfram [22] gives an example of the non-extensional behavior of λProlog, which we translate into Church's type-theoretic notation:

`p X :- q X.`	$\forall x_o.(q_{oo}(x_o) \supset p_{oo}(x_o))$
`q X :- p X.`	$\forall x.(p(x) \supset q(x))$
`r p.`	$r_{o(oo)}(p_{oo})$
`?- r q.`	The goal $r_{o(oo)}(q_{oo})$
`no.`	is not derivable from the program clauses above.

p and q are interpreted as two truth-valued functions which agree on all arguments, which must not be allowed, in general, to have the same meaning in a model.

This is also solved by distinguishing between the carriers of the types of logical formulae and the object Ω of truth-values. The logical equivalence of p and q on predicates takes place in Ω, not in the carrier of their type, so they are not required to have the same denotation in the carrier. This problem was solved using a different technique by Wolfram for the HOH fragment.

The variable indexing problem: We will need to construct a countable hierarchy of universes (signatures), in order to deal with the inconsistency that arises from allowing pre-existing logic variables to unify with constants introduced during execution for the sake of universal goals (see figure 1.1).

We will need a supply of countably many fresh variables and constants at each integer level i and at every type α, as is further explained in the development of the UCTT type theory, below.

The monotonicity problem: Because HOHH formulae include implication, and hence contain negatively occurring subformulae, the obvious choice for a least term model for a logic program is not describable as a union from below of a monotone sequence of finitely generated ICTT-model fragments. We solve this problem by extending Miller's idea of indexed Kripke

Consider the following incorrect resolution "deduction" of

$$\forall x.Qxx \supset \exists y \forall z.Qyz$$

from the empty program.

$$\begin{array}{rl}
 & \texttt{?-}\ \forall x.Qxx \supset \exists y\forall z.Qyz \\
\overset{\supset}{\rightsquigarrow} & \forall x.Qxx\ \texttt{?-}\ \exists y\forall z.Qyz \\
\overset{\exists}{\rightsquigarrow} & \forall x.Qxx\ \texttt{?-}\ \forall z.Qyz \\
\overset{\forall}{\rightsquigarrow} & \forall x.Qxx\ \texttt{?-}\ Qyc \\
\overset{[c/y]}{\rightsquigarrow} & NULL
\end{array}$$

versus the following failed deduction in which a counter keeps track of changing signatures, and the (indexed) logic variable y_0 created when signature 0 was the ambient signature, is inhibited from unification with constant c_1 belonging to signature 1.

$$\begin{array}{rll}
 & 0 & \emptyset\ \texttt{?-}\ \forall x.Qxx \supset \exists y\forall z.Qyz \\
\overset{\supset}{\rightsquigarrow} & 0 & \forall x.Qxx\ \texttt{?-}\ \exists y\forall z.Qyz \\
\overset{\exists}{\rightsquigarrow} & 0 & \forall x.Qxx\ \texttt{?-}\ \forall z.Qy_0z \\
\overset{\forall}{\rightsquigarrow} & 1 & \forall x.Qxx\ \texttt{?-}\ Qy_0c_1 \\
 & & FAIL
\end{array}$$

Figure 1: Why indexing of variables is required

models [15] to allow for fibred or indexed carriers for all the logical components of programs and goals. We have carriers $\mathsf{D}_{i,\mathsf{a}}, \mathsf{D}_{i,\mathsf{r}}, \mathsf{D}_{i,\mathsf{g}}, \mathsf{D}_{i,\mathsf{h}}$ that interpret atoms, rigid atoms, goals and propositions, and the appropriate functions between them (the additional index i is explained in the preceding paragraph). We also define a formal system (UCTT) in which these logical subtypes exist in the syntax.

2 The Calculus

Church's calculus consists of a collection of higher-order functional types together with a simply typed lambda-calculus of terms. Connectives and quantifiers are explicit typed constants in the calculus which permits the formalization of full higher-order logic without adding any new operations. In more detail:

Type expressions are formed from a set of *base types* (which must include at least the type ι of individuals and o of truth values, or *formulae*) via the functional type: if α and β are types then so is $\alpha \longrightarrow \beta$. **Church's notation** for functional type expressions is sometimes more convenient because of its brevity: $(\beta\alpha)$ denotes the type $\alpha \longrightarrow \beta$. Association, in Church's notation, is to the *left*: $\alpha\beta\gamma$ denotes $(\alpha\beta)\gamma$, meaning $\gamma \longrightarrow (\beta \longrightarrow \alpha)$. We will use his notation interchangeably with arrow notation, especially with complex type expressions.

Terms are always associated with a type expression, and are built up from typed constants and typed variables (countably many at each type α): $x^1_\alpha, x^2_\alpha, \ldots, y^1_\alpha, y^2_\alpha, \ldots$ according to the following definition:

1. A variable x_α is a term of type α.

2. A constant of type α is is a term of type α.
3. If t_1 is a term of type $\alpha \longrightarrow \beta$ and t_2 is a term of type α then $t_1 t_2$ is a term of type β.
4. If t is a term of type β then $\lambda x_\alpha.t$ is a term of type $\alpha \longrightarrow \beta$.

Terms may be displayed subscripted with their associated type, when convenient. We omit such subscripts from variables when clear from context, or when the information is of no interest.

Logical Constants The following logical constants are available to define formulas:

$$\top_o, \bot_o, \wedge_{ooo}, \vee_{ooo}, \supset_{ooo},$$

as well as

$$\Sigma_{o(o\alpha)} \quad \text{and} \quad \Pi_{o(o\alpha)},$$

the latter two existing at all types α.

Reduction We recall that the transitive closure of the operation on terms induced by permitting *contraction* of subterms that are β-redices, $\lambda x_\alpha.t\ u$, to $t[u/x]$, the result of substituting (after renaming of bound variables if necessary to avoid capture) u for all free occurrences of x in t, is called β-reduction. Contraction of η-redices $\lambda x.tx$ to t when x does not occur freely in t induces η-reduction. In this paper we will refer to the combination of both reductions, $\beta\eta$-reduction as λ-reduction, and the reflexive, symmetric, transitive closure of this relation as **λ-equivalence.** In Church's Theory, as in Miller and Nadathur's work, a canonical listing of variables is used to ensure the existence of unique canonical normal forms (sometimes denoted $\rho(t)$) for each term t.

Definition 2.1 *A* **formula** *is any term of type o. An atomic formula is a term of type o which is not λ-equivalent to any term whose head is a logical constant.*

The meaning of the logical constants $\wedge$, $\vee$ and $\supset$ is evident, while Σ and Π are used to define the quantifiers $\exists$ and $\forall$: $\exists x_\alpha P_o$ is an abbreviation for $\Sigma_{o(o\alpha)}\lambda x_\alpha.P_o$ and $\forall x_\alpha P_o$ is an abbreviation for $\Pi_{o(o\alpha)}\lambda x_\alpha.P_o$. Also $f(x_\alpha)$ is an abbreviation of $(f_{\beta\alpha}x_\alpha)$.

2.1 Deduction

We will assume familiarity with Gentzen's sequent calculus, and in particular the intuitionistic LJ (see e.g. [12]). We present here (in Fig. 2) a sequent system for the intuitionistic fragment of Church's Theory of Types, which we call ICTT, almost identical to [17] of Miller, Nadathur *et al.*

In fig. 2, all *sequents* $\Gamma \longrightarrow\!\!\!\!\!\rightarrow \Delta$ consist of a *set* of formulas (antecedents) Γ and a single formula Δ (consequent). The rules of inference shown in the figure consist of *premisses* the zero, one or two sequents above the line and *conclusions* the lone sequent below the line. Rules marked with an asterisk must not contain free occurrences of the index variable below the line. In rule λ, the primed and unprimed antecedents (resp. consequents) are λ-equivalent.

The following lemmas will prove useful. Their proofs are routine.

