A SEMANTICS FOR INTUITIONISTIC HIGHER-ORDER LOGIC SUPPORTING HIGHER-ORDER ABSTRACT SYNTAX

CHAD E. BROWN

Abstract. We give an intuitionistic extensional higher-order logic supporting higher-order abstract syntax. We give a model proving consistency of the logic, in spite of the fact that it has no standard set theoretic models. The development has been formalized in Coq using the Autosubst package.

1. Introduction

We describe a simply typed intuitionistic extensional higher-order logic [6] supporting higher-order abstract syntax [10]. The logic has an M-set model [4] where M is the monoid of substitutions (up to σ-equality) [1, 7]. We furthermore describe a Coq formalization. The Coq formalization makes use of the Autosubst package [11] to conveniently define untyped lambda terms and reason up to σ-equality. For simply typed λ-terms we use de Bruijn indices [5] combined with Church style typing [6]. Since dependent types are necessary to represent Church style typing, we do not use Autosubst for simply typed terms.

M-sets as models for simple type theory were described in [4]. That taking M to be the monoid of substitutions gives a model for classical nonextensional higher-order logic has been known since then, but unpublished. One can find some information in [12].

The M-set model is essentially the special case of a presheaf category. Hofmann also considered presheaf semantics for higher-order abstract syntax [8]. We leave the clarification of the precise relationship between Hofmann's presheaf models and the M-set model described here to future work.

2. HOL with HOAS

We give a higher-order logic which supports reasoning about untyped λ-terms using higher-order abstract syntax.

We begin by defining a set $\mathcal{T}$ of *simple types*:[1]

- ι : This is a base type which will be interpreted as untyped λ-terms (in de Bruijn representation).
- o : This is a base type of propositions.
- $\alpha \to \beta$: This is the type of functions from α to β.

We next define a family of simply typed terms. We informally describe simply typed terms using names, but the formalization uses de Bruijn indices.

Date: August 12, 2014.

[1]See `stp` in the formalization. In the formalization we also include a type of natural numbers, but we ignore this type here.

For each type $\alpha \in \mathcal{T}$, let $\mathcal{V}_\alpha$ be a countably infinite set of variables of type α. We define a set Λ_α of *terms of type* α as follows:

- For each variable x of type α, $x \in \Lambda_\alpha$.
- $\mathsf{L} \in \Lambda_{(\iota\to\iota)\to\iota}$. (This is a constant which will represent untyped λ-abstractions.)
- $\mathsf{A} \in \Lambda_{\iota\to\iota\to\iota}$. (This is a constant which will represent untyped application.)
- If $s \in \Lambda_{\alpha\to\beta}$ and $t \in \Lambda_\alpha$, then $(st) \in \Lambda_\beta$.
- If x is a variable of type α and $s \in \Lambda_\beta$, then $(\lambda x.s) \in \Lambda_{\alpha\to\beta}$.
- If $s, t \in \Lambda_o$, then $(s \to t) \in \Lambda_o$.
- If x is a variable of type α and $s \in \Lambda_o$, then $(\forall x.s) \in \Lambda_o$.
- If $s, t \in \Lambda_\alpha$, then $(s = t) \in \Lambda_o$.

We use common conventions to omit parentheses. We define $\mathcal{F}s$ to be the free variables of s and for sets A of terms we define $\mathcal{F}A$ to be $\bigcup_{s\in A}\mathcal{F}s$. We assume a capture avoiding substition s^x_t is defined. Terms of type o are called *propositions*.

We define $\bot$ to be the proposition $\forall p.p$ where x is a variable of type o.

We define $\doteq_\alpha \in \Lambda_{\alpha\to\alpha\to o}$ to be Leibniz equality: $\lambda xy.\forall p.px \to py$. We will usually write $s\doteq t$ for $(\doteq_\alpha s)t$. It is redundant to include equality as a primitive term constructor since we can always use Leibniz equality. Indeed Leibniz equality will still play a role. However (in the formalization at least) working with primitive equality is sometimes easier than working with Leibniz equality.

Before giving the proof calculus, we briefly describe some of the complications that arise when defining simply typed λ-terms using de Bruijn indices. In the first clause above, we say each variable x of type α is a term of type α. However, if we use de Bruijn indices, we will have natural numbers n not associated with any type. This is easy enough to remedy by having an environment $e : \mathbb{N} \to \mathcal{T}$. That is, instead of defining Λ_α as above, we define Λ^e_α.[2] The e changes in some of the clauses of the definition. For example, for the λ binder we would have a clause that reads:

- If $s \in \Lambda^{\alpha::e}_\beta$, then $(\lambda_\alpha s) \in \Lambda^e_{\alpha\to\beta}$.

