Higher-Order Logic Programming Languages with Constraints: a Semantics

James Lipton and Susana Nieva

[1] Wesleyan University, USA, and visiting Professor
Univ. Politécnica de Madrid, Spain
jlipton@wesleyan.edu

[2] Dep. Sistemas Informáticos y Computación
Univ. Complutense de Madrid, Spain
nieva@sip.ucm.es

Abstract. A Kripke Semantics is defined for a higher-order logic programming language with constraints, based on Church's Theory of Types and a generic constraint formalism.
Our syntactic formal system, *hoHH(C)* (higher-order hereditary Harrop formulas with constraints), which extends λProlog's logic, is shown sound and complete.
A Kripke semantics for equational reasoning in the simply typed lambda-calculus (Kripke Lambda Models) was introduced by Mitchell and Moggi in 1990. Our model theory extends this semantics to include full impredicative higher-order intuitionistic logic, as well as the executable *hoHH* fragment with typed lambda-abstraction, implication and universal quantification in goals and constraints. This provides a Kripke semantics for the full higher-order hereditarily Harrop logic of λProlog as a special case (with the constraint system chosen to be β,η-conversion).

1 Introduction

Declarative programming languages have been developed with the aim of keeping code as close as possible to some notion of a specification, while at the same time having a reasonably efficient operational interpretation. This goal has usually been pursued by taking the syntax from some underlying formalism, which gives programs and inputs independent mathematical meaning, and then defining a mechanism that makes the code executable in a way consistent with that meaning. Semantics provides a framework in which the two readings can be understood and analyzed, and their compatibility verified.

The original Horn clause-with-resolution formulation of logic programming that underlies Prolog has been quite successful, but has required a significant control component outside the logic for efficiency, and includes controversial metaprogramming predicates.

Two basic lines of research have attempted to add to the expressive power of the original Horn clause core without compromising declarative transparency. One has been based on *expanding the logic* to include, for example, executable fragments of higher-order logic with lambda-conversion, higher-order unification, type theory, linear logic,

etc. [18], and the other on *adding constraints* [12, 22, 13] in a reasonably generic fashion. The latter addition can be thought of, declaratively, as transferring the logic from a term model to more complex data domains, giving enriched notions of computation, input, output and unification.

This paper provides semantic tools for modelling logic programming with intuitionistic higher-order logic, with implication and universal quantification in goals, types, λ-terms and constraints. We have chosen to work with one particular logic programming language, *hoHH($\mathcal{C}$)*, that combines the logic underlying λProlog, higher-order hereditarily Harrop formulas over an intuitionistic formulation of Church's Theory of Types [18], with Saraswat's constraint formalism [22], because it incorporates so many of logic programming extensions of interest and along with them, many of the problems that must be overcome in modelling similar languages.

The combination of these features creates a multiple challenge for the semantics: modelling higher-order intuitionistic formulas in an impredicative logic, giving meaning to λ-terms and types and relativizing interpretations to a foreign black-box constraint system.

Several problems must be solved here. To begin with, there is the *logical intensionality* problem: one must supply a denotation of λ-terms, including those of boolean type, while simultaneously giving them truth values. Logic programming in type theory requires a certain amount of noninterference between the two. In higher-order logic, and in particular in λProlog, predicates may appear as arguments to predicates, yet logically equivalent predicates *must not give rise inevitably to identical denotations*: if F_1 and F_2 are logically equivalent formulas, it need not follow that for any higher-order predicate p of the right type $p(F_1)$ and $p(F_2)$ are equivalent. Otherwise a goal $?-p(F_1)$ with a program $p(F_2)$ (which *fails* in λProlog) could not be handled just by unification over a constraint theory. It would call the entire proof search mechanism into play just to determine first if F_1 were equivalent to F_2.

Also *impredicativity* must be dealt with *a priori*. If X is of boolean type, an instance $G[t/X]$ of a higher-order formula such as $\exists XG$ may have *greater complexity* than the original formula (just consider $t = (\exists XG)$). Thus the usual inductive definability of truth must somehow be circumvented, either by inducting on something other than formulas (types in Henkin's completeness theorem [10], or other measures as in the second class of models defined in this paper) or by a mixed approach, making the definition of truth non-inductive as in our initial Kripke semantics, or as in [23, 1, 6].

In this paper we show *hoHH($\mathcal{C}$)* is sound and complete for our Kripke structures, extending earlier partial results [17, 7, 26], and including Kripke semantics for the full logic of λProlog as a special case, obtained by taking β,η-conversion for our constraint system.

2 Higher-order Intuitionistic Logic with Constraints

In this section we briefly recapitulate for the reader's convenience the main syntactic features of the *hoHH($\mathcal{C}$)* programming language, and its sequent calculus, treated in much greater detail in [15, 16].

2.1 Syntactic preliminaries

The formalization of higher-order logic used here is based on an intuitionistic reformulation [18] of Church's theory of types [4], a theory that builds higher-order logic on top of the simply typed λ-calculus. The existence of types facilitates the incorporation of a foreign constraint system. Logical formulas are terms of type o, and constraints are terms of a new base type γ. We are thus able to define mixed theories using a single formal mechanism.

We begin by defining what terms, types, formulas and constraints are in Church's Theory of Types. In the next section (2.2), on the syntax of *hoHH(C)*, we will restrict this class of formulas to a subset of legal *clauses* and *goals*.

The main components of Church's Type Theory are *types* and *terms*. The set Ty of types, with elements α, includes at least atomic types, called sorts, and functional types, $\alpha \to \alpha$. The set of sorts must contain at least the two special sorts o and γ. The functional type $\to$ associates to the right. We will also freely make use of the more compact *Church's type notation* $\beta\alpha$ for $\alpha \to \beta$, which, since given in reverse order, associates to the left.

Typed terms, denoted by t (or t^α when displaying their type is of interest) are obtained from a set V of typed variables, x^α, and a signature Σ consisting of a set of typed constant symbols, c^α, by the abstraction and type-compatible application operations of the λ-calculus: $t := x^\alpha \mid c^\alpha \mid (\lambda x^\alpha.\, t) \mid (t^{\alpha\to\beta}\, t^\alpha)$. We omit parentheses when they are not necessary, assuming that abstraction and application are right and left associative, respectively, and that application has smaller scope than abstraction.

Terms of type o are called *logic formulas*. Terms of type γ are called *constraint formulas*, and they are usually denoted by C.

We say that a term t is in λ-*normal* form $\Lambda(t)$ when it is in both β, η-normal form. By λ-equivalence we mean α, β, η-equivalence between terms, and we denote it $\equiv_\lambda$. Every term in our type theory (the simply typed λ-calculus) is equivalent to one in normal form [9, 11].

We use the notation $\mathsf{fv}(t)$ or $\mathsf{fv}(S)$ to denote the set of free variables in a term t or in a set S, respectively. We denote simultaneous substitution of terms t_i for every free occurrence of x_i in a term t by $t[t_1/x_1, \ldots, t_n/x_n]$, or simply $t[\overline{t}/\overline{x}]$, and we will assume that $t[\overline{t}/\overline{x}]$ is in fact $\Lambda(t[\overline{t}/\overline{x}])$.

The signature Σ is partitioned into *logical* and *nonlogical constants*. The logical constants are the symbols: $\top$, $\bot$, $\wedge$, $\vee$, $\Rightarrow$, $\exists$, $\forall$, of certain specific types. Since constraints and pure logical formulas are terms of different types, Σ must contain logical constants of different types to build constraints or logical formulas. For instance, $\exists^{(\alpha\to\gamma)\to\gamma}$, $\exists^{(\alpha\to o)\to o}$, or $\wedge^{o\to o\to o}$, $\wedge^{\gamma\to o\to o}$ are always elements of Σ. The non-logical constants are those defined by the user, including a symbol for equality: $\approx^{\alpha\to\alpha\to\gamma}$, for every type α.

We use infix notation for $\approx, \wedge, \vee, \Rightarrow$, and, following Church, we abbreviate $\exists(\lambda x.F)$, $\forall(\lambda x.F)$ by $\exists xF$ and $\forall xF$, respectively. We call a logic formula in normal form whose leftmost non-parenthesis symbol is either a nonlogical constant or a variable an *atomic formula*, *rigid* in the former case, and *flexible* in the latter. This leading symbol is called the *predicate symbol* or *predicate variable*, respectively, of the atomic formula in ques-

tion. We denote atomic formulas by A. A_r represents rigid atomic formulas. For predicate variables (variables of type o) we use capital letters X, Y.

For the rest of this paper we will take the signature Σ and the initial set of variables V to be fixed, with one exception. The set of variables will be extended in the proof of the completeness theorem.