Lemma 2.2 *There is a sequent proof of $\Gamma, B \longrightarrow\!\!\!\!\!\rightarrow B$ for any set of formulas Γ and any formula B.*

Lemma 2.3 *If $\Gamma' \supseteq \Gamma$ and $\Gamma \longrightarrow\!\!\!\!\!\rightarrow \Delta$ then $\Gamma' \longrightarrow\!\!\!\!\!\rightarrow \Delta$.*

Lemma 2.4 *If $\Gamma, B(x_\alpha) \longrightarrow\!\!\!\!\!\rightarrow \Delta$ or $\Gamma \longrightarrow\!\!\!\!\!\rightarrow B(x_\alpha)$ and x_α does not occur in Γ or Δ, then $\Gamma, B(t) \longrightarrow\!\!\!\!\!\rightarrow \Delta$ or $\Gamma \longrightarrow\!\!\!\!\!\rightarrow B(t)$, respectively, for any term t of type α.*

$$\frac{}{\Gamma \longrightarrow\!\!\!\!\!\rightarrow \top} \qquad \frac{}{\Gamma, A \longrightarrow\!\!\!\!\!\rightarrow A} \qquad \frac{}{\Gamma, \bot \longrightarrow\!\!\!\!\!\rightarrow \bot}$$

$$\frac{\Gamma, B, C \longrightarrow\!\!\!\!\!\rightarrow \Delta}{\Gamma, B \wedge C \longrightarrow\!\!\!\!\!\rightarrow \Delta}\ \wedge_L \qquad \frac{\Gamma \longrightarrow\!\!\!\!\!\rightarrow B \quad \Gamma \longrightarrow\!\!\!\!\!\rightarrow C}{\Gamma \longrightarrow\!\!\!\!\!\rightarrow B \wedge C}\ \wedge_R$$

$$\frac{\Gamma, B \longrightarrow\!\!\!\!\!\rightarrow \Delta \quad \Gamma, C \longrightarrow\!\!\!\!\!\rightarrow \Delta}{\Gamma, B \vee C \longrightarrow\!\!\!\!\!\rightarrow \Delta}\ \vee_L \qquad \frac{\Gamma \longrightarrow\!\!\!\!\!\rightarrow B_i}{\Gamma \longrightarrow\!\!\!\!\!\rightarrow B_1 \vee B_2}\ \vee_R$$

$$\frac{\Gamma \longrightarrow\!\!\!\!\!\rightarrow B \quad \Gamma, C \longrightarrow\!\!\!\!\!\rightarrow \Delta}{\Gamma, B \supset C \longrightarrow\!\!\!\!\!\rightarrow \Delta}\ \supset_L \qquad \frac{\Gamma, B \longrightarrow\!\!\!\!\!\rightarrow C}{\Gamma \longrightarrow\!\!\!\!\!\rightarrow B \supset C}\ \supset_R$$

$$\frac{\Gamma, P[t/x] \longrightarrow\!\!\!\!\!\rightarrow \Delta}{\Gamma, \forall x.P \longrightarrow\!\!\!\!\!\rightarrow \Delta}\ \forall_L \qquad \frac{\Gamma \longrightarrow\!\!\!\!\!\rightarrow P}{\Gamma \longrightarrow\!\!\!\!\!\rightarrow \forall x.P}\ \forall_R *$$

$$\frac{\Gamma, P \longrightarrow\!\!\!\!\!\rightarrow \Delta}{\Gamma, \exists x.P \longrightarrow\!\!\!\!\!\rightarrow \Delta}\ \exists_L * \qquad \frac{\Gamma \longrightarrow\!\!\!\!\!\rightarrow P[t/x]}{\Gamma \longrightarrow\!\!\!\!\!\rightarrow \exists x.P}\ \exists_R$$

$$\frac{\Gamma' \longrightarrow\!\!\!\!\!\rightarrow \Delta'}{\Gamma \longrightarrow\!\!\!\!\!\rightarrow \Delta}\ \lambda \qquad \frac{\Gamma \longrightarrow\!\!\!\!\!\rightarrow \bot}{\Gamma \longrightarrow\!\!\!\!\!\rightarrow B}\ \bot_R$$

Figure 2: Higher-order Sequent Rules

Comparison with Church's original calculus Calling this system the intuitionistic fragment of Church's Theory of Types as presented in [6] requires a bit of justification, since his calculus includes a typed equality operator, a choice operator, and, because its logic is classical, all connectives are reduced to disjunction, negation and universal quantification. His proof system is composed of 6 rules of inference and 11 axiom schemas. If we remove those schemas which involve equality and the choice operator, we obtain a set of 6 axioms, the last of which is the clearly classical

$$\forall x_\alpha [p \vee f_{o\alpha}] \supset p \vee \Pi_{o(o\alpha)} f_{o\alpha}.$$

If we leave only axiom schemas 1-5, we obtain a theory (IC) which, it can be shown, is equivalent to ICTT. We will not take the time to establish this here, but it is straightforward to show that IC axioms 1-5 are sound with respect to the semantics presented below, but that axiom 6 fails.

3 Models for ICTT

We define a class of models for ICTT with respect to which ICTT is sound and complete.

Mitchell and Moggi [19] incorporate applicative structures into Kripke models to model equational reasoning in the typed lambda calculus. We use a similar approach to combine applicative structures with Heyting algebra semantics [21, Chap. 13, §4] for constructive logic.

Definition 3.1 *A lattice is a partially ordered set with two operations $\wedge$ and $\vee$, the meet and join, satisfying the following axioms for all members a, b and c of the set:*

$$a \leq a \vee b \qquad b \leq a \vee b \tag{3}$$

$$a \wedge b \leq a \qquad a \wedge b \leq b \tag{4}$$

$$a \le c \text{ and } b \le c \text{ implies } a \vee b \le c \tag{5}$$
$$c \le a \text{ and } c \le b \text{ implies } c \le a \wedge b \tag{6}$$

A lattice is distributive if for all members a, b and c of the partially ordered set,

$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c) \tag{7}$$
$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c) \tag{8}$$

The join $\bigvee S$ of an arbitrary set S, if it exists, is the least upper bound of the set. Likewise, the meet $\bigwedge S$ would be its greatest lower bound.

The meet and join serve to model conjunction and disjunction, but we need an additional operator corresponding to implication. The following definition of a Heyting (or pseudo-Boolean) algebra is taken from [21].

Definition 3.2 *A Heyting algebra H is a lattice with a least element $\perp$ and an operation $\to$ defined on all pairs of elements of H such that*

$$a \wedge b \le c \text{ if and only if } a \le b \to c \tag{9}$$

In a complete Heyting algebra, arbitrary meets and joins exist.

We will use Heyting algebras with certain arbitrary meets and joins corresponding to universal and existential quantification.

In [21] we also learn that a Heyting algebra is distributive and that if $\bigvee B$ exists then

$$a \wedge \bigvee B = \bigvee \{a \wedge b : b \in B\} \tag{10}$$

It is clear from (4) and (9) that a Heyting algebra has a top element, which we denote $\top$.

Example 3.3 *The open subsets of $\mathbb{R}$ form a Heyting algebra with $\vee$ denoting union, $\wedge$ the interior of the intersection, $\le$ the subset relation, $\perp$ the empty set and $a \to b := \bigcup\{w : w \cap a \subseteq b\}$. Then $\top$ is evidently $\mathbb{R}$ and (10) holds, but the analogous statement*

$$a \vee \bigwedge B = \bigwedge \{a \vee b : b \in B\}$$

is false: let a be $\mathbb{R}\backslash\{0\}$ and B be the set of intervals $\{(-\frac{1}{n}, \frac{1}{n}) : n \in \mathbb{N}\}$. Here $a \vee \bigwedge B = a \vee \emptyset = a$ but $\bigwedge\{a \vee b : b \in B\} = \bigwedge\{\mathbb{R} : b \in B\} = \mathbb{R}$.

This false proposition corresponds to the formula $A \vee \forall x.B(x)$ iff $\forall x.(A \vee B(x))$, which is not true in constructive logic and which we therefore do not want to force in all models. This formula corresponds in turn to Axiom 6^α of [6], which makes Church's Theory of Types classical.

3.1 Applicative Structures

We will make use of the notion of applicative structures, a well-known semantical framework for the simply-typed lambda calculus, first introduced by H. Friedman in [11]. (See also [18].)

Definition 3.4 *A typed applicative structure $\langle \mathsf{D}, \mathsf{App}, \mathsf{Const}\rangle$ consists of a set D_α for each type α, a function $\mathsf{App} : \mathsf{D}_{\beta\alpha} \times \mathsf{D}_\alpha \to \mathsf{D}_\beta$ for each pair (α, β) of types, and an intepretation function Const taking constants of type α to elements of D_α.*

A typed applicative structure is extensional if for every f, g in $\mathsf{D}_{\beta\alpha}$, $\mathsf{App}(f,d) = \mathsf{App}(g,d)$ for all $d \in \mathsf{D}_\alpha$ implies $f = g$. Hence any two functions in the language which behave alike semantically must be identified in the semantics.

An assignment φ is a function from the free variables of the language into D which respects types. Given a typed applicative structure D, an environmental model consists of a (total) function $[\![\ ]\!]_\varphi$ from the open terms of the language into D for each assignment φ such that $[\![\lambda x_\alpha.M_\beta]\!]_\varphi \in \mathsf{D}_{\beta\alpha}$ and the following equalities hold:

$$\begin{aligned}
[\![c]\!]_\varphi &= \mathsf{Const}(c) && \text{for constants } c\\
[\![x]\!]_\varphi &= \varphi(x) && \text{for variables } x\\
[\![(MN)]\!]_\varphi &= \mathsf{App}([\![M]\!]_\varphi, [\![N]\!]_\varphi)\\
\mathsf{App}([\![\lambda x_\alpha.M_\beta]\!]_\varphi, d) &= [\![M]\!]_{\varphi[x\to d]}
\end{aligned}$$

The requirement that $[\![\ ]\!]_\varphi$ be a total function, called the *environment model condition* in [18], insures that all lambda terms will have an interpretation.

Now we can define models for our intuitionistic theory of types. We add certain restrictions to typed applicative structures for ICTT to handle the logical constants and predicates.