In this clause, $(\alpha :: e)$ means the function sending 0 to α and sending $n + 1$ to $e(n)$. Making this formal definition of simply typed terms is not difficult. However, defining substitution and shifting (a special case of renaming) is tricky. It is not clear if the techniques used by Autosubst can be applied in an example like this with dependent types, but we leave this question for others to investigate. The author defined substitution and shifting by hand and proved the appropriate properties.[3]

Since our focus is not on de Bruijn indices we return the informal version using names. We define a $(\beta\eta)$ conversion relation on terms of the same type in two steps. We first define $s\Rightarrow_1 t$ if s β or η reduces to t in one step. We then define $s\approx t$ to be the reflexive, symmetric, transitive closure of $\Rightarrow_1$.[4]

[2]See `stm` in the formalization.

[3]See `stmshift_var`, `stmshift_aux`, `stmshift`, `stmparsubst` and `stmsubst` in the formalization. Also, `stmshift_var_eq_match_eq` for an example of a lemma which was needed but may not hold for some definitions of shifting.

[4]See `conv_1` and `conv` in the formalization.

$$\frac{}{\Gamma \vdash s}\ s \in \Gamma \qquad \frac{\Gamma, s \vdash t}{\Gamma \vdash s \to t} \qquad \frac{\Gamma \vdash s \to t \qquad \Gamma \vdash s}{\Gamma \vdash t} \qquad \frac{\Gamma \vdash s}{\Gamma \vdash \forall x.s}\ x \in \mathcal{V}_\alpha \setminus \mathcal{F}\Gamma$$

$$\frac{\Gamma \vdash \forall x.s}{\Gamma \vdash s^x_t}\ x \in \mathcal{V}_\alpha, t \in \Lambda_\alpha \qquad \frac{}{\Gamma \vdash s = s} \qquad \frac{\Gamma \vdash s = t}{\Gamma \vdash s \dot{=} t} \qquad \frac{\Gamma \vdash s}{\Gamma \vdash t}\ s \approx t$$

$$\frac{\Gamma, s \vdash t \qquad \Gamma, t \vdash s}{\Gamma \vdash s = t}\ s, t \in \Lambda_o \qquad \xi\ \frac{\Gamma \vdash s = t}{\Gamma \vdash (\lambda x.s) = (\lambda x.t)}\ x \in \mathcal{V}_\alpha \setminus \mathcal{F}\Gamma \text{ AND } s, t \in \Lambda_\beta$$

$$\frac{\Gamma \vdash (\mathsf{A} s_1 t) = (\mathsf{A} s_2 t)}{\Gamma \vdash s_1 = s_2} \qquad \frac{\Gamma \vdash (\mathsf{A} s t_2) = (\mathsf{A} s t_2)}{\Gamma \vdash t_1 = t_2} \qquad \frac{\Gamma \vdash (\mathsf{L} s) = (\mathsf{L} t)}{\Gamma \vdash s = t} \qquad \frac{\Gamma \vdash (\mathsf{A} s t) = (\mathsf{L} u)}{\Gamma \vdash \bot}$$

FIGURE 1. Proof Calculus

We define $\Gamma \vdash s$ where Γ is a set of propositions and s is a proposition by the rules given in Figure 1.[5] Most of these are the usual rules of natural deduction. For equality we use reflexivity as the introduction rule and use Leibniz equality to give the elimination rule. The conversion rule ensures the logic respects $\beta\eta$-equivalence. In addition we have propositional extensionality and ξ-extensionality rules. (It is well-known that η combined with ξ give full functional extensionality [3].) We then have rules for reasoning with higher-order abstract syntax. These allow us to infer that the components of untyped applications are equal if two applications are equal. The most surprising rule states that if two untyped λ-abstractions are equal, then their bodies (which have type $\iota \to \iota$) are equal. Finally untyped applications are different from untyped λ-abstractions.

It is not clear if this logic has a model at all. Since untyped applications are different from untyped λ-abstractions, there is no trivial model with ι interpreted as a singleton. However, the rule for L implies L is an injection from the function type $\iota \to \iota$ to ι. Taken together, we conclude there is no standard set-theoretic model since this would contradict a form of Cantor's Theorem.

We will give an M-set model. This model will have the added benefit that the intepretation of ι consists precisely of untyped λ-terms. When we write closed terms of type ι using A and L, they will be interpreted as the corresponding closed untyped λ-terms. For example, the interpretation of the simply typed term

$$\mathsf{A}(\mathsf{L}(\lambda x.\mathsf{A}xx))(\mathsf{L}(\lambda x.\mathsf{A}xx))$$

will be the untyped λ-term

$$(\lambda(00))(\lambda(00)).$$

[5]In the formalization see `nd`.