2.2 The programming language *hoHH(C)*

In [18] Miller *et. al.* identified the so-called *uniformity* property as a fundamental requirement for a logic programming language. This property guarantees completeness of goal-oriented search for proofs with respect to the underlying logic of intuitionistic type theory. Our language extends the class of *higher-order Hereditary Harrop formulas* of λProlog to include constraints in such a way as to preserve uniformity, as it is shown in detail in [15].

The constraint system $\mathcal{C}$ The constraints we will consider here belong to a generic system $\mathcal{C}$ that is assumed to satisfy certain conditions. Following [22], we view a *constraint system* as a pair $\mathcal{C} = \langle \mathcal{L}_\mathcal{C}, \vdash_\mathcal{C} \rangle$, where $\mathcal{L}_\mathcal{C}$ is a set of λ-terms of type γ in normal form built up from Σ and V, and $\vdash_\mathcal{C}$ is a binary *entailment relation* between sets of constraints, Γ, and single constraints, C. $\mathcal{C}$ is required to satisfy:

- Every λ-term in normal form of type γ, built up using constraint predicate symbols (of type $\alpha \to \gamma$), the logical constants $\top^\gamma$, $\bot^\gamma$, $\exists^{(\alpha\to\gamma)\to\gamma}$ and optionally, other suitably typed logical constants (such as $\wedge^{\gamma\to\gamma\to\gamma}, \Rightarrow^{\gamma\to\gamma\to\gamma}, \forall^{(\alpha\to\gamma)\to\gamma}$) is in $\mathcal{L}_\mathcal{C}$.
- All equations $t_1 \approx t_2$ are in $\mathcal{L}_\mathcal{C}$.
- All the inference rules for equality and for those connectives included in $\mathcal{L}_\mathcal{C}$ that are valid in intuitionistic logic are valid inferences in $\vdash_\mathcal{C}$.
- *Compactness:* $\Gamma \vdash_\mathcal{C} C$ holds iff $\Gamma_0 \vdash_\mathcal{C} C$ for some finite $\Gamma_0 \subseteq \Gamma$.
- $\Gamma \vdash_\mathcal{C} C$ implies $\Gamma\sigma \vdash_\mathcal{C} C\sigma$ for every substitution σ.
- If $t_1 \equiv_\lambda t_2$, then $\vdash_\mathcal{C} t_1 \approx t_2$.
- The cut-rule is allowed in $\mathcal{C}$: if $\Gamma' \vdash_\mathcal{C} \Gamma$ and $\Gamma \vdash_\mathcal{C} C$, then $\Gamma' \vdash_\mathcal{C} C$.

An often used example is the constraint system $\mathcal{R}$ of Real-closed Fields: $\mathcal{L}_\mathcal{R}$ is a language with all classical logical connectives including negation, and $\Gamma \vdash_\mathcal{R} C$ holds iff $Ax_\mathcal{R} \cup \Gamma \vdash_\approx C$, where $Ax_\mathcal{R}$ is Tarski's axiomatization of the real numbers [24], and $\vdash_\approx$ is the entailment relation of classical logic with equality.

Now we spell out the syntax of our programming language. The atomic formulas of *hoHH(C)*, like those of λProlog's *hoHH* are limited to those formed by application of predicate symbols (symbols of type $\alpha \to o$) to *positive terms*, in accordance with the following definition.

Definition 1. *The set of* positive terms *consists of all the terms in λ-normal form built up from Σ and V, not containing constraint predicate symbols, nor the logical constants $\bot$ and $\Rightarrow$.*

A positive atomic formula *is an atomic logical formula containing only positive terms.*

We remind the reader that in higher-order logic, logical formulas can appear as subterms of atomic formulas, so this restriction is significant.

Given a generic constraint system $\mathcal{C}$ satisfying the requirements listed above, the syntax of the constraint-enriched formal system *hoHH($\mathcal{C}$)* consists of the following fragment of higher-order logic.

Definition 2. *The set of* definite clauses*, with elements denoted by D, and the set of* goals*, with elements denoted by G, are sets of formulas, in λ-normal form, defined by the following syntactic rules:*

$D := A_r \mid D_1 \wedge D_2 \mid G \Rightarrow A_r \mid \forall x D$

$G := A \mid C \mid G_1 \wedge G_2 \mid G_1 \vee G_2 \mid D \Rightarrow G \mid C \Rightarrow G \mid \exists x G \mid \forall x G$

where A is a positive atomic formula, A_r a rigid positive atomic formula. Notice that $\top$ and $\bot$ are constraints, so they are goals.

A program Δ is a finite set of definite clauses (just called "clauses" for the rest of the paper).

Clauses are always terms of type o. Goals that are pure constraint formulas C (which may themselves contain connectives of type e.g. $\gamma \to \gamma \to \gamma$) have type γ, but compound goals built up from them using the definition just given, must be of type o. Thus, for instance, depending on the nature of G_1, G_2, a goal of the form $G_1 \wedge G_2$ might be built with $\wedge^{\gamma\to o\to o}$, $\wedge^{o\to\gamma\to o}$ $\wedge^{\gamma\to\gamma\to o}$ or $\wedge^{o\to o\to o}$. When type of the logical constants can be deduced from the context, the typing is not shown.

Example 1. Consider the instance *hoHH($\mathcal{R}$)*. The following program can be written.

$\Delta = \{\forall x\forall y(x^2 + y^2 \approx 2 \Rightarrow \mathit{circle}(x, y)), \forall x\forall y(x^2 + 6y^2 \approx 2 \Rightarrow \mathit{ellipse}(x, y)),$
$\forall X(X \approx circle \vee X \approx ellipse \Rightarrow figure(X))\}.$

And the goal

$G \equiv \exists X_1 \exists X_2 (figure(X_1) \wedge figure(X_2) \wedge \neg(X_1 \approx X_2) \wedge X_1(x, y) \wedge X_2(x, y))$.

However, the formula $\forall x(\exists y((y^2 \approx x \vee ellipse(x, y)) \Rightarrow circle(x, y)))$ is not a logical formula of the language *hoHH($\mathcal{C}$)*, because it is not a clause, due to the existential quantifier, nor a goal, since the disjunction $y^2 \approx x \vee ellipse(x, y)$ is not allowed in the antecedent of a goal.

The *elaboration* of a program Δ is a mapping from programs to sets of implicative clauses. It is the set $elab(\Delta) = \bigcup_{D\in\Delta} elab(D)$, where $elab(D)$ is defined by the following rules:

$elab(A_r) = \{\top \Rightarrow A_r\}$, $\quad elab(D_1 \wedge D_2) = elab(D_1) \cup elab(D_2)$,

$elab(G \Rightarrow A_r) = \{G \Rightarrow A_r\}$, $\quad elab(\forall x D) = \{\forall x D' \mid D' \in elab(D)\}$.

We will assume that any goal, constraint or element of any $elab(\Delta)$ is in normal form.

The Proof Rules We now give the underlying sequent calculus, $ho\mathcal{UC}$, that makes the collection of $program; constraint \vdash goal$ triples in *hoHH($\mathcal{C}$)* a (nondeterministic) logic programming language. Possible interpreter design along the lines, say of [21] are not discussed here. This proof system combines traditional inference rules with the entailment relation of a generic constraint system $\mathcal{C}$.

Sequents have finite sets of programs and constraints on the left and single goals on the right. $\Delta; \Gamma \vdash_{ho\mathcal{UC}} G$ means the sequent $\Delta; \Gamma \vdash G$ is derivable in $ho\mathcal{UC}$. When either or Δ or Γ is infinite, we mean the sequent $\Delta'; \Gamma' \vdash G$ is derivable in $ho\mathcal{UC}$ for some finite subsets $\Delta' \subseteq \Delta$ and $\Gamma' \subseteq \Gamma$.

C is called an *answer constraint for G from Δ* when $\Delta; C \vdash_{ho\mathcal{UC}} G$. For instance, in Example (1), $x^2 \approx 2 \wedge y \approx 0$ is an answer constraint. As with many constraint formalisms, constraints built up progressively on the left during a bottom-up construction of a proof of a given goal constitute the *output* of this programming language, considerably extending the expressive power of conventional logic programming, where outputs are restricted to equations of the form $variable = term$.