Definition 3.5 *A Heyting applicative structure $\langle \mathsf{D}, \mathsf{App}, \mathsf{Const}, \Omega\rangle$ for* ICTT *is a typed applicative structure with an associated Heyting algebra Ω and function ω from D_o to Ω such that for each f in $\mathsf{D}_{o\alpha}$, Ω contains the meets and joins*

$$\bigwedge\{\omega(\mathsf{App}(f,d)) : d \in \mathsf{D}_\alpha\} \text{ and } \bigvee\{\omega(\mathsf{App}(f,d)) : d \in \mathsf{D}_\alpha\},$$

and the following conditions are satisfied:

$$\begin{aligned}
\omega(\mathsf{Const}(\top_o)) &= \top_\Omega\\
\omega(\mathsf{Const}(\bot_o)) &= \bot_\Omega\\
\omega(\mathsf{App}(\mathsf{App}(\mathsf{Const}(\wedge_{ooo}), d_1), d_2)) &= \omega(d_1) \wedge \omega(d_2)\\
\omega(\mathsf{App}(\mathsf{App}(\mathsf{Const}(\vee_{ooo}), d_1), d_2)) &= \omega(d_1) \vee \omega(d_2)\\
\omega(\mathsf{App}(\mathsf{App}(\mathsf{Const}(\supset_{ooo}), d_1), d_2)) &= \omega(d_1) \to \omega(d_2)\\
\omega(\mathsf{App}(\mathsf{Const}(\Sigma_{o(o\alpha)}), f)) &= \bigvee\{\omega(\mathsf{App}(f,d)) : d \in \mathsf{D}_\alpha\}\\
\omega(\mathsf{App}(\mathsf{Const}(\Pi_{o(o\alpha)}), f)) &= \bigwedge\{\omega(\mathsf{App}(f,d)) : d \in \mathsf{D}_\alpha\}
\end{aligned}$$

A definition identifying D_o with Ω (*i.e.*,, restricting ω to the identity function) would be more natural but would make $A \wedge B$ indiscernable from $B \wedge A$ in the model and thereby identify the truth values of $P_{oo}(A_o \wedge B_o)$ and $P_{oo}(B_o \wedge A_o)$. This identity holds neither in ICTT as presented here nor in HOHH, to be considered in the next chapter.

An assignment is a function from the variables in the language to the elements of the model which allows us to give meaning to open terms. An assignment φ must satisfy $\varphi(x_\alpha) \in \mathsf{D}_\alpha$.

Definition 3.6 *A model for* ICTT *is an assignment-indexed interpretation function $\mathfrak{H} = \{\mathfrak{H}(\ )_\varphi : \varphi \text{ an assignment}\}$ into a Heyting applicative structure $\langle \mathsf{D}, \mathsf{App}, \mathsf{Const}, \Omega\rangle$ which takes (possibly open) terms of type α into D_α and satisfies the following:*

$$\begin{aligned}
\mathfrak{H}(c)_\varphi &= \mathsf{Const}(c) && \textit{for constants } c\\
\mathfrak{H}(x)_\varphi &= \varphi(x) && \textit{for variables } x\\
\mathfrak{H}(M)_\varphi &= \mathfrak{H}(N)_\varphi && M\ \lambda\textit{-equivalent to } N\\
\mathfrak{H}((MN))_\varphi &= \mathsf{App}(\mathfrak{H}(M)_\varphi, \mathfrak{H}(N)_\varphi)\\
\mathsf{App}(\mathfrak{H}(\lambda x_\alpha.M_\beta)_\varphi, d) &= \mathfrak{H}(N)_{\varphi[d/x]})
\end{aligned}$$

Given a model $\mathfrak{H}$ and an assignment φ, we say that φ satisfies B in $\mathfrak{H}$ if $\omega(\mathfrak{H}(B_o)_\varphi) = \top_\Omega$; we abbreviate this assertion to $\mathfrak{H} \models_\varphi B_o$. We say B_o is valid in $\mathfrak{H}$ (equivalently, $\mathfrak{H} \models B_o$) if $\mathfrak{H} \models_\varphi B_o$ for every assignment φ. We abbreviate $\omega(\mathfrak{H}(B_o)_\varphi)$ to $[\![B_o]\!]_\varphi$. We also omit the subscript φ when our intentions are clear.

3.2 Soundness and Completeness

The following lemma is easy to prove and motivates the definition given below of the meaning of a sequent in a model.

Lemma 3.7 *There is a sequent proof of* $\Gamma \longrightarrow\!\!\!\!\!\rightarrow B \supset C$ *if and only if there is a sequent proof of* $\Gamma, B \longrightarrow\!\!\!\!\!\rightarrow C$.

Definition 3.8 (Truth of sequents in a model) *We define the meaning of a sequent in a model by*

$$[\![\Gamma \longrightarrow\!\!\!\!\!\rightarrow \Delta]\!] := [\![\bigwedge \Gamma \supset \Delta]\!]$$

where $\bigwedge \Gamma$ *signifies the conjunction of the elements of* Γ *and where we recall that, in an intuitionistic calculus, the consequent* Δ *is restricted to a single formula.*

Note that $[\![\bigwedge \Gamma \supset \Delta]\!] = \top$ if and only if $\top \leq [\![\bigwedge \Gamma \supset \Delta]\!]$, which is to say $\top \leq [\![\bigwedge \Gamma]\!] \to [\![\Delta]\!]$, which by the condition on $\to$ is equivalent to $\top \wedge [\![\bigwedge \Gamma]\!] \leq [\![\Delta]\!]$, which in turn is equivalent to $[\![\bigwedge \Gamma]\!] \leq [\![\Delta]\!]$ by (4) and (6). We will abbreviate $[\![\bigwedge \Gamma]\!]$ to $[\![\Gamma]\!]$ and use the formulation $[\![\Gamma]\!] \leq [\![\Delta]\!]$ or more explicitly, $[\![\Gamma]\!]_\varphi \leq [\![\Delta]\!]_\varphi$, henceforth.

The proofs of the following theorems can be found in [8, 9]

Theorem 3.9 (Soundness) *If* $\Gamma \longrightarrow\!\!\!\!\!\rightarrow \Delta$ *is provable in* ICTT *then* $[\![\Gamma \longrightarrow\!\!\!\!\!\rightarrow \Delta]\!]$ *equals* $\top_\Omega$ *in every model* $\mathfrak{H}$ *of* ICTT.

Theorem 3.10 (Completeness) *If* $[\![\Gamma]\!] \leq [\![\Delta]\!]$ *in all models* $\mathfrak{H}$*, then there is a sequent proof of* $\Gamma \longrightarrow\!\!\!\!\!\rightarrow \Delta$.

4 Stratifying logic to reflect Program Structure

In our definition of an operational logic and proof theory we formalize some of the metamathematics implicit in the interpretation and logic programs and in the structure of logical states. Since indexed constants and signature levels, and program-goal pairs are handled explicitly in the interpreter and in run-time logical states, they appear formally in the syntax. Each type is assumed to have indexed variables and constants. Programs and goals and their fundamental components have their own logical subtypes. In order not to encumber the syntax excessively, or make too great a departure from the type theory on which it is based, we have refrained from introducing indexed types to reflect computation stages, but in the *semantics* we will have stratified carriers for terms at different index levels. Our operational formal system is called UCTT, for the Uniform-proof (resolution) version of Church's Theory of types.

Before presenting the calculus we review the definition of HOHH programs and goals, upon which UCTT based.

4.1 Higher-order Programs

Let $\mathcal{H}$ be the set of *positive terms*, the (λ-normal form) terms which do not include $\perp$ or $\supset$; A will refer to an atomic formula over $\mathcal{H}$ and A_r to a rigid atomic formula over $\mathcal{H}$. In this case, an atomic formula is a λ-normal form term whose top-level symbol is not a logical constant, and a rigid atomic formula is an atomic formula whose top level symbol is also not a variable. Let G refer to a goal formula in λ-normal form, D to a definite clause (that is, a single program formula), and P to a set of program formulae; $P \cup D$ will stand for $P \cup \{D\}$.

Definition 4.1 HOHH *program formulae D and goal formulae G are given by:*

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

The HOHH$^+$ fragment is similar to this one, but A_r is restricted not to rigid atoms but to atomic formulae in which the head may not be "essentially existentially quantified". λ-Prolog is, in fact, slightly stronger, even, than this fragment. See [14] for details.

In UCTT rigid atoms, atoms, programs and goals are simply the terms of the corresponding *subtypes* of the type h of logical formulae. Certain constants and functions are barred from the calculus to make it impossible to form illegal programs or goals. In particular, all terms appearing in atomic formulae are positive.