3. Untyped Lambda Terms and the Monoid

The set $\hat{\Lambda}$ of untyped λ terms are given by the grammar

$$s, t ::= n|(st)|(\lambda s)$$

where n ranges over natural numbers (de Bruijn indices [5]). We use Autosubst [11] in Coq [9] to formalize untyped λ-terms. Autosubst provides a definition of a substitution operation for free and also gives many properties. Autosubst also gives tactics `asimpl` and `autosubst` which were used repeatedly to solve goals in the formal proofs.

Substitutions (σ, τ) are functions from natural numbers to $\hat{\Lambda}$. Let M be the set of all substititutions. The substitution ε which maps each n to the term n is called the *identity substitution*. Given $\sigma, \tau \in M$, we can compose σ and τ to form $\sigma\tau$ by taking each n to the term $\tau(\sigma n)$ (where the application of τ to σn is via the substitution operation). Using functional extensionality it is easy to see that this operation on substitutions is associative. Also, ε is a two-sided identity for the operation. Hence M is a monoid. Furthermore, Autosubst builds in these facts so that it is easy to make use of the monoid properties in the formalization.

An M-set is a set A with an operation taking $a \in A$ and $\sigma \in M$ to an element $a\sigma \in A$. This operation is called the *action* of the M-set. The action must satisfy two properties: $a\varepsilon = a$ for all $a \in A$ and $(a\sigma)\tau = a(\sigma\tau)$ for all $a \in A$ and $\sigma, \tau \in M$.

We will interpret simple types as M-sets.

A very easy example of an M-set is $\hat{\Lambda}$ itself. We take the action to be the substitution operation. Indeed this will be the interpretation of ι.

In the formalization we define a record type of M-sets consisting of a type of elements with a partial equivalence relation (per) on it (representing the set A with its intended notion of equality) and an action respecting the per and satisfying the two properties above.[6]

4. The Model

We define an M-set $\mathcal{D}_\alpha$ for each $\alpha \in \mathcal{T}$. As already mentioned we define $\mathcal{D}_\iota$ to be the M-set of untyped λ-terms $\hat{\Lambda}$ with the action given by the substitution operation.[7]

We have many options for interpreting the type o of propositions. In [4] we considered two possibilities giving classical (nonextensional) models: $\mathcal{D}_o$ could be the two element set with a trivial action or $\mathcal{D}_o$ could be the power set of M with an action taking $X\sigma$ to $\{\tau|\sigma\tau \in X\}$. Here we take $\mathcal{D}_o$ to be a Heyting algebra given by a subset of the power set of M. This Heyting algebra corresponds to how truth values are interpreted in a presheaf topos.

To be specific we take $\mathcal{D}_o$ to be the set of right ideals of M. A set $X \subseteq M$ is a *right ideal* if $\sigma\tau \in X$ whenever $\sigma \in X$ and $\tau \in M$. Note that two specific right ideals are the empty set and M. In fact, arbitrary intersections and arbitrary unions of right ideals are right ideals. This is enough to know the right ideals form a Heyting algebra. The

[6]See `Mset` in the formalization.

[7]See `Di` in the formalization.

action on $\mathcal{D}_o$ is given by taking $X\sigma$ to be $\{\tau|\sigma\tau \in X\}$. It is easy to see that $X\sigma$ is a right ideal: If $\sigma\tau \in X$ and $\mu \in M$, then $\sigma\tau\mu \in X$ and so $\tau\mu \in X\sigma$.[8]

Finally we interpret function types.[9] Assume $\mathcal{D}_\alpha$ and $\mathcal{D}_\beta$ are M-sets. We take $\mathcal{D}_{\alpha\to\beta}$ to be the M-set

$$\{f : M \times \mathcal{D}_\alpha \to \mathcal{D}_\beta | \forall\sigma\tau \in M.\forall a \in \mathcal{D}_\alpha.f(\sigma, a)\tau = f(\sigma\tau, a\tau)\}.$$

The action is given by taking $f\mu$ to be $f\mu(\sigma, a) = f(\mu\sigma, a)$.

In the formalization this interpretation of types is defined as a recursive function `tpinterp`.

We next need to interpret the simply typed terms to be elements of the corresponding M-set. Two specific terms we will need to interpret are the constants A and L.