The set of rules of this proof system appears in Figure 1.

$$\frac{\Gamma \vdash_{\mathcal{C}} C}{\Delta; \Gamma \vdash C}\ (C_R) \qquad \frac{\Delta; \Gamma \vdash \exists \overline{x}((A'_r \approx A_r) \wedge G')}{\Delta; \Gamma \vdash A_r}\ (Clause)\ (*), \text{where}$$

$$\forall \overline{x}(G' \Rightarrow A'_r) \text{ is } \alpha\text{-equivalent to a formula of } elab(\Delta)$$

$$\frac{\Delta; \Gamma \vdash F \quad \Gamma \vdash_{\mathcal{C}} X \approx t}{\Delta; \Gamma \vdash X t_1 \ldots t_n}(Flex),\ F \equiv \Lambda((X\, t_1 \ldots t_n)[t/X]), \mathsf{fv}(t) \subseteq \mathsf{fv}(\overline{t}), t \text{ positive}$$

$$\frac{\Delta; \Gamma \vdash G_i}{\Delta; \Gamma \vdash G_1 \vee G_2}\ (\vee_R)\ (i = 1, 2) \qquad \frac{\Delta; \Gamma \vdash G_1 \quad \Delta; \Gamma \vdash G_2}{\Delta; \Gamma \vdash G_1 \wedge G_2}\ (\wedge_R)$$

$$\frac{\Delta, D; \Gamma \vdash G}{\Delta; \Gamma \vdash D \Rightarrow G}\ (\Rightarrow_R) \qquad \frac{\Delta; \Gamma, C \vdash G}{\Delta; \Gamma \vdash C \Rightarrow G}\ (\Rightarrow_{C_R})$$

$$\frac{\Delta; \Gamma, C \vdash G[y/x] \quad \Gamma \vdash_{\mathcal{C}} \exists y C}{\Delta; \Gamma \vdash \exists x G}\ (\exists_R)(*), \qquad \frac{\Delta; \Gamma \vdash G[y/x]}{\Delta; \Gamma \vdash \forall x G}\ (\forall_R)(*)$$

(*) $\overline{x}$, y do not appear free in the sequent of the conclusion.

Fig. 1. $ho\mathcal{UC}$ Sequent Rules

In all rules except (C_R), the principal formula is not a constraint. This means that any connective introduced by the rules must have target type o (and not γ).

This calculus is similar to those defined for higher-order formulas in the literature (see e.g. [18]), but the presence of constraints induces some modifications. The (λ) rule, that transforms formulas by λ-conversion, is not needed in $ho\mathcal{UC}$ because every formula in a sequent of a proof is in λ-normal form. Note that, save for the $(Flex)$ rule *no substitution of a compound term for a variable is made during the application of the rules*, illustrating what might be viewed as a fundamental slogan for this calculus: *constraints are generalized terms*. The burden is shifted from terms to potentially more expressive predicates in the constraint system. This is perhaps best exemplified in the $(\exists_R)$ rule, discussed more at length in [15, 16], by use of which substitutions can be simulated by constraints C on the left which are inhabited, i.e. those for which a proof of $\exists y C$ can be found. This may just mean replacing a substitution $[t/x]$ in conventional logic programming by an equality constraint of the form $x \approx t$. However, it is considerably more powerful, because constraints allow for a more general description of a potential witness for an existentially quantified formula where a specific term might not

exist. For example, in $\mathcal{R}$ the constraint $(x * x \approx 2)$ may represent $\sqrt{2}$, which is not a legal term.

The use of constraints also broadens the scope of backchaining by means of the combination of the $(Clause)$ and $(\exists_R)$ rules. Inspection of the $(Clause)$ rule shows we are not required, as in conventional logic programming, to unify the head of the selected clause with the atomic goal to be solved, but rather to solve a new existentially quantified goal that, by the use just discussed of the $(\exists_R)$ rule, will result in a search for a *constraint* that implies equality of the atomic goal and the clause head. The $(Clause)$ rule is not applied to flexible atoms, instead flexible atoms are managed with the $(Flex)$ rule, which permits non-atomic instantiation of the predicate variable.

Proposition 1. *The following rules are admissible in $ho\mathcal{UC}$ (if the premises are derivable, the conclusion is derivable).*

$$\frac{\Delta; \Gamma, C[y/x] \vdash G}{\Delta; \Gamma, \exists x C \vdash G}\ (\exists_{\mathcal{C}_L})\,(*) \qquad \frac{\Delta; \Gamma, C \vdash G \quad \Gamma \vdash_{\mathcal{C}} C}{\Delta; \Gamma \vdash G}\ (cut_{\mathcal{C}}) \qquad \frac{\Delta; \Gamma \vdash G[t/x]}{\Delta; \Gamma, y \approx t \vdash G[y/x]}\ (Subst)\,(*)$$

where the condition $(*)$ means $y \notin \mathsf{fv}(\Delta, \Gamma, G, \exists x C, t)$, t positive.

Proof. By the induction on the length of the derivation, analyzing cases according to the last rule applied, and using the properties of the relation $\vdash_{\mathcal{C}}$. □

In fact, as shown in [15], the proof system $ho\mathcal{UC}$ is equivalent, with antecedents and consequents restricted to the executable *hoHH(C)* fragment, to the extended calculus $ho\mathcal{IC}$ (higher-order Intuitionistic Calculus over $\mathcal{C}$) which includes the full intuitionistic theory of types with constraints. $ho\mathcal{IC}$ therefore manipulates not necessarily positive terms, has rules introducing connectives in the left, and a simple axiom for dealing with atoms, instead of $(Clause)$. That equivalence means that, for any program Δ, for any set of constraints Γ, and for any goal G: $\Delta; \Gamma \vdash_{ho\mathcal{IC}} G \iff \Delta; \Gamma \vdash_{ho\mathcal{UC}} G$. Therefore *hoHH(C)* satisfies the so-called uniformity property and can be considered as an abstract logic programming language in the sense defined in [18]. In practical terms this means that a search for a proof restricted to an operational interpretation of the connectives does not sacrifice any theorems.

Positive-atomic generated formulas These formulas constitute a subset of formulas in Church's theory of types that plays an special role in the definition of the Kripke semantics for our logic programming language. For these formulas (that include any *hoHH(C)* formula) a well-ordering can be defined which allows an induction argument in the proof of completeness of $ho\mathcal{UC}$.

Definition 3. *A formula in Church's theory of types is called* positive-atomic generated *or just* PA-generated*, if it is built up using logical constants from positive atomic formulas and constraints, and it is in normal form.*

Definition 4. *Let F be a PA-generated formula. We define the* non-positive depth *of F, $\delta(F)$, to be the length of the longest path from the root node of the parse tree of F to any occurrence of implication. Inductively:*

If F is positive or a constraint then $\delta(F) = 0$, otherwise
$\delta(F_1 \Diamond F_2) = 1 + max(\delta(F_1), \delta(F_2))$, where $\Diamond \in \{\wedge, \vee, \Rightarrow\}$,
$\delta(QxF) = 1 + \delta(F)$, where Q is $\forall$ or $\exists$.

This measure induces a well-founded order on the set of PA-generated logical formulas.

Lemma 1. *For any PA-generated formula F and any positive term t, $\delta(F[t/x]) = \delta(F)$. If $F_1 \Diamond F_2$, QxF are non-positive PA-generated logical formulas, then:*

i) $\delta(F_i) < \delta(F_1 \Diamond F_2)$*, for* $i = 1, 2$.
ii) $\delta(F[t/x]) < \delta(QxF)$ *for any positive term* t.

We finish this section with an example that illustrates some of the expressive power of *hoHH(C)*, when it is used to formalize inductive inference. Critical use is made of the availability of universal quantification in goals to specify induction conclusions and of nested implication in Hereditarily Harrop clauses to capture induction hypotheses. In addition, the presence of (arithmetic) constraints reduces the difficulty of many induction proofs, transferring some of the burden of proof to the constraint solver.

Example 2. Consider the instance *hoHH*($\mathcal{N}$), i.e. using the equational theory of the natural numbers as our constraint system.
The predicate *even* can be defined by the program clauses below, where the operator $+$ is managed by the constraint system.

$even(0), \forall\, x((even(x+2) \vee (x \geq 2 \wedge even(x-2))) \Rightarrow even(x))$.

In a proof of the property $\forall x \forall y(\mathit{even}(x) \wedge \mathit{even}(y) \Rightarrow \mathit{even}(x+y))$, the induction step corresponds to the resolution of the goal:

$$\forall x(\forall y(\mathit{even}(x) \wedge \mathit{even}(y) \Rightarrow \mathit{even}(x+y)) \Rightarrow \forall y(\mathit{even}(x+2) \wedge \mathit{even}(y) \Rightarrow \mathit{even}((x+2)+y))).$$

Applying the $(\Rightarrow_R)$ rule in reverse (i.e. the so-called *augment* rule of λProlog) the clause $D \equiv \forall y(\mathit{even}(x) \wedge \mathit{even}(y) \Rightarrow \mathit{even}(x+y))$, corresponding to the induction hypothesis, is added to the program as a local clause. Then it will be used during subsequent deduction steps, in particular in the proof of the subgoal $\mathit{even}((x+2)+y)$.