4.2 The UCTT calculus

The base types are the types ι of individuals, h of truth-values, a of atoms, r of rigid atoms, p of programs and g of goals. All other types are functional types $\beta\alpha$. The logical constants are $\top_{\mathsf{g}}$, $\top_{\mathsf{h}}$, $\wedge_{\mathsf{ppp}}$, $\wedge_{\mathsf{ggg}}$, $\wedge_{\mathsf{hhh}}$, $\vee_{\mathsf{ggg}}$, $\vee_{\mathsf{hhh}}$, $\supset_{\mathsf{gpp}}$, $\supset_{\mathsf{prg}}$, $\Sigma_{\mathsf{g(g\alpha)}}$, $\Sigma_{\mathsf{h(h\alpha)}}$, $\Pi_{\mathsf{g(g\alpha)}}$, $\Pi_{\mathsf{p(p\alpha)}}$ and $\Pi_{\mathsf{h(h\alpha)}}$. There may be no other constants or variables of types p or g, nor of functional types involving p or g. Constants of type $\mathsf{r}\alpha$ and both constants and variables of type $\mathsf{a}\alpha$ are allowed but no other functional types involving r or a are.

4.3 Signatures

As explained in an earlier section and in figure (1.1), we will need to construct a countable hierarchy of universes (signatures), in order to deal with the inconsistency that arises from allowing pre-existing logic variables to unify with constants introduced during execution for the sake of universal goals.

Let $\mathcal{U}_0(\alpha)$ include all constants and variables of type α appearing in the original programs and goals along with countably many more logic variables; constants and variables of $\mathcal{U}_0$ are considered to be labelled with the integer 0. The set $\mathcal{U}_i(\alpha)$ will include $\mathcal{U}_{i-1}(\alpha)$ along with a new set of countably many logic variables and countably many fresh constants of type α.

A (positive) term over $\mathcal{U}_i$ contains only typed constants and variables from the set $\mathcal{U}_i(\alpha)$. A program over $\mathcal{U}_i$ is a program whose terms are all over $\mathcal{U}_i$. In the models, the restriction $f_i(c_j) \in \mathcal{U}_{\max\{i,j\}}$ will be desirable, but in the syntax λ-conversions will not introduce undesirable indices.

4.4 UCTT Resolution Rules for HOHH formulae

From now on, programs-formulae, goals, rigid atoms and atoms refer to λ-normal forms of terms of the correspoinding types $\mathsf{p}, \mathsf{g}, \mathsf{r}$ and a. We now present the proof theory for UCTT. Only goals may occur as theorems, only signature-program pairs as premisses.

Instead of a sequent system, we use an operational proof system so as better to model the behavior of programs. Rather than automate the $\wedge_L$ and $\forall_L$ rules of the sequent calculus explicitly, we decompose the program into conjuncts and instances. For technical reasons we retain the original program clauses as well.

Definition 4.2 *For a given program P over $\mathcal{U}_i$, the* extended program over $\mathcal{U}_i$, *denoted $[i;P]$, is defined recursively as follows:*

- *If $D \in P$, then $D \in [i;P]$.*
- *If $D_1 \wedge D_2 \in [i;P]$, then $D_1 \in [i;P]$ and $D_2 \in [i;P]$.*
- *If $\forall x_\alpha.D_\beta \in [i;P]$, then $\rho(D(t)) \in [i;P]$ for all positive terms t over $\mathcal{U}_i(\alpha)$.*

Note that we often identify programs, as sets of formulae, with the single formula obtained by taking the conjunction of its members, which leads to some abuse of notation. If P is a program formula, the first clause of the definition above would make no sense unless it read $P \in [i;P]$. But we understand it to apply to the program $\{P\}$.

Besides the familiar index restrictions, substitution must also respect sorts.

Definition 4.3 *Let the signatures be given as above.*

A **state** *is a trio $\langle i;P \mid G\rangle$ where P is a program and G a goal, both over the signature $\mathcal{U}_i$; i is referred to as the* **state index**.

A **state vector** *$\langle i_1;P_1 \mid G_1\rangle, \ldots, \langle i_j;P_j \mid G_j\rangle, \ldots, \langle i_n;P_n \mid G_n\rangle$ is a finite sequence of states. State vectors will be abbreviated to $\mathfrak{A}$, $\langle i;P \mid G\rangle$, $\mathfrak{B}$, where $\mathfrak{A}$ and $\mathfrak{B}$ denote (possibly empty) finite sequences of states.*

A **null** *resolution vector is a state vector of the form $\langle i_1;P_1 \mid \top\rangle, \ldots, \langle i_n;P_n \mid \top\rangle$. It may be the empty sequence.*

Note that the NULL vector may itself be a derivation with no steps.

State vectors are required on several accounts: the index provides for manipulation of universal quantification; the program is included because it may be augmented to prove conditional goals; and thus a vector of multiple states is necessary for the decomposition of conjunctive goals, since proof of the conjuncts may require different manipulations of index and program.

When a substitution is applied to a program P, it is understood to be applied to each clause of P. When applied to a state vector $\mathfrak{A}$, it is understood to be applied to the program and to the goal of each state in the vector. For example, if $\mathfrak{A}$ were the vector $\langle i_1;P_1 \mid G_1\rangle, \ldots, \langle i_n;P_n \mid G_n\rangle$, then $\mathfrak{A}\theta$ would be $\langle i_1;P_1\theta \mid G_1\theta\rangle, \ldots, \langle i_n;P_n\theta \mid G_n\theta\rangle$.

We will call a substitution *legal* if it is legal in the usual sense of avoiding variable capture and is in addition restricted by the labels, so that only terms over the universe $\mathcal{U}_i$ are be substituted for a logic variable of label i. In the case of quantified variables, however, we use bracket notation to indicate that the replacement is made in accordance with the resolution rules for quantifiers regardless of the indices involved.

Resolution rules The resolution reduction rules for HOHH are nearly identical to those of the first-order case. We make minor changes for handling higher-order terms. *Legal* substitutions preserve index-level and types. The operator ρ appearing in the *instance* and *generic* rules is lifted from Andrews and Miller's papers. It is a map from lambda-terms to some suitably chosen *canonical* representation of their normal forms, obtained, e.g. by fixing some canonical order on all variables.

identity $\mathfrak{A},\ \langle i;P \mid A'\rangle,\ \mathfrak{B} \overset{\theta}{\leadsto} \mathfrak{A}\theta,\ \mathfrak{B}\theta$

where θ is legal, A atomic, $A'\theta =_\beta A\theta$, and $A \in [i;P]$.

backchain $\mathfrak{A},\ \langle i;P \mid A'\rangle,\ \mathfrak{B} \overset{\theta}{\leadsto} \mathfrak{A}\theta,\ \langle i;P\theta \mid G\theta\rangle,\ \mathfrak{B}\theta$

where θ is legal, A atomic, $(G \supset A) \in [i;P]$, and $A'\theta =_\beta A\theta$.

or$_j$ $\mathfrak{A},\ \langle i; P \mid G_1 \vee G_2\rangle,\ \mathfrak{B} \overset{\vee}{\rightsquigarrow} \mathfrak{A},\ \langle i; P \mid G_j\rangle,\ \mathfrak{B}$

and $\mathfrak{A},\ \langle i; P \mid G_1 \wedge G_2\rangle,\ \mathfrak{B} \overset{\wedge}{\rightsquigarrow} \mathfrak{A},\ \langle i; P \mid G_1\rangle,\ \langle i; P \mid G_2\rangle,\ \mathfrak{B}$

instance $\mathfrak{A},\ \langle i; P \mid \exists x.G\rangle,\ \mathfrak{B} \overset{\exists}{\rightsquigarrow} \mathfrak{A},\ \langle i; P \mid \rho((\lambda x.G)(v_i))\rangle,\ \mathfrak{B}$

where v_i is a fresh logic variable of label i.

augment $\mathfrak{A},\ \langle i; P \mid D \supset G\rangle,\ \mathfrak{B} \overset{\supset}{\rightsquigarrow} \mathfrak{A},\ \langle i; P \cup D \mid G\rangle,\ \mathfrak{B}$

generic $\mathfrak{A},\ \langle i; P \mid \forall x.G\rangle,\ \mathfrak{B} \overset{\forall}{\rightsquigarrow} \mathfrak{A},\ \langle i+1; P \mid \rho((\lambda x.G)(c_{i+1}))\rangle,\ \mathfrak{B}$

where c_{i+1} is a fresh constant of label $i+1$.

4.5 Uniform Algebras

Our model theory for UCTT is a variant of Heyting Applicative Structure semantics for the intuitionistic version of Church's Theory of types sketched earlier in the paper. It brings together several of the ideas discussed above: Miller's indexed model theory for a first-order fragment of FOHH (we assign meaning to goals G *with respect to ambient programs* P), stratification of carriers to represent different index levels, separate carriers for programs and goals and their components, and an object Ω of truth values (a lattice with enough infinite suprema) distinct from the carrier of logical formulae. Like Miller's model, carriers of programs have a Kripke-like (weak) poset structure. Unlike Miller's model, the carriers of programs and goals do not consist of the programs and goals themselves, but of interpreted programs and goals. Thus our models are not term-models by default as far as formulae are concerned.