Let $\hat{\mathsf{A}}$ be the function given by $\hat{\mathsf{A}}(\sigma, s)(\tau, t) = ((s\tau)t)$. It is easy to check $\hat{\mathsf{A}}$ is in $\mathcal{D}_{\iota\to\iota\to\iota}$ and $\hat{\mathsf{A}}\mu = \hat{\mathsf{A}}$.

Let $S \in M$ be the substitution taking each natural number n to the de Bruijn index $n + 1$. We take $\hat{\mathsf{L}}$ to be the function taking $\hat{\mathsf{L}}(\sigma, f) = (\lambda f(S, 0))$ for each $\sigma \in M$ and $f \in \mathcal{D}_{\iota\to\iota}$. It is easy to check $\hat{\mathsf{L}} \in \mathcal{D}_{(\iota\to\iota)\to\iota}$ and $\hat{\mathsf{L}}\mu = \hat{\mathsf{L}}$.

In order to prove soundness of the proof system for this interpretation of L we will need to know the function sending $f \in \mathcal{D}_{\iota\to\iota}$ to $\hat{\mathsf{L}}(\varepsilon, f)$ is injective. We argue this fact here. Let $f, g \in \mathcal{D}_{\iota\to\iota}$ be given. Suppose $\hat{\mathsf{L}}(\varepsilon, f) = \hat{\mathsf{L}}(\varepsilon, g)$. This means $(\lambda f(S, 0)) = (\lambda g(S, 0))$. This means $f(S, 0)$ and $g(S, 0)$ are the same untyped λ-term. We will prove $f = g$. Let $\sigma \in M$ and $s \in \mathcal{D}_\iota$. Let $s :: \sigma$ be the substitution sending 0 to s and $n + 1$ to $\sigma(n)$. Since $f, g \in \mathcal{D}_{\iota\to\iota}$ we know

$$\begin{aligned} f(\sigma, s) = f(S(s :: \sigma), 0(s :: \sigma)) = f(S, 0)(s :: \sigma) = g(S, 0)(s :: \sigma) \\ = g(S(s :: \sigma), 0(s :: \sigma)) = g(\sigma, s) \end{aligned}$$

as desired.

We are now in a position to define the evaluation function for simply typed terms. Since we need to interpret variables, the evaluation function will depend on an assignment sending each variable $x \in \mathcal{V}_\alpha$ to an element of $\mathcal{D}_\alpha$. As in [4] the evaluation function also depends on an element of the monoid M. Given an assignment φ and substitution $\sigma \in M$, we let $\varphi\sigma$ be the assignment taking x to $\varphi(x)\sigma$ (i.e., we can act on assignments). Also, given an assignment φ, a variable $x \in \mathcal{V}_\alpha$ and an element $a \in \mathcal{D}_\alpha$, we let φ^x_a be the substitution which sends x to a and each other y to $\varphi(y)$.

We will denote the evaluation function by $[\![s]\!]^\sigma_\varphi$ where $s \in \Lambda_\alpha$, $\sigma \in M$ and φ is an assignment.[10] We define it by giving the following equations.

$$[\![x]\!]^\sigma_\varphi = \varphi x$$

$$[\![\mathsf{A}]\!]^\sigma_\varphi = \hat{\mathsf{A}}$$

$$[\![\mathsf{L}]\!]^\sigma_\varphi = \hat{\mathsf{L}}$$

$$[\![st]\!]^\sigma_\varphi = [\![s]\!]^\sigma_\varphi(\varepsilon, [\![t]\!]^\sigma_\varphi)$$

[8]See `Do` in the formalization.

[9]See `Dar` in the formalization.

[10]See `eval` in the formalization. Note that the fact that we used Church style typing allows us to define `eval` as a total function.

$$[\![\lambda x.s]\!]^{\sigma}_{\varphi}(\tau, a) = [\![s]\!]^{\sigma\tau}_{(\varphi\tau)^{x}_{a}}$$

$$[\![s \to t]\!]^{\sigma}_{\varphi} = \bigcup\{Z \in \mathcal{D}_o | \forall mu \in M.\mu \in Z \to \varepsilon \in [\![s]\!]^{\sigma\mu}_{\varphi\mu} \to \varepsilon \in [\![t]\!]^{\sigma\mu}_{\varphi\mu}\}$$

$$[\![\forall x.s]\!]^{\sigma}_{\varphi} = \{\tau \in M | \forall \mu \in M.\forall a \in \mathcal{D}_{\alpha}.\varepsilon \in [\![s]\!]^{\sigma\tau\mu}_{(\varphi\tau\mu)^{x}_{a}}\} \text{ where } x \in \mathcal{V}_{\alpha}$$

$$[\![s = t]\!]^{\sigma}_{\varphi} = \{\tau \in M | [\![s]\!]^{\sigma\tau}_{\varphi\tau} = [\![t]\!]^{\sigma\tau}_{\varphi\tau}\}$$

By induction on s one can prove the following theorem.[11]

Theorem 4.1. $[\![s]\!]^{\sigma}_{\varphi}\tau = [\![s]\!]^{\sigma\tau}_{\varphi\tau}$.