An interesting feature of such a deduction using a mix of constraints and logic is that since $+$ is a constraint operator, the search tree will be considerably pruned. For instance, during the proof, it is the constraint solver that checks the satisfiability of certain constraints such as $\forall x \forall y \exists x_1 (x + x_1 \approx (x+2) + y)$. In addition, the usual search problem of the choice of the variable on which induction is done is irrelevant here. The proof is also successful if y is chosen instead of x, because in this case, the constraint solver will deal with $\forall y \forall x \exists x_1 (x_1 + y \approx x + (y+2))$, in the same way as before.

3 Higher-Order Kripke Semantics

We first fix conventions and notation for the elementary model theory of the simply typed λ-calculus and the notion of an applicative structure indexed over a partially ordered set. Next we will define Kripke models for the full underlying logic (Church's Intuitionistic Theory of Types) without constraints that requires indexing models of the

λ-calculus as well. Then our Kripke models are modified to deal with the the fragment *hoHH(C)*. Constraints and formulas including constraints must be interpreted and additional conditions must be imposed. The need for Kripke semantics arises from the existence of intuitionistic connectives in our logic.

3.1 Semantic preliminaries

We start by recalling the definition of a model of the typed λ-calculus.

When considering indexed families $S = \{S_k\}, T = \{T_k\}$ of sets, we will say $f : S \to T$ is an *indexed function* if in fact f itself is a family of functions, indexed over the same set as S, T which *respects the indexed structure*, that is to say $f = \{f_k : S_k \to T_k\}$.

Definition 5. *A* Typed Applicative Structure (TAS)*,* $\mathsf{D} = \langle D, \mathsf{App}, \mathsf{Const}\rangle$*, is given by:*

- *a type-indexed family of sets* $D = \{D^\alpha | \alpha \in \mathsf{Ty}\}$*, each member of which,* D^α*, is called the* carrier *for the type* α*,*
- *a family of functions* $\mathsf{App} = \{\mathsf{App}^{\alpha\beta} : D^{\beta\alpha} \times D^\alpha \to D^\beta | \alpha, \beta \in \mathsf{Ty}\}$*, and a type preserving indexed family of assignment functions* $\mathsf{Const} = \{\mathsf{Const}^\alpha : \Sigma^\alpha \to D^\alpha | \alpha \in \mathsf{Ty}\}$*, where* $\Sigma^\alpha \subseteq \Sigma$ *is the set of constants of type* α*.*

Definition 6. *Let* D *be a TAS. A* D-environment η *is a function from the set of variables into* D *which respects types.*

Definition 7. *Given a typed applicative structure* $\mathsf{D} = \langle D, \mathsf{App}, \mathsf{Const}\rangle$*, a* D-environmental model $[\![\ ]\!]$ *consists of an indexed family* $\{[\![\ ]\!]_\eta | \eta$ *a* D*-environment*$\}$ *of* total *functions from the terms into* D*, respecting types, for which the following hold, for any* D*-environment* η*:*

$$
\begin{array}{lcl}
[\![c]\!]_\eta & = & \mathsf{Const}(c), \quad \textit{for constants } c, \\
[\![x]\!]_\eta & = & \eta(x), \quad \textit{for variables } x, \\
[\![(t_1 t_2)]\!]_\eta & = & \mathsf{App}([\![t_1]\!]_\eta, [\![t_2]\!]_\eta), \\
[\![\lambda x^\alpha . t^\beta]\!]_\eta & = & d', \textit{ where } d' \in D^{\beta\alpha}, \ d' \textit{ is the } \text{unique } \textit{element such that for any } d \in D^\alpha, \\
 & & \mathsf{App}(d', d) = [\![t^\beta]\!]_{\eta[x:=d]}, \textit{ where } \eta[x := d] \textit{ is the } \mathsf{D}\textit{-environment} \\
 & & \textit{coinciding with } \eta, \textit{ save on } x, \textit{ where its value is } d.
\end{array}
$$

A model *is a triple* $\langle \mathsf{D}, [\![\ ]\!], \eta\rangle$*, where* D *is a TAS,* $[\![\ ]\!]$ *is a* D*-environmental model and* η *is a* D*-environment.*

Note that existence and uniqueness of the d' denoting $\lambda x.t$ in environment η is imposed (following [19]) as part of the definition. The condition is quite strong: it ensures the substitution lemma (below and in [19]). It also guarantees uniqueness of an interpretation for a given environment, as is easily shown by induction on term structure. For this reason, when the existence of such a $[\![\ ]\!]_\eta$ is clear from context we will refer to the model together with its environment as the pair $\langle \mathsf{D}, \eta\rangle$.

3.2 Kripke Models for Church's Intuitionistic Higher Order Logic

One of the most widely used semantics for intuitionistic logic was introduced by Kripke (1963). In Tarski models for classical logic, one must supply a domain and interpretations for function, relation and constant symbols. Kripke models, however consist of a

partially ordered collection of such domains, together with certain compatibility conditions between them. Perhaps the best way to visualize such a semantics is to think of a Kripke model as a function defined on a poset $(W, \leq)$, associating with each world $w \in W$ a domain D_w together with interpretations of the language.

Definition 8. *Let $(W, \leq)$ be a partially ordered set. A* $(W, \leq)$-indexed typed applicative structure *is a family $\mathcal{D} = \{\mathsf{D}_w | w \in W\}$, where for each $w \in W$, $\mathsf{D}_w = \langle D_w, \mathsf{App}_w, \mathsf{Const}_w \rangle$ is a typed applicative structure. For each $w \leq w' \in W$ the following conditions must be satisfied:*

- *Monotonicity: $D_w \subseteq D_{w'}$.*
- *$\mathsf{App}_w(f, d) = \mathsf{App}_{w'}(f, d)$, for any pair (f, d) (of the corresponding type) in D_w.*
- *$\mathsf{Const}_w(c) = \mathsf{Const}_{w'}(c)$, for any $c \in \Sigma$.*

Now we define the so-called forcing relation, $\Vdash$ between members w of W and certain members of D^o_w. We may think of $\Vdash$ as a partial function mapping such pairs, when defined, to $true$ or $false$. Note that, unlike conventional Kripke models, forcing is defined *entirely within the semantics*, that is to say as a relation between worlds and *denotations* of formulas, rather than syntactic formulas themselves. Since logical formulas are terms in higher-order logic, we must supply both a denotation and a truth value for formulas of type o. We are thus able to deal with a problem mentioned in the introduction, namely to allow formulas with the same truth values in all models to have different denotations.

Definition 9. *A* Kripke applicative structure *for Church's intuitionistic theory of types is a quadruple $\mathcal{K} = \langle W, \leq, \mathcal{D}, \Vdash \rangle$, where:*

$(W, \leq)$ is a poset.
$\mathcal{D} = \{\mathsf{D}_w | w \in W\}$ is a $(W, \leq)$-indexed typed applicative structure.
$\Vdash$ is a binary forcing *relation between worlds $w \in W$ and logical elements d in D^o_w (written $w \Vdash d$), satisfying:*

- *The* monotonicity requirement*, if $d \in D^o_w$ and $w \Vdash d$ then for any $w' \in W$ with $w' \geq w$ we have $w' \Vdash d$.*
- *The* logical conditions*, where $d_1 \cdot d_2$ abbreviates $\mathsf{App}_w(d_1, d_2)$, and the underlined symbols $\underline{c}$ denote the interpreted logical constants $\mathsf{Const}_w(c)$, for the appropriate world w:*

  1. $w \Vdash \underline{\top}$ *always,* 2. $w \Vdash \underline{\bot}$ *never,*
  3. $w \Vdash \underline{\wedge} \cdot d_1 \cdot d_2$ *iff* $w \Vdash d_1$ *and* $w \Vdash d_2$,
  4. $w \Vdash \underline{\vee} \cdot d_1 \cdot d_2$ *iff* $w \Vdash d_1$ *or* $w \Vdash d_2$,
  5. $w \Vdash \underline{\Rightarrow} \cdot d_1 \cdot d_2$ *iff for any* $w' \geq w$ *if* $w' \Vdash d_1$, *then* $w' \Vdash d_2$,
  6. $w \Vdash \underline{\exists} \cdot f^{o\alpha}$ *iff for some* $d \in D^\alpha_w$, $w \Vdash f \cdot d$,
  7. $w \Vdash \underline{\forall} \cdot f^{o\alpha}$ *iff for every* $w' \geq w$ *and* $d \in D^\alpha_{w'}$, $w' \Vdash f \cdot d$.