The interpretation of logic (program-goal pairs) in our semantics arises from the composition of two maps:

$$\mathsf{p} \times \mathsf{g} \overset{\mathfrak{U}\times\mathfrak{U}}{\longrightarrow} \mathsf{D}_{\mathsf{p}} \times \mathsf{D}_{\mathsf{g}} \overset{\mathcal{I}}{\longrightarrow} \Omega$$

where p and g stand for the sets of terms of program and goal type respectively, mapped by $\mathfrak{U}$ into carriers of their type in some *Uniform Applicative Structure* $\mathfrak{D}$, defined below, and $\mathcal{I} = \{\mathcal{I}_i : i \in \mathbb{N}\}$ is an integer-indexed family of maps from stratified carriers $\mathsf{D}_{i,\mathsf{p}} \times \mathsf{D}_{i,\mathsf{g}}$ into the object of truth values Ω. Readers familiar with Miller's forcing notation [15] will see that our interpretation $I_i(\mathfrak{U}(P), \mathfrak{U}(G)) = \top_\Omega$ of the program-goal pair P, G corresponds to his

$$\mathcal{I}, P \Vdash G.$$

But we generalize his forcing in a number of ways: we include universal quantification (which requires adding the signature-levels $i \in \mathbb{N}$), higher-order logic (which requires the presence of an applicative structure to model lambda-abstraction and application), and we factor our interpretation through a structure which supplies carriers $\mathsf{D}_p, \mathsf{D}_g$ for programs as well as goals.

We now give the details.

Definition 4.4 *A uniform applicative structure* $\langle \mathsf{D}, \mathsf{App}, \mathsf{Const}, \mathsf{Inc}\rangle$ *for* UCTT *is a typed applicative structure with sets* $\mathsf{D}_{i,\alpha}$ *for each index* i *and type* α*, and functions* $\mathsf{Inc}_{\mathsf{pr}} : \mathsf{D}_{i,\mathsf{r}} \to \mathsf{D}_{i,\mathsf{p}}$*,* $\mathsf{Inc}_{\mathsf{ga}} : \mathsf{D}_{i,\mathsf{a}} \to \mathsf{D}_{i,\mathsf{g}}$*,* $\mathsf{Inc}_{\mathsf{ar}} : \mathsf{D}_{i,\mathsf{r}} \to \mathsf{D}_{i,\mathsf{a}}$*,* $\mathsf{Inc}_{\mathsf{gh}} : \mathsf{D}_{i,\mathsf{h}} \to \mathsf{D}_{i,\mathsf{g}}$ *and* $\mathsf{Inc}_{(i+1)i} : \mathsf{D}_{i,\alpha} \to \mathsf{D}_{i+1,\alpha}$ *(and by composition* Inc_{jk} *for* $j > k$*). The interpretation of functions satisfies* $\mathsf{App}(f,t) \in \mathsf{D}_{\max\{i,j\},\beta}$ *for all* f *in* $\mathsf{D}_{i,\beta\alpha}$ *and* $t \in \mathsf{D}_{j,\alpha}$ *and* $\mathsf{Inc}_{ij}(\mathsf{App}(g,s)) = \mathsf{App}(\mathsf{Inc}_{ij}(g), \mathsf{Inc}_{ij}(s))$*.*

There is a distinguished element $\top$ *of* $\mathsf{D}_{0,\mathsf{g}}$*. Also* $\langle \mathsf{D}_{i,\mathsf{p}}, \le_{\mathsf{p}}, \wedge_{\mathsf{ppp}}, \Pi_{\mathsf{p}(\mathsf{p}\alpha)}\rangle$ *must be a pre-order with a commutative binary operation* $\wedge_{\mathsf{ppp}}$ *and a unary operator* $\Pi_{\mathsf{p}(\mathsf{p}\alpha)}$*, such that for* $p, q \in \mathsf{D}_{i,\mathsf{p}}$ *and* $f \in \mathsf{D}_{i,\mathsf{p}\alpha}$*,* $p \le_{\mathsf{p}} p$*,* $p \le_{\mathsf{p}} p \wedge q$*,* $\le_{\mathsf{p}}$ *is transitive and* $\mathsf{App}(f,d) \le_{\mathsf{p}} \mathsf{App}(\Pi_{\mathsf{p}(\mathsf{p}\alpha)}, f)$*.*

Next, we define an interpretation function from the uniform applicative structure into a Heyting algebra Ω. We distinguish the Heyting algebra operations with subscripts Ω where necessary.

Definition 4.5 *An* **intepretation** *on a uniform applicative structure is a function* $\mathcal{I} : i \times \mathsf{D}_{i,\mathsf{p}} \times \mathsf{D}_{i,\mathsf{g}} \to \Omega$, $i \in \mathbb{N}$, *defined for all goals with the condtion*

$$\mathcal{I}_i(\mathsf{Inc}_{\mathsf{pr}}(r), \mathsf{Inc}_{\mathsf{ga}}(\mathsf{Inc}_{\mathsf{ar}}(r))) = \top_\Omega \tag{11}$$

for atoms and the following more familiar equalities involving goals:

$$\mathcal{I}_i(p, \top) = \top_\Omega \tag{12}$$

$$\mathcal{I}_i(p, g \wedge_{\mathsf{ggg}} g') = \mathcal{I}_i(p, g) \wedge_\Omega \mathcal{I}_i(p, g') \tag{13}$$

$$\mathcal{I}_i(p, g \vee_{\mathsf{ggg}} g') = \mathcal{I}_i(p, g) \vee_\Omega \mathcal{I}_i(p, g') \tag{14}$$

$$\mathcal{I}_i(p, q \supset_{\mathsf{ggp}} g) = \mathcal{I}_i(p \wedge_{\mathsf{ppp}} q, g) \tag{15}$$

$$\mathcal{I}_i(p, \mathsf{App}(\Sigma_{\mathsf{g}(\mathsf{g}\alpha)}, f)) = \bigvee_{d \in \mathsf{D}_{i,\alpha}} \mathcal{I}_i(p, \mathsf{App}(f, d)) \qquad \text{for } f \in \mathsf{D}_{i,\mathsf{g}\alpha} \tag{16}$$

$$\mathcal{I}_i(p, \mathsf{App}(\Pi_{\mathsf{g}(\mathsf{g}\alpha)}, f)) = \bigvee_{d \in S} \mathcal{I}_{i+1}(p, \mathsf{App}(f, d)) \qquad \text{for } f \in \mathsf{D}_{i,\mathsf{g}\alpha} \tag{17}$$

where $S = \mathsf{D}_{i+1,\alpha} \setminus \mathsf{Inc}_{(i+1)i}(\mathsf{D}_{i,\alpha})$. *We also add the following conditions on* $\leq_{\mathsf{p}}$*:*

$$p \leq_{\mathsf{p}} p' \text{ implies } \mathcal{I}_i(p, g) \leq_\Omega \mathcal{I}_i(p', g) \tag{18}$$

$$(g \supset_{\mathsf{pgr}} r) \leq_{\mathsf{p}} p \text{ implies } \mathcal{I}_i(p, g) \leq_\Omega \mathcal{I}_i(p, \mathsf{Inc}_{\mathsf{ga}}(\mathsf{Inc}_{\mathsf{ar}}(r))) \tag{19}$$

And finally, we interpret the logic in our structure.

Definition 4.6 *A* **uniform algebra** *or* **model** *for* UCTT *is a uniform applicative structure together with an assignment-indexed map* $\mathfrak{U}$ *from terms to* D *sending elements of* $\mathcal{U}_i(\alpha)$ *to elements of* $\mathsf{D}_{i,\alpha}$, *in an injective fashion; that is,* $\mathfrak{U}(c) \notin \mathsf{Inc}_{(i+1)i}(\mathfrak{U}(\mathcal{U}_i(\alpha)))$ *for the fresh constants* $c \in \mathcal{U}_{i+1}(\alpha)$. *We restrict the interpretation of functions so that* $\mathsf{App}(\mathfrak{U}(\lambda x.t)_\varphi, d) = \mathfrak{U}(t)_{\varphi[d/x]}$ *and require that the following simple conditions also be met:*

$$\begin{aligned}
\mathfrak{U}(\top_{\mathsf{g}}) &= \top \\
\mathfrak{U}(t_1 t_2)_\varphi &= \mathsf{App}(\mathfrak{U}(t_1)_\varphi, \mathfrak{U}(t_2)_\varphi) \\
\mathfrak{U}(P_1 \wedge P_2)_\varphi &= \mathfrak{U}(P_1)_\varphi \wedge_{\mathsf{ppp}} \mathfrak{U}(P_2)_\varphi \\
\mathfrak{U}(G_1 \wedge G_2)_\varphi &= \mathfrak{U}(G_1)_\varphi \wedge_{\mathsf{ggg}} \mathfrak{U}(G_2)_\varphi \\
\mathfrak{U}(G_1 \vee G_2)_\varphi &= \mathfrak{U}(G_1)_\varphi \vee_{\mathsf{ggg}} \mathfrak{U}(G_2)_\varphi \\
\mathfrak{U}(G \supset A_r)_\varphi &= \mathfrak{U}(G)_\varphi \supset_{\mathsf{prg}} \mathfrak{U}(A_r)_\varphi \\
\mathfrak{U}(P \supset G)_\varphi &= \mathfrak{U}(P)_\varphi \supset_{\mathsf{ggp}} \mathfrak{U}(G)_\varphi \\
\mathfrak{U}(\Sigma\lambda x.G)_\varphi &= \mathsf{App}(\Sigma_{\mathsf{g}(\mathsf{g}\alpha)}, \mathfrak{U}(\lambda x.G)_\varphi) \\
\mathfrak{U}(\Pi\lambda x.P)_\varphi &= \mathsf{App}(\Pi_{\mathsf{p}(\mathsf{p}\alpha)}, \mathfrak{U}(\lambda x.P)_\varphi) \\
\mathfrak{U}(\Pi\lambda x.G)_\varphi &= \mathsf{App}(\Pi_{\mathsf{g}(\mathsf{g}\alpha)}, \mathfrak{U}(\lambda x.G)_\varphi)
\end{aligned}$$

The notation $\mathfrak{U}(P)$ *where* P *is a set of program formulae rather than a single program formula, is meant to denote* $\mathfrak{U}(\bigwedge P)$.