Using Theorem 4.1 it becomes clear that we have alternative characterizations of when the interpretation of an implication or equation.

Lemma 4.1. *Let $s, t \in \Lambda_o$, $\sigma, \tau \in M$ and φ be an assignment. τ is in $[\![s \to t]\!]^{\sigma}_{\varphi}$ if and only if for all $\mu \in M$ $\tau\mu \in [\![s]\!]^{\sigma}_{\varphi}$ implies $\tau\mu \in [\![t]\!]^{\sigma}_{\varphi}$.*

Lemma 4.2. *Let $s, t \in \Lambda_{\alpha}$, $\sigma, \tau \in M$ and φ be an assignment. τ is in $[\![s = t]\!]^{\sigma}_{\varphi}$ if and only if $[\![s]\!]^{\sigma}_{\varphi}\tau = [\![t]\!]^{\sigma}_{\varphi}\tau$.*

Given Lemmas 4.1 and 4.2 the reader would be justified in wondering why these characterizations were not used in the definition of $[\![s]\!]^{\sigma}_{\varphi}$. Indeed they could be. The definitions were chosen to look similar to the definition in the universal quantification case. It is possible that there is a characterization of the universal quantifier case similar to Lemmas 4.1 and 4.2. If this is the case, then an alternative definition of the evaluation function would be justified. We leave investigation of this to future work.

The following results are easy to prove and justify soundness of conversion.

Lemma 4.3. $[\![s^{x}_{t}]\!]^{\sigma}_{\varphi} = [\![s]\!]^{\sigma}_{\varphi^{x}_{[\![t]\!]^{\sigma}_{\varphi}}}$

Lemma 4.4. $[\![(\lambda x.s)t]\!]^{\sigma}_{\varphi} = [\![s^{x}_{t}]\!]^{\sigma}_{\varphi}$

Lemma 4.5. *If $x \notin \mathcal{F}s$, then $[\![\lambda x.sx]\!]^{\sigma}_{\varphi} = [\![s]\!]^{\sigma}_{\varphi}$*

Lemma 4.6. *If $s \Rightarrow t$, then $[\![s]\!]^{\sigma}_{\varphi} = [\![t]\!]^{\sigma}_{\varphi}$*

Lemma 4.7. *If $s \approx t$, then $[\![s]\!]^{\sigma}_{\varphi} = [\![t]\!]^{\sigma}_{\varphi}$*

Before proceeding to the main result, we prove a lemma about Leibniz equality.[12]

Lemma 4.8. *Let $a, b \in \mathcal{D}_{\alpha}$ be given. If $a\tau = b\tau$, then $\tau \in [\![\dot{=}_{\alpha}]\!]^{\sigma}_{\varphi}(\varepsilon, a)(\varepsilon, b)$.*

Proof. Suppose $a\tau = b\tau$. We need to prove $\tau \in [\![\lambda xy.\forall p.px \to py]\!]^{\sigma}_{\varphi}(\varepsilon, a)(\varepsilon, b)$. By definition we know

$$[\![\lambda xy.\forall p.px \to py]\!]^{\sigma}_{\varphi}(\varepsilon, a)(\varepsilon, b) = [\![\forall p.px \to py]\!]^{\sigma}_{\psi}$$

where ψ is $((\phi)^{x}_{a})^{y}_{b}$. Hence we need to prove $\tau \in [\![\forall p.px \to py]\!]^{\sigma}_{\psi}$. Let $\mu \in M$ and $P \in \mathcal{D}_{\alpha\to o}$ be given. We need to prove $\varepsilon \in [\![px \to py]\!]^{\sigma\tau\mu}_{(\psi\tau\mu)^{p}_{P}}$. We prove this by applying

[11]See `eval_thm1` in the formalization. Actually `eval_thm1` proves two conjuncts. The first conjunct states that the evaluation function gives an element in the domain of the per and the second conjunct corresponds to Theorem 4.1. In the informal presentation, we simply use equality instead of a per so the theorem is simpler.

[12]See `eval_SLeibEq_I` in the formalization.