Several features of this definition are different from its first order counterpart. Firstly, the forcing relation (viewed as a truth-valued function) is *partial*: $w \Vdash d$ need not be defined for all members of D^o_w. In particular, note that there is no atomic case,

since the individuals on the right of the forcing relation are not syntactic formulas, but rather denotations in the carrier of type o. Because of the impredicativity of higher-order logic, a Kripke applicative structure is not necessarily uniquely determined by an atomic assignment of truth at each world (taking *atoms* to mean denotations in D^o_w of atomic formulas). Also monotonicity of forcing must be imposed by definition on all formulas at once.

Definition 10. *Let $\mathcal{K}$ be a Kripke applicative structure. A $\mathcal{K}$-*environment *η is a family $\{\eta_w | w \in W\}$ of D_w-environments satisfying the following coherence property, for each variable x and each pair $w, w' \in W$ with $w \leq w'$: $\eta_w(x) = \eta_{w'}(x)$.*

We can now extend the notion of environmental model to Kripke applicative structures along the lines of Definition 7.

Definition 11. *Given a Kripke applicative structure $\mathcal{K} = \langle W, \leq, \mathcal{D}, \Vdash \rangle$, a* Kripke environmental model *for $\mathcal{K}$, or a $\mathcal{K}$-*interpretation *is an indexed family $\{ [\![\ ]\!]_\eta |\ \eta$ a $\mathcal{K}$-environment$\}$ where for each world $w \in W$, $[\![\ ]\!]_{\eta_w}$ is a* total *function from the set of terms into D_w, respecting types, and which, for each η_w, satisfies the conditions of Definition 7. For instance, $[\![x]\!]_{\eta_w} = \eta_w(x)$, and $[\![(t_1 t_2)]\!]_{\eta_w} = \mathsf{App}_w([\![t_1]\!]_{\eta_w}, [\![t_2]\!]_{\eta_w})$.*

Given a Kripke environmental model for $\mathcal{K}$ we will define the *$\mathcal{K}$-interpretation of a term t* over a $\mathcal{K}$-environment η at the world w to be $[\![t]\!]_{\eta_w}$.

Putting the whole package together, we can define our Kripke semantics.

Definition 12. *A* Kripke model *$\langle \mathcal{K}, [\![\ ]\!], \eta \rangle$ for Church's intuitionistic theory of types is given by a Kripke applicative structure $\mathcal{K}$, a Kripke environmental model $[\![\ ]\!]$ for $\mathcal{K}$ and a $\mathcal{K}$-environment η. Furthermore, for each formula F and world w, $w \Vdash [\![F]\!]_{\eta_w}$ is defined (true or false).*

As before, the notation can be simplified to $\langle \mathcal{K}, \eta \rangle$, as $[\![\ ]\!]_\eta$ is uniquely induced.

Now we are able to interpret the intuitionistic formulation of Church's logic into our semantics in a straightforward manner.

Definition 13. *Let $\mathcal{K} = \langle W, \leq, \mathcal{D}, \Vdash \rangle$, $\langle \mathcal{K}, [\![\ ]\!], \eta \rangle$ be a Kripke model, and let F be a logical formula, i.e. a term of type o. Then we say F is* forced *at w (or true at w) with environment η, and write $w \Vdash_\eta F$, whenever $w \Vdash [\![F]\!]_{\eta_w}$. If F is forced at every $w \in W$ in this environment, we write $\mathcal{K} \models_\eta F$. If either property holds in the presence of* all *$\mathcal{K}$-environments, then we write $w \Vdash F$ or, respectively, $\mathcal{K} \models F$ and say that $\mathcal{K}$ models or* satisfies *F (or that F is true in $\mathcal{K}$).*

The previous definitions are extended to finite sets of formulas S, in the natural way. For instance, $w \Vdash_\eta S$, means $w \Vdash_\eta F$ for all $F \in S$. In addition we will say that $\langle \mathcal{K}, \eta \rangle$ models or *satisfies* a sequent $\Delta; \Gamma \vdash G$ when for any world w, if $w \Vdash_\eta \Delta$ and $w \Vdash_\eta \Gamma$ then $w \Vdash_\eta G$.

The whole of intuitionistic type theory can be proved to be sound and complete with respect to this Kripke semantics. Since our interest is to adapt our semantics to the logic programming formalism *hoHH(C)* we will restrict attention to soundness and completeness for that case. First we will need to extend these definitions to include constraint systems, and modify the logical conditions.

3.3 Kripke Models for *hoHH(C)*

In our language, since we are thinking of the constraint system as a generic black box, about which we want to say as little as possible, instead of additional structural properties we add a global requirement of soundness with respect to constraint deductions, and preservation of congruence properties of $\approx$. Since the formalization of constraints in Church's type theory only requires the added presence of a reserved type γ of constraints, and for each type α an equality relation symbol $\approx_{\gamma\alpha\alpha}$ in the language, there is nothing to add to the basic framework save interpretations for any new constant symbols and a new forcing relation between worlds w and the carriers D^{γ}_{w} of the constraints. We represent the new forcing relation with the same symbol as logical forcing, since, as with the proof theory, we can always tell which one we are using by inspecting the types of the terms present.

However, in the fragment *hoHH(C)*, only positive terms are allowed in atomic formulas. Thus it is sufficient to define the forcing relation $w \Vdash_{\eta} F$, for PA-generated formulas F, which include both goals and clauses. The relevance that positive terms have in the syntax of *hoHH(C)* will be also reflected in the semantics by defining a *semantic counterpart* to the set of positive terms. This means defining, for each type α and world w, a subset $D^{\alpha+}_{w} \subseteq D^{\alpha}_{w}$, where positive terms of type α must be interpreted.

Definition 14. *A uniform $\mathcal{C}$ -*Kripke model for *hoHH(C) is a triple $\langle \mathcal{K}, [\![\ ]\!], \eta \rangle$, where:*

- $\mathcal{K} = \langle W, \leq, \mathcal{D}, \Vdash \rangle$ *satisfies the requirements of Definition 9 with the following changes:*
  - *For each type α and world w, there is a distinguished subset $D^{\alpha+}_{w}$ of D^{α}_{w} (written D^{+}_{w} when the type is not relevant).*
  - *The forcing relation $\Vdash$ is extended to a relation between W and $D^{o} \cup D^{\gamma}$, and it is defined for (at least) the members of $D^{o} \cup D^{\gamma}$ corresponding to $[\![F]\!]$, for all PA-generated formulas F.*
    As for the logical conditions of this definition, Conditions 6 and 7 of the definition of models for the full theory of types are restricted to members of $D^{\alpha+}_{w}$:
    6'. $w \Vdash \underline{\exists} \cdot f^{o\alpha}$ iff for some $d \in D^{\alpha+}_{w}$, $w \Vdash f \cdot d$,
    7'. $w \Vdash \underline{\forall} \cdot f^{o\alpha}$ iff for every $w' \geq w$ and $d \in D^{\alpha+}_{w'}$, $w' \Vdash f \cdot d$.
- $[\![\ ]\!]$ *is a $\mathcal{K}$-interpretation and η a $\mathcal{K}$-environment, such that for any $w \in W$, if t^{α} is a positive term, then $[\![t]\!]_{\eta_w} \in D^{\alpha+}_{w}$. As a consequence $\eta_w(x) \in D^{\alpha+}_{w}$, for every variable x.*

- *In addition $\Vdash$ must satisfy the following $\mathcal{C}$ -*conditions *for every $w \in W$:*
  - *$\mathcal{C}$ -*soundness*: For every Γ, C, if $\Gamma \vdash_{\mathcal{C}} C$ then, if $w \Vdash_{\eta} \Gamma$ then $w \Vdash_{\eta} C$.*
  - Congruence*:*
    (a) For every A_r, A'_r, such that $w \Vdash_{\eta} A_r \approx A'_r$, if $w \Vdash_{\eta} A_r$, then $w \Vdash_{\eta} A'_r$.
    (b) For every flexible atom $Xt_1 \ldots t_n$ and positive term t, such that $w \Vdash_{\eta} X \approx t$, if $w \Vdash_{\eta} \Lambda((Xt_1 \ldots t_n)[t/X])$, then $w \Vdash_{\eta} (Xt_1 \ldots t_n)$.
  - *$\mathcal{C}$ -*existential condition*: For every $w \in W$, $r \in D^{\gamma\alpha}_{w}$, $w \Vdash \underline{\exists}^{\gamma(\gamma\alpha)} \cdot r \iff$ for some $d \in D^{\alpha+}_{w}, w \Vdash r \cdot d$.*

We will repeatedly make use of the following technical consequence of these definitions, whose proof is by a straightforward induction on λ-term structure.