We have preprocessed the set of programs into equivalence classes by identifying P with $\bigwedge P$, but if a finer model theory were desired, we could add operators $\cup_{\mathsf{ppp}}$ and $\subseteq_{\mathsf{ppp}}$ to the definition of uniform algebra.

Lemma 4.7 (Substitution) *If* θ *is a substitution* $\{(x_1, t_1), \ldots, (x_n, t_n)\}$, *abbreviated* $[\overline{t}/\overline{x}]$, *then for any term* N, $\mathfrak{U}(N\theta)_\varphi = \mathfrak{U}(N)_{\varphi[\overline{\mathfrak{U}(t)_\varphi}/\overline{x}]}$.

4.6 Soundness and Completeness

The reader will notice that the Soundness and Completeness Theorems below are restricted to closed goals proved by closed programs. This condition does not restrict the expressive power of the language because whatever is supposed to be expressed by an open program or goal can be expressed in closed form using universal and existential quantification, respectively. Even variable-sharing between a program P and a goal G can be expressed using existential quantification of a new goal $P' \supset G$ where P' is the conjunction of the clauses of P and the new program is empty.

To prove soundness and completeness we again make a distinction between the universes $\mathcal{U}_i$ and their ground versions $\mathcal{V}_i$.

Theorem 4.8 *Suppose $\mathfrak{D}$ is a state vector, $x_1, \ldots, x_m$ are the logic variables of $\mathfrak{D}$, j is the largest index appearing in $\mathfrak{D}$, and there is an SLD resolution of NULL from $\mathfrak{D}$. Then there are positive terms $s_1, \ldots, s_m$ over $\mathcal{V}_j$ such that for all $\langle i; P \mid G' \rangle$ in $\mathfrak{D}$, in any uniform algebra,*

$$\mathcal{I}_i(\mathfrak{U}(P[s_1/x_1, \ldots, s_m/x_m]), \mathfrak{U}(G'[s_1/x_1, \ldots, s_m/x_m])) = \top_\Omega.$$

Proof. The proof is by induction on the length n of the SLD derivation. Where convenient, we abbreviate $[s_1/x_1, \ldots, s_m/x_m]$ to $[\overline{s}/\overline{x}]$.

Base case: If $n = 0$ then the vector is of the form $\langle i; P_1 \mid \top \rangle \cdots \langle i; P_n \mid \top \rangle$. Then $\mathcal{I}_i(\mathfrak{U}(P_j[\overline{s}/\overline{x}]), \top) = \top_\Omega$ by (12), as desired.

Inductive cases: Assume that the theorem holds for all SLD derivations of length less than n. We consider the possible outermost resolution steps:

Without loss of generality, we consider the or$_1$ rule, $\mathfrak{A}$, $\langle i; P \mid G_1 \vee G_2 \rangle$, $\mathfrak{B} \overset{\vee}{\rightsquigarrow} \mathfrak{A}$, $\langle i; P \mid G_1 \rangle$, $\mathfrak{B}$. By induction we assume that for all states $\langle i; Q \mid H \rangle$ in $\mathfrak{A}$, $\langle i; P \mid G_1 \rangle$, $\mathfrak{B}$, we have

$$\mathcal{I}_i(\mathfrak{U}(Q[s_1/x_1, \ldots, s_k/x_k], \mathfrak{U}(H[s_1/x_1, \ldots, s_k/x_k])) = \top_\Omega.$$

Let $x_{k+1}, \ldots, x_m$ denote those logic variables of G_2 not among $x_1, \ldots, x_k$. Select any positive terms $s_{k+1}, \ldots, s_m$. By (14),

$$\mathcal{I}_i(\mathfrak{U}(P[s_1/x_1, \ldots, s_m/x_m]), \mathfrak{U}((G_1 \vee G_2)[s_1/x_1, \ldots, s_m/x_m])) = \top_\Omega,$$

and thus all the states of the desired state vector are true.

If the last SLD-step involved $\langle i; P \mid G_1 \wedge G_2 \rangle \overset{\wedge}{\rightsquigarrow} \langle i; P \mid G_1 \rangle$, $\langle i; P \mid G_2 \rangle$, we assume that both $\mathcal{I}_i(\mathfrak{U}(P), \mathfrak{U}(G_1)) = \top_\Omega$ and $\mathcal{I}_i(\mathfrak{U}(P), \mathfrak{U}(G_2)) = \top_\Omega$ under the substitution $[\overline{s}/\overline{x}]$. Then $\mathcal{I}_i(\mathfrak{U}(P[\overline{s}/\overline{x}]), \mathfrak{U}((G_1 \wedge G_2)[\overline{s}/\overline{x}])) = \top_\Omega$ follows from (13).

If the last SLD-step involved $\langle i; P \mid D \supset G \rangle \overset{\supset}{\rightsquigarrow} \langle i; P \cup D \mid G \rangle$, assume that

$$\mathcal{I}_i(\mathfrak{U}((P \cup D)[\overline{s}/\overline{x}]), \mathfrak{U}(G[\overline{s}/\overline{x}])) = \top_\Omega.$$

Then $\mathcal{I}_i(\mathfrak{U}(P[\overline{s}/\overline{x}]), \mathfrak{U}((D \supset G)[\overline{s}/\overline{x}])) = \top_\Omega$ by (15).

In the case of $\mathfrak{A}$, $\langle i; P \mid A' \rangle$, $\mathfrak{B} \overset{\theta}{\rightsquigarrow} \mathfrak{A}\theta$, $\mathfrak{B}\theta \rightsquigarrow \cdots \rightsquigarrow$ NULL, we have $A'\theta = A\theta$, A' atomic, and $A \in [i; P]_{\mathcal{U}}$. We assume

$$\mathcal{I}_i(\mathfrak{U}(Q\theta[r_1/y_1, \ldots, r_l/y_l]), \mathfrak{U}(H\theta[r_1/y_1, \ldots, r_l/y_l])) = \top_\Omega$$

for each state $\langle i; Q\theta \mid H\theta \rangle$ in $\mathfrak{A}\theta, \mathfrak{B}\theta$ and rewrite $\theta \circ [r_1/y_1, \ldots, r_l/y_l]$ in the required form as $[s_1/x_1, \ldots, s_k/x_k]$. Thus we have

$$\mathcal{I}_i(\mathfrak{U}(Q[s_1/x_1, \ldots, s_k/x_k]), \mathfrak{U}(H[s_1/x_1, \ldots, s_k/x_k])) = \top_\Omega$$

for each state $\langle i; Q \mid H \rangle$ in $\mathfrak{A}$, $\mathfrak{B}$. We extend the above substitution to $\theta' = [s_1/x_1, \ldots, s_m/x_m]$, $m \geq k$, which replaces any remaining logic variables in P or A' by fresh variables. Note that the states of $\mathfrak{A}\theta'$ and $\mathfrak{B}\theta'$ are still sent to $\top_\Omega$. Since $A\theta' \in [i; P\theta']_{\mathcal{V}}$, $\mathfrak{U}(A\theta') \leq_{\mathsf{p}} \mathfrak{U}(P\theta')$, and by (11) and (18)

$$\top_\Omega =_\Omega \mathcal{I}_i(\mathsf{Inc}(\mathfrak{U}(A\theta')), \mathsf{Inc}(\mathfrak{U}(A\theta'))) \leq_\Omega \mathcal{I}_i(\mathfrak{U}(P\theta'), \mathsf{Inc}(\mathfrak{U}(A\theta'))).$$

In the case of $\mathfrak{A}$, $\langle i; P \mid A'\rangle$, $\mathfrak{B} \overset{\theta}{\rightsquigarrow} \mathfrak{A}\theta$, $\langle i; P\theta \mid G\theta\rangle$, $\mathfrak{B}\theta$ for $G \supset A \in [i; P]_{\mathcal{U}}$ and $A\theta = A'\theta$, we assume that

$$\mathcal{I}_i(\mathfrak{U}(Q\theta[r_1/y_1, \ldots, r_k/y_k]), \mathfrak{U}(H\theta[r_1/y_1, \ldots, r_k/y_k])) = \top_\Omega$$

holds for each state $\langle i; Q\theta \mid H\theta\rangle$ of the latter state vector. As in the previous case, we rewrite $\theta \circ [r_1/y_1, \ldots, r_l/y_l]$ in the required form and extend it if necessary to $\theta' = [s_1/x_1, \ldots, s_m/x_m]$, replacing any logic variables in $G \supset A$ with free variables. Again, $\mathcal{I}_i(\mathfrak{U}(Q\theta'), \mathfrak{U}(H\theta')) = \top_\Omega$ for all states in $\mathfrak{A}$, $\langle i; P \mid G\rangle$, $\mathfrak{B}$, $A'\theta' = A\theta'$, and $(G \supset A)\theta' \in [i; P\theta']_{\mathcal{V}}$. Hence $\mathfrak{U}((G \supset A)\theta') \leq_{\mathsf{p}} \mathfrak{U}(P\theta')$ and using (19),

$$\top_\Omega = \mathcal{I}_i(\mathfrak{U}(P\theta'), \mathfrak{U}(G\theta')) \leq_\Omega \mathcal{I}_i(\mathfrak{U}(P[s_1/x_1, \ldots, s_m/x_m]), \mathfrak{U}(A[s_1/x_1, \ldots, s_m/x_m])),$$

as desired.