Lemma 4.1. Let $\nu \in M$ be given and assume $\nu \in [\![px]\!]^{\sigma\tau\mu}_{(\psi\tau\mu)^p_P}$. That is, we assume $\nu \in P(\varepsilon, a\tau\mu)$. We must prove $\nu \in [\![py]\!]^{\sigma\tau\mu}_{(\psi\tau\mu)^p_P}$. That is, we must prove $\nu \in P(\varepsilon, b\tau\mu)$. Since $a\tau = b\tau$ we are done. □

The main result is soundness.[13]

Theorem 4.2. *If* $\Gamma \vdash s$ *and* $\tau \in [\![u]\!]^\sigma_\varphi$ *for every* $u \in \Gamma$, *then* $\tau \in [\![s]\!]^\sigma_\varphi$.

Proof. We prove this by induction on the derivation of $\Gamma \vdash s$. In each case we assume we have a σ, τ and φ such that $\tau \in [\![u]\!]^\sigma_\varphi$ for every $u \in \Gamma$. The inductive hypothesis can be applied to the premises of the rule (possibly changing σ, τ and φ) and we must prove $\tau \in [\![s]\!]^\sigma_\varphi$.

Soundness of the hypothesis rule is clear. Soundness of the conversion rule follows from Lemma 4.7. For the implication introduction and elimination rules we use Lemma 4.1.

For the introduction rule for the universal quantifier suppose x is a variable of type α not free in Γ. We must prove $\tau \in [\![\forall x.s]\!]^\sigma_\varphi$. By definition this means we need to prove $\varepsilon \in [\![s]\!]^{\sigma\tau\mu}_{(\varphi\tau\mu)^x_a}$ for all $\mu \in M$ and $a \in \mathcal{D}_\alpha$. Let $\mu \in M$ and $a \in \mathcal{D}_\alpha$ be given. We will apply the inductive hypothesis with $\sigma\tau\mu$, ε and $(\varphi\tau\mu)^x_a$. Since $\tau \in [\![u]\!]^\sigma_\varphi$ we also know $\tau\mu \in [\![u]\!]^\sigma_\varphi$ for each $u \in \Gamma$. Using Theorem 4.1 we know $\varepsilon \in [\![u]\!]^{\sigma\tau\mu}_{\varphi\tau\mu}$ for each $u \in \Gamma$. Since x is not free in Γ we also know $\varepsilon \in [\![u]\!]^{\sigma\tau\mu}_{(\varphi\tau\mu)^x_a}$ for each $u \in \Gamma$.[14] Hence the inductive hypothesis applies and we have $\varepsilon \in [\![s]\!]^{\sigma\tau\mu}_{(\varphi\tau\mu)^x_a}$ as desired.

We now consider the elimination rule for the universal quantifier. The inductive hypothesis gives $\tau \in [\![\forall x.s]\!]^\sigma_\varphi$. Hence $\varepsilon \in [\![s]\!]^{\sigma\tau\mu}_{(\varphi\tau\mu)^x_a}$ for every $\mu \in M$ and $a \in \mathcal{D}_\alpha$. In particular, $\varepsilon \in [\![s]\!]^{\sigma\tau\varepsilon}_{(\varphi\tau\varepsilon)^x_{[\![t]\!]^{\sigma\tau}_{\varphi\tau}}}$. That is, $\varepsilon \in [\![s]\!]^{\sigma\tau}_{(\varphi\tau)^x_{[\![t]\!]^{\sigma\tau}_{\varphi\tau}}}$. Applying Theorem 4.1 and Lemma 4.3 we have

$$[\![s]\!]^{\sigma\tau}_{(\varphi\tau)^x_{[\![t]\!]^{\sigma\tau}_{\varphi\tau}}} = [\![s]\!]^{\sigma\tau}_{(\varphi^x_{[\![t]\!]^\sigma_\varphi})\tau} = [\![s]\!]^{\sigma}_{(\varphi^x_{[\![t]\!]^\sigma_\varphi})}\tau = [\![s^x_t]\!]^\sigma_\varphi\tau.$$

Hence $\varepsilon \in [\![s^x_t]\!]^\sigma_\varphi\tau$. That is, $\tau \in [\![s^x_t]\!]^\sigma_\varphi$ as desired.

Soundness of the equality introduction rule follows immediately from Lemma 4.2. For the equality elimination rule, suppose we have $\tau \in [\![s = t]\!]^\sigma_\varphi$ by inductive hypothesis. We need to prove $\tau \in [\![s \dot{=} t]\!]^\sigma_\varphi$. By Lemma 4.2 we know $[\![s]\!]^\sigma_\varphi\tau = [\![t]\!]^\sigma_\varphi\tau$. By Lemma 4.8 we know $\tau \in [\![s \dot{=} t]\!]^\sigma_\varphi$ as desired.