Lemma 2 (Substitution). *Let $\langle \mathcal{K}, [\![\ ]\!], \eta \rangle$ be a uniform $\mathcal{C}$ -Kripke model. For any positive term t, any PA-formula F, and any world w, $[\![F[t/x]]\!]_{\eta_w} = [\![F]\!]_{\eta_w[x:=[\![t]\!]_{\eta_w}]}$.*

4 Soundness and Completeness of *hoHH(C)*

Since the language *hoHH(C)* is based on the calculus $ho\mathcal{UC}$, our aim is to prove the equivalence between provability in $ho\mathcal{UC}$ and validity in every uniform $\mathcal{C}$ -Kripke model.

4.1 Soundness of $ho\mathcal{UC}$

We begin by showing that what is provable in $ho\mathcal{UC}$ is true.

Theorem 1 (Soundness). *For every Δ, Γ, G, if $\Delta; \Gamma \vdash_{ho\mathcal{UC}} G$ holds, then the sequent $\Delta; \Gamma \vdash G$ is satisfied in every uniform $\mathcal{C}$ -Kripke model for hoHH(C).*

Proof. Let $\langle \mathcal{K}, \eta \rangle$ be a uniform $\mathcal{C}$ -Kripke model for *hoHH(C)*. The proof proceeds by induction on the length of the proof of the sequent $\Delta; \Gamma \vdash G$. The inductive hypothesis is that all sequents with shorter proofs are satisfied at every world in every uniform $\mathcal{C}$ -Kripke model. We consider the the most interesting case of the induction here, and leave the rest as an exercise for the reader.

Let w be any world such that $w \Vdash_\eta \Delta$ and $w \Vdash_\eta \Gamma$. If $\Delta; \Gamma \vdash_{ho\mathcal{UC}} A_r$ is derived using the $(Clause)$ rule as a final step, then there is a variant $\forall \overline{x}(G' \Rightarrow A'_r)$ of a clause of $elab(\Delta)$, such that the sequent $\Delta; \Gamma \vdash \exists \overline{x}((A'_r \approx A_r) \wedge G')$ has a proof shorter than the proof of $\Delta; \Gamma \vdash A_r$. By the induction hypothesis, $w \Vdash_\eta \exists \overline{x}((A'_r \approx A_r) \wedge G')$. Then there are $\overline{d} \in D^+_w$ such that $w \Vdash_{\eta[\overline{x}:=\overline{d}]} A'_r \approx A_r$ and $w \Vdash_{\eta[\overline{x}:=\overline{d}]} G'$. It is easy to prove that $w \Vdash_\eta \Delta$ implies $w \Vdash_\eta \forall \overline{x}(G' \Rightarrow A'_r)$, then $w \Vdash_{\eta[\overline{x}:=\overline{d}]} G' \Rightarrow A'_r$. We have $w \Vdash_{\eta[\overline{x}:=\overline{d}]} A'_r$, because $w \Vdash_{\eta[\overline{x}:=\overline{d}]} G'$. We conclude $w \Vdash_\eta A_r$, because $\overline{x}$ are not free in A_r, $w \Vdash_{\eta[\overline{x}:=\overline{d}]} A'_r \approx A_r$, and by the congruence of $\langle \mathcal{K}, \eta \rangle$. □

4.2 Completeness

The proof of the completeness is based on the construction of a particular uniform $\mathcal{C}$ -Kripke model, $\mathcal{U}^{\mathcal{C}}$, in such a way that $\Vdash$ coincides with $\vdash_{ho\mathcal{UC}}$ *when the latter is defined*. We are not able to completely define $\Vdash$ this way because provability of a sequent $\Delta; \Gamma \vdash G$ in $ho\mathcal{UC}$ only makes sense when G is a goal, whereas the relation $\Vdash$ must be defined for more general (PA-generated) formulas.

To define the model, we will need to make use of a restricted version of the so-called Lindenbaum Lemma for the constraint system $\mathcal{C}$.

The Lindenbaum construction for $\mathcal{C}$ In its original form, in classical logic, the Lindenbaum lemma (see e.g. [25]) states that a consistent set of sentences can be extended to a maximal consistent set. In our setting, to prove completeness, we only need to ensure that constraint theories satisfy a pure-variable form of the existential part of this claim, namely that if a formula A is not derivable from a theory Γ then the theory can be extended to one that still does not prove A and has the existence property: if it derives an existential formula, it proves a pure-variable instance over a language enriched only with new variables, but with no new constants.

In the following we will assume that X is a complete set of variables, by which we mean that it contains countably many variables $x^\alpha_1, x^\alpha_2 \ldots$ for each type expression α.

Definition 15. *A set of constraints Γ is said to be* $\exists$-saturated *over a complete set of variables X if for any constraint C, whenever $\Gamma \vdash_{\mathcal{C}} \exists x C$ then for some y in X, we have $\Gamma \vdash_{\mathcal{C}} C[y/x]$.*

Lemma 3. *Let V be a complete set of variables (assumed to be the base set of variables used to build terms in this paper), and let X be a disjoint complete set of variables. For any Δ, Γ and G, with free variables in V, if $\Delta; \Gamma \vdash G$ is not derivable in $ho\mathcal{UC}$, then there is an extension $\hat{\Gamma}$ of Γ, with free variables in $V \cup X$, which:*

- *is $\exists$-saturated over $V \cup X$, and*
- *maintains $\Delta; \hat{\Gamma} \not\vdash_{ho\mathcal{UC}} G$.*

Some adaptation is required to make the proof of Lindenbaum lemma [25] suitable for *hoHH(C)*. In particular, instead of adding Henkin *constants* as witnesses for existential formulas, fresh *variables* are added. This is needed in the completeness theorem because of the special character of quantifier rules where variables and constraints, rather than terms, act as witnesses. Otherwise, the proof is straightforward.

The uniform $\mathcal{C}$-Kripke model $\mathcal{U}^{\mathcal{C}}$ We begin now the construction of the model we will use to establish completeness. We start with a countable sequence of countable complete sets of fresh variables $X_0 \subset X_1 \subset \cdots \subset X_n \subset \cdots$ where each $X_{i+1} \setminus X_i$ is countably infinite.

Definition 16. *Given a constraint system $\mathcal{C}$ we define $\mathcal{U}^{\mathcal{C}} = \langle W, \leq, \mathcal{D}, \Vdash \rangle$, as follows:*

- *$(W, \leq)$, the ordered set of worlds is defined as:*
  $W = \{\langle \Delta, \Gamma, n\rangle \mid \Delta$ is a finite set of clauses over Σ and X_n; Γ is a set of constraints over Σ and X_n, $\exists$-saturated over $X_n\}$.
  $\langle \Delta_1, \Gamma_1, n_1\rangle \leq \langle \Delta_2, \Gamma_2, n_2\rangle \stackrel{\text{def}}{\iff} \Delta_1 \subseteq \Delta_2, \Gamma_1 \subseteq \Gamma_2$ and $n_1 \leq n_2$.
- *$\mathcal{D} = \{\mathsf{D}_w | w \in W\}$.*
  For each $w = \langle \Delta, \Gamma, n\rangle$, D_w is defined as follows:
  D^{α}_w is the set of open λ-terms in normal form of type α over Σ and the set of variables X_n.
  $D^{\alpha+}_w$ is the subset consisting of the positive terms of D^{α}_w.
  $\mathsf{Const}_w(c) = c, \quad c \in \Sigma. \qquad \mathsf{App}_w(t_1, t_2) = \Lambda(t_1 t_2)$.
- *The relation $\Vdash$ is defined for the elements of $D^o_w \cup D^{\gamma}_w$ that are PA-generated formulas. In order to define $\langle \Delta, \Gamma, n\rangle \Vdash F$, we use induction on the non-positive depth $\delta(F)$:*
  (1) If F is a constraint or a positive logical formula ($\delta(F) = 0$), then $\langle \Delta, \Gamma, n\rangle \Vdash F \stackrel{\text{def}}{\iff} \Delta; \Gamma \vdash_{ho\mathcal{UC}} F$. This case includes pure constraints, and rigid and flexible atoms.
  (2) For PA-formulas F that do not satisfy the preceding condition ($\delta(F) > 0$), $\langle \Delta, \Gamma, n\rangle \Vdash F$ is defined according to the definition of $\Vdash$ for uniform C-Kripke models. For instance, $\langle \Delta, \Gamma, n\rangle \Vdash \forall x^{\alpha} F \stackrel{\text{def}}{\iff}$ for every $\langle \Delta', \Gamma', n'\rangle \in W$, $\langle \Delta, \Gamma, n\rangle \leq \langle \Delta', \Gamma', n'\rangle$, and every $t \in D^{\alpha+}_{\langle \Delta', \Gamma', n'\rangle}$, $\langle \Delta', \Gamma', n'\rangle \Vdash \mathsf{App}_{\langle \Delta', \Gamma', n'\rangle}(\lambda x.F, t)$, i.e., $\langle \Delta', \Gamma', n'\rangle \Vdash F[t/x]$.