For the resolution step involving $\langle i; P \mid \exists y.G\rangle \overset{\exists}{\rightsquigarrow} \langle i; P \mid G[v/y]\rangle$, where v is a logic variable of label i, we assume

$$\mathcal{I}_i(\mathfrak{U}(P[s_0/v, s_1/x_1, \ldots, s_m/x_m]), \mathfrak{U}(G[v/y][s_0/v, s_1/x_1, \ldots, s_m/x_m])) = \top_\Omega.$$

Then $\top_\Omega \leq \mathcal{I}_i(\mathfrak{U}(P[s_1/x_1, \ldots, s_m/x_m]), \mathfrak{U}((\exists y.G)[s_1/x_1, \ldots, s_m/x_m]))$ by (16).

For the resolution step involving $\langle i; P \mid \forall y_\alpha.G\rangle \overset{\forall}{\rightsquigarrow} \langle i+1; P \mid G[c/y]\rangle$, where c is a constant of index $i+1$, we assume $\mathcal{I}_{i+1}(\mathfrak{U}(P), \mathfrak{U}(G[c/y])) = \top_\Omega$ under $[\overline{s}/\overline{x}]$. Then (17) yields the desired result

$$\top_\Omega = \bigvee_{d \in \mathsf{D}_{i+1,\alpha}} \mathcal{I}_{i+1}(\mathfrak{U}(p), \mathfrak{U}(\mathsf{App}(f, d))) = \mathcal{I}_i(\mathfrak{U}(P[\overline{s}/\overline{x}]), \mathfrak{U}((\forall y.G)[\overline{s}/\overline{x}])).$$

□

Corollary 4.9 (Soundness) *If there is an SLD proof of a closed goal G from a closed program P, then for all uniform algebras $\mathfrak{U}$, $\mathcal{I}(\mathfrak{U}(P), \mathfrak{U}(G)) = \top_\Omega$.*

We now establish completeness of resolution. Let Ω be $\{\top, \bot\}$.[2] Consider the uniform **term** algebra $\mathfrak{T} = \langle \mathsf{D}, \mathsf{App}, \mathsf{Const}, \mathsf{Inc}\rangle$ with D_α the set of principal normal forms of closed UCTT-terms of type α; let App and Const build the obvious terms and Inc be identity functions. Let $P \leq_{\mathsf{p}} P'$ if $[i; P] \subseteq [i; P']$.

For this particular situation, we define a **weak interpretation on atoms** to be an indexed family of functions $I = \{I_i : i \in \mathbb{N}\}$, where for each natural number i, $I_i : \mathsf{D}_{i,\mathsf{p}} \times \mathsf{D}_{i,\mathsf{a}} \to \Omega$ satisfying $p \leq_{\mathsf{p}} q$, $i \leq j$ implies $I_i(p, a) \leq_\Omega I_j(q, a)$. Let its extension to all goals, $I_i : \mathsf{D}_{i,\mathsf{p}} \times \mathsf{D}_{i,\mathsf{g}} \to \Omega$, $i \in \mathbb{N}$ satisfy only (12), (13), (14), (15), (16) and (17), but not necessarily the properties (18) and (19) enjoyed by interpretations.

Lemma 4.10 *Weak interpretations form a complete lattice with the partial order*

$$I \sqsubseteq I' := (\forall i \in \mathbb{N}) I_i \sqsubseteq_i I'_i$$

where the order $\sqsubseteq_i$ on the interpretation fibers I_i is defined pointwise.

[2] The reader should note that this does not make our semantics classical. See Section 4.7 for details.

Since the the order (on the I_i and the I) is built up pointwise, it is easily seen to inherit the structure of a complete lattice from Ω.

We define a "T_P" operator on these weak interpretations in the usual way: $T(I)_i(P,A) = \top_\Omega$ if $A \in [i;P]_\mathcal{V}$ or there is a clause $G \supset A$ in $[i;P]_\mathcal{V}$ such that $I_i(P,G)$.

A more general definition, for arbitrary complete lattices Ω, which reduces to the preceding one for the two-valued $\Omega = \{\top, \bot\}$ is:

$$T(I)_i(P,A) = \bigvee_{G \supset A \in [i;P]_\mathcal{V}} I_i(P,G). \tag{20}$$

provided we adopt the convention that $\top \supset A \in [i;P]$ whenever the atom A is in $[i;P]$.

Since T is defined in terms of a supremum, and the sup operation $\bigsqcup$ on weak interpretations is defined pointwise, proof of the continuity of T on weak interpretations boils down to a straightforward commutation of Ω-suprema. By the Tarski-Knaster theorem, we obtain the fact that T has a least fixed point, and that it is the supremum of the chain $\mathcal{I}_\bot, T(\mathcal{I}_\bot), T^2(\mathcal{I}_\bot), \ldots$.

Lemma 4.11 *The fixed point $\mathcal{I} = T^\infty(I_\bot)$ is a genuine interpretation, not just a weak one. That is to say, in addition to satisfying (12), (13), (14), (15), (16) and (17) of definition (4.5) it also satisfies properties (18) and (19):*

$$\begin{array}{rcl} p \leq_\mathsf{p} p' & \textit{implies} & \mathcal{I}_i(p,g) \leq_\Omega \mathcal{I}_i(p',g) \\ (g \supset_\mathsf{pgr} r) \leq_\mathsf{p} p & \textit{implies} & \mathcal{I}_i(p,g) \leq_\Omega \mathcal{I}_i(p, \mathsf{Inc}_\mathsf{ga}(\mathsf{Inc}_\mathsf{ar}(r))). \end{array}$$

The proof is straightforward. We will sketch the argument for the second property. Suppose P is a program, G a goal, and R a rigid atom, both closed, and that $G \supset R \leq_\mathsf{p} P$, that is to say, for all i $[i; G \supset R] \subseteq [i;P]$. In particular, $G \supset R \in [i;P]$. By definition (20) of T, for any natural number n (and any index i)

$$(T^{n+1}(I_\bot))_i(P,R) = \bigvee_{B \supset R \in [i;P]_\mathcal{V}} (T^n(I_\bot))_i(P,B).$$

Since $G \supset R$ lies in $[i;P]_\mathcal{V}$, $T^n(I_\bot))_i(P,G)$ is one of the terms in the supremum

$$\bigvee_{B \supset R \in [i;P]_\mathcal{V}} T^n(I_\bot))_i(P,B).$$

Therefore, $T^n(I_\bot)_i(P,G) \leq T^{n+1}(I_\bot)_i(P,R)$. Taking suprema, we obtain $(T^\infty(I_\bot))_i(P,G) \leq (T^\infty(I_\bot))_i(P,R)$ as we wanted to show.

We use the fixed point of T to establish completeness.

Theorem 4.12 *If $\mathcal{I}_i(\mathfrak{U}(P), \mathfrak{U}(G)) = \top_\Omega$ in all uniform algebras $\mathfrak{U}$, then there is an SLD derivation $\langle i; P \mid G\rangle \leadsto \cdots \leadsto$ NULL whose substitutions are the identity on the free variables of P and G.*

Proof. We consider the uniform algebra discussed above, and drop all reference to $\mathfrak{U}$, which is the identity here. In the usual fashion we can show $T^k(I_\bot)(i,P,G) = \top_\Omega$ for some k. The proof is by induction on k.

Base case: For $k = 1$, the proof is by induction on the structure of G: If the goal is atomic, A, then $T(I_\bot)(i,P,A) = \top_\Omega$ and $A \in [i;P]_\mathcal{V} \subset [i;P]_\mathcal{U}$. The desired SLD derivation $\langle i; P \mid A\rangle \leadsto$ *NULL* uses the *identity* rule. If the goal is $\top$, the associated NULL vector $\langle i; P \mid \top\rangle$ constitutes an SLD proof.

If the goal is of the form $G_1 \vee G_2$, then we have, say, $T(I_\bot)(i,P,G_1) = \top_\Omega$. By the induction hypothesis, we have a derivation $\langle i; P \mid G_1\rangle \leadsto \cdots \leadsto$ NULL satisfying the condition on substitutions, which by the *or* rule extends to $\langle i; P \mid G_1 \vee G_2\rangle \leadsto \langle i; P \mid G_1\rangle \leadsto \cdots \leadsto$ NULL.