Next we turn to the extensionality rules. For propositional extensionality, we need to prove $\tau \in [\![s = t]\!]^\sigma_\varphi$. By Lemma 4.2 it is enough to prove the two right ideals $[\![s]\!]^\sigma_\varphi\tau$ and $[\![t]\!]^\sigma_\varphi\tau$ are equal. Suppose $\mu \in [\![s]\!]^\sigma_\varphi\tau$. That is, $\tau\mu \in [\![s]\!]^\sigma_\varphi$. We can apply the inductive hypothesis with the first premise with Γ, s, $\tau\mu$, σ and φ to conclude $\tau\mu \in [\![t]\!]^\sigma_\varphi$. Hence $\mu \in [\![t]\!]^\sigma_\varphi\tau$ as desired. Similarly, if $\mu \in [\![t]\!]^\sigma_\varphi\tau$, then we can use the inductive hypothesis given by the second premise of the rule to obtain $\mu \in [\![s]\!]^\sigma_\varphi\tau$.

We now consider the ξ rule. Let x be a variable of type α not free in Γ. We need to prove $\tau[\![(\lambda x.s) = (\lambda x.t)]\!]^\sigma_\varphi$. Applying Lemma 4.2 we need to prove the two functions

[13]See `nd_sound` in the formalization.

[14]We need a coincidence result to justify this step. We omit this detail here since the formalization with de Bruijn indices does not require an explicit coincidence result.

$[\![\lambda x.s]\!]^\sigma_\varphi\tau$ and $[\![\lambda x.t]\!]^\sigma_\varphi\tau$ are equal. Let $\mu \in M$ and $a \in \mathcal{D}_\alpha$ be given. Recalling the action on functions, we compute

$$([\![\lambda x.s]\!]^\sigma_\varphi\tau)(\mu, a) = [\![\lambda x.s]\!]^\sigma_\varphi(\tau\mu, a) = [\![s]\!]^{\sigma\tau\mu}_{(\varphi\tau\mu)^x_a}$$

and

$$([\![\lambda x.t]\!]^\sigma_\varphi\tau)(\mu, a) = [\![\lambda x.t]\!]^\sigma_\varphi(\tau\mu, a) = [\![t]\!]^{\sigma\tau\mu}_{(\varphi\tau\mu)^x_a}.$$

Applying the inductive hypothesis with $\sigma\tau\mu$, ε and $(\varphi\tau\mu)^x_a$ we have $\varepsilon \in [\![s = t]\!]^{\sigma\tau\mu}_{(\varphi\tau\mu)^x_a}$. Using Lemma 4.2 we conclude $[\![s]\!]^{\sigma\tau\mu}_{(\varphi\tau\mu)^x_a} = [\![t]\!]^{\sigma\tau\mu}_{(\varphi\tau\mu)^x_a}$ as desired.

Finally we consider the rules for A and L.

We first argue soundness of the two rules giving injectivity of A. In the first rule the inductive hypothesis gives $\tau \in [\![(\mathsf{A}s_1t_1) = (\mathsf{A}s_2t_2)]\!]^\sigma_\varphi$. By definition we have

$$(([\![s_1]\!]^{\sigma\tau}_{\varphi\tau})([\![t]\!]^{\sigma\tau}_{\varphi\tau})) = [\![(\mathsf{A}s_1t)]\!]^{\sigma\tau}_{\varphi\tau} = [\![(\mathsf{A}s_2t)]\!]^{\sigma\tau}_{\varphi\tau} = (([\![s_2]\!]^{\sigma\tau}_{\varphi\tau})([\![t]\!]^{\sigma\tau}_{\varphi\tau})).$$

The only way these two untyped λ-terms, both of them applications, can be equal is if both components are equal. Hence we know

$$[\![s_1]\!]^{\sigma\tau}_{\varphi\tau} = [\![s_2]\!]^{\sigma\tau}_{\varphi\tau}.$$

We conclude $\tau \in [\![s_1 = s_2]\!]^\sigma_\varphi$ as desired. The proof for the second rule is analogous.

Next we argue soundness of the rule stating that terms given by A and L are never equal. The inductive hypothesis gives

$$(([\![s]\!]^{\sigma\tau}_{\varphi\tau})([\![t]\!]^{\sigma\tau}_{\varphi\tau})) = [\![(\mathsf{A}st)]\!]^{\sigma\tau}_{\varphi\tau} = [\![\mathsf{L}u]\!]^{\sigma\tau}_{\varphi\tau} = (\lambda([\![u]\!]^{\sigma\tau}_{\varphi\tau}(S, 0))).$$

That is, an untyped application is equal to an untyped abstraction. This is impossible, and so there cannot be a σ, τ and φ such that $\tau \in [\![u]\!]^\sigma_\varphi$ for every $u \in \Gamma$.