Note that the relation $\Vdash$ in this model is defined by induction on the non-positive depth of formulas, with positive and constraints formulas as the base case. In this way, we avoid problems with impredicativity, by working with a well-founded order. When a quantified non-positive formula is instantiated with positive terms, the instance is simpler with respect to that order, in accordance with Lemma 1.

We will show that when $\mathcal{U}^{\mathcal{C}}$ is supplied with a particular environment, it gives rise to a uniform $\mathcal{C}$ -Kripke model for *hoHH($\mathcal{C}$)*.

For $\mathcal{U}^{\mathcal{C}}$, given any environment η there is a unique induced environmental model $[\![\ ]\!]$, satisfying, for all worlds w, the condition $[\![t]\!]_{\eta_w} = t\theta_{\eta_w}$, where θ_{η_w} is the substitution mapping each x free in t to $\eta_w(x)$. Let id be the *identity environment*: for every $w = \langle \Delta, \Gamma, n\rangle \in W$, id_w maps each variable $x \in X_n$ to itself, and let $[\![\ ]\!]$ be the induced environmental model. We will prove that $\langle \mathcal{U}^{\mathcal{C}}, id\rangle$ is a uniform $\mathcal{C}$ -Kripke model for *hoHH($\mathcal{C}$)*.

We first establish some technical properties of the relation $\Vdash$, that defines $\mathcal{U}^{\mathcal{C}}$.

Lemma 4. *For every world $\langle \Delta, \Gamma, n\rangle$, and every clause D over Σ and X_n, if for all $D' \in elab(D)$, $\langle \Delta, \Gamma, n\rangle \Vdash D'$, then $\langle \Delta, \Gamma, n\rangle \Vdash D$.*

The proof is by induction on the sum of the non-positive depths of the formulas of $elab(D)$.

Lemma 5. *For every world $\langle \Delta, \Gamma, n\rangle$, and every $F \in \Delta \cup \Gamma$, we have $\langle \Delta, \Gamma, n\rangle \Vdash F$.*

Proof. Sketch. The proof of $\langle \Delta, \Gamma, n\rangle \Vdash C$, $C \in \Gamma$, is immediate, because, for all $C \in \Gamma$, $\Gamma \vdash_{\mathcal{C}} C$. In order to prove $\langle \Delta, \Gamma, n\rangle \Vdash D$, $D \in \Delta$, we proceed by induction on the non-positive depth of D. The base case implies that D is positive. It is then easy to show that $\Delta; \Gamma \vdash_{ho\mathcal{UC}} D$, so $\langle \Delta, \Gamma, n\rangle \Vdash D$, by definition. For the inductive case, in order to prove $\langle \Delta, \Gamma, n\rangle \Vdash D$, we first show that $\langle \Delta, \Gamma, n\rangle \Vdash D'$ for every $D' \in elab(D)$, then conclude $\langle \Delta, \Gamma, n\rangle \Vdash D$, using Lemma 4. If $D' \in elab(D)$, then $D' \equiv \forall \overline{x}(G \Rightarrow A_r)$. In order to establish $\langle \Delta, \Gamma, n\rangle \Vdash \forall \overline{x}(G \Rightarrow A_r)$ we need to make use of the following claim:

$$\text{If } \forall \overline{x}(G \Rightarrow A_r) \in elab(\Delta), \text{ then for all } \langle \Delta', \Gamma', n'\rangle \in W, \langle \Delta, \Gamma, n\rangle \leq \langle \Delta', \Gamma', n'\rangle \text{ and } \overline{t} \in D^{+}_{\langle \Delta', \Gamma', n'\rangle}, \text{ if } \langle \Delta', \Gamma', n'\rangle \Vdash G[\overline{t}/\overline{x}] \text{ then, } \Delta'; \Gamma' \vdash_{ho\mathcal{UC}} G[\overline{t}/\overline{x}].$$

From this fact, proving $\langle \Delta, \Gamma, n\rangle \Vdash \forall \overline{x}(G \Rightarrow A_r)$ can be reduced to proving that $\Delta; \Gamma \vdash_{ho\mathcal{UC}} A_r[\overline{t}/\overline{x}]$ (for any $\overline{t}$), if $\Delta; \Gamma \vdash_{ho\mathcal{UC}} G[\overline{t}/\overline{x}]$. But this is easy to prove using $(Subst)$ and $(Clause)$, since $\forall \overline{x}(G \Rightarrow A_r) \in elab(\Delta)$.

The proof of the claim is by induction on $\delta(G[\overline{t}/\overline{x}])$. The base case is trivial. For the inductive step, we must consider the possible structure of $G[\overline{t}/\overline{x}]$. We show here the most interesting case:

$G[\overline{t}/\overline{x}] \equiv D' \Rightarrow G'$: If $\Delta'' = \Delta' \cup \{D'\}$, $\langle \Delta'', \Gamma', n'\rangle \Vdash D'$, applying the outer induction on $\delta(D)$, since $D' \in \Delta''$, and observe that in fact $\delta(D') < \delta(D)$, because if $\forall \overline{x}(G \Rightarrow A_r) \in elab(D)$ and D non-positive, then $\delta(D) > \delta(G) = \delta(G[\overline{t}/\overline{x}])$, by Lemma 1, and $\delta(G[\overline{t}/\overline{x}]) > \delta(D')$. So $\langle \Delta'', \Gamma', n'\rangle \Vdash G'$, since $\langle \Delta', \Gamma', n'\rangle \Vdash D' \Rightarrow G'$ and $\langle \Delta'', \Gamma', n'\rangle \geq \langle \Delta', \Gamma', n'\rangle$. Then, $\Delta', D'; \Gamma' \vdash_{ho\mathcal{UC}} G'$, by the induction on $\delta(G[\overline{t}/\overline{x}])$. Therefore $\Delta'; \Gamma' \vdash_{ho\mathcal{UC}} D' \Rightarrow G'$, according to $(\Rightarrow_R)$. □

Proposition 2. *For all worlds $\langle \Delta, \Gamma, n\rangle$ and goal G, with free variables in X_n:*

$$\textit{If } \langle \Delta, \Gamma, n\rangle \Vdash G, \textit{ then } \Delta; \Gamma \vdash_{ho\mathcal{UC}} G.$$

Proof. By induction on the non-positive depth of G. The argument is similar to that of the claim established in the proof of Lemma 5. □

Lemma 6. *$\langle \mathcal{U}^{\mathcal{C}}, [\![\ ]\!], id\rangle$ is a uniform $\mathcal{C}$ -Kripke model for hoHH($\mathcal{C}$).*

Proof. The requirements of Definition 14 must be proved. It is easy to prove that $(W, \leq)$ is a poset, $\mathcal{D}$ a $(W, \leq)$-indexed *TAS*, $[\![\ ]\!]$ a $\mathcal{U}^{\mathcal{C}}$-interpretation and id a $\mathcal{U}^{\mathcal{C}}$-environment. In addition $[\![t]\!]_{id_w} = t$, so if t is a positive term, $[\![t]\!]_{id_w} \in D^+_w$. Let us show the requirements for $\Vdash$ are satisfied.

Monotonicity requirement. By induction on the non-positive depth of PA-formulas. For the base case, the monotonicity of $\Vdash$ is derived from the monotonicity of $\vdash_{ho\mathcal{UC}}$ with respect to Δ and Γ. The inductive step is straightforward.

Logical conditions. For PA-formulas that are not constraints neither positive logical formulas, those conditions are satisfied by definition. For constraints and positive formulas, the arguments are straightforward[1]. Here we establish one of the more delicate cases:

$\underline{\exists x G}$: Suppose $\langle \Delta, \Gamma, n\rangle \Vdash \exists x G$, then by definition of $\Vdash$ and $(\exists_R)$ rule, there is C such that $\Delta; \Gamma, C \vdash_{ho\mathcal{UC}} G[y/x]$ and $\Gamma \vdash_{\mathcal{C}} \exists y C$, where y is not free in $\Delta, \Gamma, \exists x G$. Since Γ is $\exists$-saturated over X_n, then $\Gamma \vdash_{\mathcal{C}} C[z/y]$ for some $z \in X_n$. By the properties of $ho\mathcal{UC}$, $\Delta; \Gamma, C[z/y] \vdash_{ho\mathcal{UC}} G[z/x]$, because y was fresh. But this implies that $\Delta; \Gamma \vdash_{ho\mathcal{UC}} G[z/x]$, from the fact $\Gamma \vdash_{\mathcal{C}} C[z/y]$ and $(cut_{\mathcal{C}})$. Therefore there is $z \in D^+_{\langle \Delta, \Gamma, n\rangle}$, such that $\langle \Delta, \Gamma, n\rangle \Vdash G[z/x]$.
Conversely if there is $t \in D^+_{\langle \Delta, \Gamma, n\rangle}$ such that $\langle \Delta, \Gamma, n\rangle \Vdash G[t/x]$, then $\Delta; \Gamma \vdash_{ho\mathcal{UC}} G[t/x]$ by definition, and $\Delta; \Gamma, y \approx t \vdash_{ho\mathcal{UC}} G[y/x]$, with y fresh, applying $(subst)$. So $\Delta; \Gamma \vdash_{ho\mathcal{UC}} \exists x G$ in accordance with $(\exists_R)$. Hence we can conclude $\langle \Delta, \Gamma, n\rangle \Vdash \exists x G$, because $\exists x G$ is positive and the definition of $\Vdash$.