If the goal is $G_1 \wedge G_2$, we assume $\langle i; P \mid G_1\rangle \rightsquigarrow \cdots \rightsquigarrow$ NULL, where the free variables of P and G_2 do not figure in the substitutions of the derivation, and $\langle i; P \mid G_2\rangle \rightsquigarrow \cdots \rightsquigarrow$ NULL, under the analogous condition. We may interleave these derivations to obtain $\langle i; P \mid G_1\rangle,\ \langle i; P \mid G_2\rangle \rightsquigarrow \cdots \rightsquigarrow$ NULL. By the *and* rule,

$$\langle i; P \mid G_1 \wedge G_2\rangle \rightsquigarrow \langle i; P \mid G_1\rangle,\ \langle i; P \mid G_2\rangle \rightsquigarrow \cdots \rightsquigarrow \text{NULL},$$

as desired.

In the case $T(I_\perp)(i, P, D \supset G') = \top_\Omega$, we know $T(I_\perp)(i, P \cup D, G') = \top_\Omega$ and so by the induction hypothesis, we have a derivation $\langle i; P \cup D \mid G'\rangle \rightsquigarrow \cdots \rightsquigarrow$ NULL satisfying the condition on substitutions. By the *augment* rule, we then have the desired SLD resolution $\langle i; P \mid D \supset G'\rangle \rightsquigarrow \langle i; P \cup D \mid G'\rangle \rightsquigarrow \cdots \rightsquigarrow$ NULL.

For a goal of the form $\exists x.G'$, $T(I_\perp)(i+1, P, G'[t/x]) = \top_\Omega$ and by the induction hypothesis the derivation $\langle i; P \mid G'[t/x]\rangle \rightsquigarrow \cdots \rightsquigarrow$ NULL for some term t over $\mathcal{V}_i$ satisfies the condition on substitutions. *I.e.*, $\langle i; P[t/v] \mid G'[v/x][t/v]\rangle \rightsquigarrow \cdots \rightsquigarrow$ NULL for a fresh variable v of index i. Since the substitution $[t/v]$ is legal, by a lemma, $\langle i; P \mid G'[v/x]\rangle \rightsquigarrow \cdots \rightsquigarrow$ NULL holds. This new derivation can be extended to $\langle i; P \mid \exists x.G'\rangle \overset{\exists}{\rightsquigarrow} \langle i; P \mid G'[v/x]\rangle \rightsquigarrow \cdots \rightsquigarrow$ NULL and this extended resolution satisfies the condition on substitutions.

For a goal of the form $\forall x.G'$, we know that $T(I_\perp)(i+1, P, G'[c/x]) = \top_\Omega$ for $c \in \mathcal{V}_{i+1} \setminus \mathcal{V}_i$. By the induction hypothesis, $\langle i+1; P \mid G'[c/x]\rangle \rightsquigarrow \cdots \rightsquigarrow$ NULL satisfies the condition on substitutions. Since $c \in \mathcal{U}_{i+1} \setminus \mathcal{U}_i$, this derivation can be extended with the generic rule to $\langle i; P \mid \forall x.G'\rangle \overset{\forall}{\rightsquigarrow} \langle i+1; P \mid G[c/x]\rangle \rightsquigarrow \cdots \rightsquigarrow$ NULL.

Induction cases: Again, the proof for $k > 1$ is by induction on the structure of G. For an atomic goal A, according to either $A \in [i; P]_{\mathcal{V}}$ or $T^{k-1}(I_\perp)(i, P, G') = \top_\Omega$ and $G' \supset A \in [i; P]_{\mathcal{V}}$. The former case was handled above. In the latter, we have an SLD derivation $\langle i; P \mid G'\rangle \rightsquigarrow \cdots \rightsquigarrow$ NULL by the induction hypothesis on k. Since $G' \supset A \in [i; P]_{\mathcal{U}}$, we apply the backchain rule with an empty substitution, extending the given derivation to $\langle i; P \mid A\rangle \rightsquigarrow \langle i; P \mid G'\rangle \rightsquigarrow \cdots \rightsquigarrow$ NULL.

The non-atomic cases are analogous to those for $n = 1$. □

Corollary 4.13 (Completeness) *Let Ω be $\{\top, \perp\}$. If $\mathcal{I}_i(\mathfrak{U}(P), \mathfrak{U}(G)) = \top_\Omega$ in all uniform algebras $\mathfrak{U}$, then there is an SLD proof of G from P.*

It should be noted that by a straightforward adaptation of the proof of the corresponding theorem in Miller et. al. [17], an SLD proof of G from P exists iff a proof of $P \twoheadrightarrow G$ exists in ICTT. Thus, $\mathcal{I}_i(\mathfrak{U}(P), \mathfrak{U}(G)) = \top_\Omega$ in all uniform algebras $\mathfrak{U}$ if and only if $[\![P \twoheadrightarrow G]\!] = \top$ in all ICTT uniform algebras $\mathfrak{C}$.

4.7 Remarks on Uniform Algebras

The use of $\Omega = \{\top, \perp\}$ corresponds to the two-valued forcing relationship in our first-order models and so the model used in the completeness theorem is just the higher-order analogue of the Miller model. In fact, if we let the pairs (i, p) for $p \in \mathsf{D}_{i,p}$ be the nodes of a Miller-style model with the order inherited from the preorders for $\mathsf{D}_{i,p}$ and the inclusion Inc_{ji}, we obtain an elementarily equivalent semantics by setting $i, P \Vdash G$ when $\mathcal{I}_i(\mathfrak{U}(P), \mathfrak{U}(G)) = \top_\Omega$. The case $\Omega = \{\top, \perp\}$ corresponds to a Kripke model with a classical metatheory, that is, a decidable forcing relation. It suffices to establish completeness.

In future work, we hope to explore the significance of varying Ω. Surely this wider class of models includes Miller-style models with a constructive metatheory, and it may capture other notions of operational semantics for logic programming. For example, it is not hard to show that c-semantics ([4, 7]) corresponds to an Ω given by the set of open substitutions.

References

[1] Peter Andrews. Resolution in type theory. *Journal of Symbolic Logic*, 36:414–432, 1971.

[2] Peter Andrews. General models and extensionality. *Journal of Symbolic Logic*, 37(2):395–397, 1972.

[3] Mino Bai and Harold Blair. General model theoretic semantics for higher-order horn logic programming. In *Logic Programming and Automated Reasoning, International Conference LPAR'92*, volume 624 of *LNCS*. Springer, 1992.

[4] A. Bossi, M. Gabrielli, G. Levi, and M. C. Meo. A compositional semantics for logic programs. *Theoretical Computer Science*, 122(1-2):3–47, 1994.

[5] Jawahar Chirimar. *Proof Theoretic Approach to Specification Languages*. PhD thesis, University of Pennsylvania, 1995.

[6] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[7] Marco Comini. *An abstract interpretation framework for Semantics and Diagnosis of logic programs*. PhD thesis, Dipartimento di Informatica, Universitá di Pisa, 1998.

[8] M. DeMarco. *Higher Order Logic Programming in the Theory of Types*. PhD thesis, Wesleyan University, 1999.

[9] M. DeMarco and J. Lipton. The semantics of and intuitionistic version of church's type theory and higher order hereditary harrop logic programming. To appear.

[10] Michael Fourman and Dana S. Scott. Sheaves and logic. In *Applications of Sheaves*, volume 753 of *Lecture Notes in Mathematics*, pages 302–401. Springer, 1979.

[11] Harvey Friedman. Equality between functionals. In R. Parikh, editor, *Logic Colloquium*, volume 453 of *Lecture Notes in Mathematics*, pages 22–37. Springer, 1975.

[12] J.Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge University Press, 1998.

[13] Leon Henkin. Completeness in the theory of types. *Journal of Symbolic Logic*, 15:81–91, 1950.

[14] Dale Miller. λ-prolog: an introduction to the language and its logic. manuscript available at `http://www.cse.psu.edu/ dale`.

[15] Dale Miller. A logical analysis of modules in logic programming. *Journal of Logic Programming*, pages 79–108, 1989.

[16] Dale Miller. A multiple-conclusion meta-logic. In *Proceedings of the 1994 Symposium on Logics in Computer Science*, pages 272–281, 1994.

[17] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 1990.

[18] John Mitchell. *Foundations for Programming Languages*. MIT Press, Cambridge, Massachusetts, 1996.

[19] John Mitchell and Eugenio Moggi. Kripke-style models for typed lambda calculus. *Annals of Pure and Applied Logic*, 51:99–124, 1991.

[20] G. Nadathur. *A Higher-Order Logic as the Basis for Logic Programming*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, 1987.

[21] A. S. Troelstra and D. van Dalen. *Constructivism in Mathematics: An Introduction*, volume 2. Elsevier Science Publishers, 1988.

[22] D. A. Wolfram. A semantics for λ-prolog. *Theoretical Computer Science*, 136:277–289, 1994.