Earlier we previewed soundness of the rule giving injectivity of L. We repeat this argument in the current context. Applying the inductive hypothesis we know $\tau \in [\![(\mathsf{L}s) = (\mathsf{L}t)]\!]^\sigma_\varphi$. By definition we have

$$(\lambda([\![s]\!]^{\sigma\tau}_{\varphi\tau}(S, 0))) = [\![\mathsf{L}s]\!]^{\sigma\tau}_{\varphi\tau} = [\![\mathsf{L}t]\!]^{\sigma\tau}_{\varphi\tau} = (\lambda([\![t]\!]^{\sigma\tau}_{\varphi\tau}(S, 0)))$$

and so

$$[\![s]\!]^{\sigma\tau}_{\varphi\tau}(S, 0) = [\![t]\!]^{\sigma\tau}_{\varphi\tau}(S, 0).$$

We need to prove $\tau \in [\![s = t]\!]^\sigma_\varphi$. That is, we need to prove $[\![s]\!]^{\sigma\tau}_{\varphi\tau} = [\![t]\!]^{\sigma\tau}_{\varphi\tau}$. Recall that $[\![s]\!]^{\sigma\tau}_{\varphi\tau}$ and $[\![t]\!]^{\sigma\tau}_{\varphi\tau}$ are functions in $\mathcal{D}_{\iota\to\iota}$ so it is enough to prove they are equal as functions. Let $\mu \in M$ and $u \in \mathcal{D}_\iota$ be given. Let $(u :: \mu) \in M$ be the substitution taking 0 to u and $n + 1$ to $\mu(n)$. We have

$$[\![s]\!]^{\sigma\tau}_{\varphi\tau}(\mu, u) = [\![s]\!]^{\sigma\tau}_{\varphi\tau}(S, 0)(u :: \mu) = [\![t]\!]^{\sigma\tau}_{\varphi\tau}(S, 0)(u :: \mu) = [\![t]\!]^{\sigma\tau}_{\varphi\tau}(\mu, u)$$

as desired. $\square$

5. Conclusion

We have given an intuitionistic extensional higher-order logic supporting reasoning about untyped λ-terms using higher-order abstract syntax. The logic currently only builds in the fact that applications and abstractions are injective and disjoint. A possible extension would be to include some appropriate induction principle. Candidate induction principles can be found in [8].

On the one hand, the formalization in Coq was made much easier by the use of Autosubst. On the other hand, the use of dependent types to represent simply typed terms in Church-style meant that the current version of Autosubst could not be used at this level.

References

[1] Abadi, M., Cardelli, L., Curien, P.L., Lévy, J.J.: Explicit substitutions. Journal of Functional Programming **1**(4), 375–416 (1991)

[2] Barendregt, H.: The Lambda Calculus: its Syntax and Semantics, revised edn. North-Holland, Amsterdam (1984)

[3] Benzmüller, C., Brown, C.E., Kohlhase, M.: Higher-order semantics and extensionality. The Journal of Symbolic Logic **69**, 1027–1088 (2004)

[4] Brown, C.E.: M-set models. In: Reasoning in Simple Type Theory: Festschrift in Honor of Peter B. Andrews on His 70th Birthday. College Publications (2008)

[5] de Bruijn, N.: Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. Indagationes Mathematicae (Proceedings) **34**(5), 381–392 (1972)

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

[7] Dowek, G., Hardin, T., Kirchner, C.: Higher order unification via explicit substitutions. Information and Computation **157**(1–2), 183 – 235 (2000)

[8] Hofmann, M.: Semantical analysis of higher-order abstract syntax. In: Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science, LICS '99, pp. 204–213. IEEE Computer Society, Washington, DC, USA (1999)

[9] The Coq development team: The Coq proof assistant reference manual. LogiCal Project (2004). URL `http://coq.inria.fr`. Version 8.0

[10] Pfenning, F., Elliot, C.: Higher-order abstract syntax. SIGPLAN Notices **23**(7), 199–208 (1988)

[11] Schäfer, S., Tebbi, T.: Autosubst. URL `https://www.ps.uni-saarland.de/autosubst/`

[12] Zhang, X.: Using LEO-II to prove properties of an explicit substitution M-set model. Bachelor's thesis, Saarland University (2008)