The proof for the $\mathcal{C}$ *-conditions* are routine, and left to the reader. □

We will refer to our uniform $\mathcal{C}$ -Kripke model simply as $\mathcal{U}^{\mathcal{C}}$. By the definitions of $[\![\ ]\!]$ and id, the notation $\langle \Delta, \Gamma, n\rangle \Vdash_{id} G$ is equivalent to $\langle \Delta, \Gamma, n\rangle \Vdash G$.

Finally, we prove that the formal system $ho\mathcal{UC}$, is complete for $\mathcal{C}$ -Kripke semantics. That means that any $ho\mathcal{UC}$ sequent true in all of our models is derivable.

Theorem 2 (Completeness of *ho*$\mathcal{UC}$). *For every Δ, Γ, G over Σ, and V, if every uniform $\mathcal{C}$ -Kripke model for hoHH($\mathcal{C}$) satisfies the sequent $\Delta; \Gamma \vdash G$, then $\Delta; \Gamma \vdash_{ho\mathcal{UC}} G$.*

Proof. Suppose, for a contradiction, that there are Δ, Γ, G, such that any uniform $\mathcal{C}$ -Kripke model for *hoHH($\mathcal{C}$)* satisfies $\Delta; \Gamma \vdash G$, but there is no $ho\mathcal{UC}$ derivation of the sequent $\Delta; \Gamma \vdash G$. By the Lindenbaum Lemma (3), there is a set of constraints, Γ', that extends Γ, $\exists$-saturated over certain X_n, such that there is no $ho\mathcal{UC}$ derivation of the sequent $\Delta; \Gamma' \vdash G$.

So in the model $\mathcal{U}^{\mathcal{C}}$, by Lemma 5, $\langle \Delta, \Gamma', n\rangle \Vdash_{id} \Delta$, and $\langle \Delta, \Gamma', n\rangle \Vdash_{id} \Gamma$. Hence, $\langle \Delta, \Gamma', n\rangle \Vdash_{id} G$, because $\mathcal{U}^{\mathcal{C}}$ satisfies $\Delta; \Gamma \vdash G$, by the hypothesis of the theorem. Then by Proposition 2, $\Delta; \Gamma' \vdash_{ho\mathcal{UC}} G$, contradicting the hypothesis of Γ'. □

Logical Intensionality As discussed in the first section, one of our aims was to produce a model theory in which logical equivalence of two logical formulas F_1 and

[1] Notice that those formulas are always goals.

F_2 would not necessarily imply validity of $p(F_1) \Rightarrow p(F_2)$ for every predicate symbol p. Take p be a constant of type $o \rightarrow o$, A_r a rigid atomic formula, and consider $\mathcal{C}$ for which $\approx$ coincides with $\equiv_\lambda$. In the model $\mathcal{U}^{\mathcal{C}}$, $p(A_r) \Rightarrow p(A_r \wedge A_r)$ is *not* forced at the root node $\langle \emptyset, \emptyset, 0 \rangle$, since by Proposition 2, this would mean that $\emptyset; \emptyset \vdash_{ho\mathcal{U}\mathcal{C}} p(A_r) \Rightarrow p(A_r \wedge A_r)$, and hence $\{p(A_r)\}; \emptyset \vdash_{ho\mathcal{U}\mathcal{C}} p(A_r \wedge A_r)$. Since $\nvdash_{\mathcal{C}} A_r \approx A_r \wedge A_r$, this is impossible.

5 Conclusion

We have introduced a semantic framework based on Kripke structures for Intuitionistic Higher-Order Type Theory with constraints to model the declarative content of a representative higher-order constraint logic programming language, with simply typed λ-terms, implication and universal quantification in goals. The underlying logic of λ-Prolog is covered as a special case. We have shown the program calculus sound and complete.

We build on Mitchell-Moggi Kripke λ-models [20], but go well beyond equational reasoning, to model predicates in an impredicative higher-order logic with constraints.

Our results extend earlier work on declarative semantics for some executable fragments of the logic: First-order Hereditarily Harrop formulas [17], classical Higher-order Horn formulas in [26, 2] and semantics for *hoHH* in [5, 14] and [8] for *HH*($\mathcal{C}$).

A key direction for future work is to understand how to adapt the framework defined here to deal with polymorphic types, linear logic or a linear constraint system, or to exploit the constraint framework for specific abstract syntax and metaprogramming applications. It would also be of interest to define Kripke models more sensitive operationally to a specific proof procedure, as well as to study observational equivalence and abstract interpretation in this context, a matter for further research.

Acknowledgements This work was partially supported by the Spanish projects 'MERIT-FORMS': TIN2005-09207-C03-03, 'PROMESAS-CAM': S-0505/TIC/0407. The first author's research was also partially supported by the Pitney-Bowes Corporation.

The authors wish to thank Olivier Hermant and Frank Pfenning for helpful comments.

References

1. P. Andrews. Resolution in type theory. *Journal of Symbolic Logic*, 36(3), 1971.
2. M. Bai and H. 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.
3. C. Benzmüller, C. E. Brown, and M. Kohlhase. Higher order semantics and extensionality. *Journal of Symbolic Logic*, 69:1027–1088, 2004.
4. A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.
5. M. DeMarco. *Higher Order Logic Programming in the Theory of Types*. PhD thesis, Wesleyan University, 1999.
6. M. DeMarco and J. Lipton. Completeness and cut elimination in the intuitionistic theory of types. *The Journal of Logic and Computation*, December 2005.

7. D. M. Gabbay and U. Reyle. N-prolog: an extension of prolog with hypothetical implications i. *Journal of Logic Programming*, 1(4):319–355, 1984.
8. M. García-Díaz and S. Nieva. Providing declarative semantics for HH extended constraint logic programs. In *PPDP '04*, pages 55–66. ACM Press, 2004.
9. J. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge University Press, 1998.
10. L. Henkin. Completeness in the theory of types. *Journal of Symbolic Logic*, 15:81–91, June 1950.
11. J. R. Hindley. *Basic simple type theory*. Cambridge University Press, New York, NY, USA, 1997.
12. J. Jaffar and M. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.
13. R. Jagadeesan, G. Nadathur, and V. A. Saraswat. Testing concurrent systems: An interpretation of intuitionistic logic. In *FSTTCS*, pages 517–528, 2005.
14. E. Lastres. *A Semantics for Logic Programs based on HH Formulas*. PhD thesis, Università di Pisa, 2002.
15. J. Leach and S. Nieva. A higher-order logic programming language with constraints. In *FLOPS'01*, LNCS 2024, pages 108–122. Springer, 2001.
16. J. Leach, S. Nieva, and M. Rodríguez-Artalejo. Constraint logic programming with hereditary Harrop formulas. *Theory and Practice of Logic Programming*, 1(4):409–445, 2001.
17. D. Miller. A logical analysis of modules in logic programming. *Journal of Logic Programming*, pages 79–108, 1989.
18. D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51(1-2):125–157, 1991.
19. J. Mitchell. *Foundations for Programming Languages*. MIT Press, Cambridge, Massachusetts, 1996.
20. J. Mitchell and E. Moggi. Kripke-style models for typed lambda calculus. *Annals of Pure and Applied Logic*, 51:99–124, 1991.
21. G. Nadathur. A proof procedure for the logic of hereditary harrop formulas. *Journal of Automated Reasoning*, 11:115–145, 1993.
22. V. Saraswat. The category of constraints is cartesian closed. In *Proc. of the 7th Symposium on Logic in Computer Science (LICS '92)*, pages 341–345. IEEE Press, 1992.
23. M. Takahashi. A proof of cut-elimination in simple type theory. *J. Math. Soc. Japan*, 19(4):399–410, 1967.
24. A. Tarski. *A decision method for elementary algebra and geometry*. University of California Press, 1951.
25. D. van Dalen. *Logic and Structure*. Springer, 2004.
26. D. A. Wolfram. A semantics for λprolog. *Theoretical Computer Science*, 136:277–289, 1994.