Brief Notes on the Category Theoretic Semantics of
Simply Typed Lambda Calculus

Andrew Pitts

Notation: comma-separated snoc lists

When presenting logical systems and type theories, it is common to write finite lists of things using a comma to indicate the cons-operation and with the head of the list at the right. With this convention there is no common notation for the empty list; we will use the symbol “⋄”. Thus ML-style list notation

\[ \text{nil} \ a :: \text{nil} \ b :: a :: \text{nil} \ \text{etc} \]

becomes

\[ \text{⋄} \ \text{⋄} , a \ \text{⋄} , a , b \ \text{etc} \]

For non-empty lists, it is very common to leave the initial part “⋄,” of the above notation implicit, for example just writing \( a, b \) instead of \( \text{⋄}, a, b \).

Write \( X^* \) for the set of such finite lists with elements from the set \( X \).

1 Syntax of the simply typed \( \lambda \)-calculus

Fix a countably infinite set \( \mathbb{V} \) whose elements are called variables and are typically written \( x, y, z, \ldots \)

The simple types (with product types) \( A \) over a set \( Gnd \) of ground types are given by the following grammar, where \( G \) ranges over \( Gnd \):

\[
A ::= G \mid \text{unit} \mid A \times A \mid A \to A
\]

Write \( \text{ST}(Gnd) \) for the set of simple types over \( Gnd \).

The syntax trees \( t \) of the simply typed \( \lambda \)-calculus (STLC) over \( Gnd \) with constants drawn from a set \( \text{Con} \) are given by the following grammar, where \( c \) ranges over \( \text{Con} \), \( x \) over \( \mathbb{V} \) and \( A \) over \( \text{ST}(Gnd) \):

\[
t ::= c \mid x \mid () \mid (t, t) \mid fst \ t \mid snd \ t \mid \lambda x : A. t \mid tt
\]
We identify such syntax trees modulo remaining of λ-bound variables. More formally a simply typed λ-term is an equivalence class of syntax trees for the following, inductively defined relation of α-equivalence

\[ \begin{align*}
  c &=_a c \\
  x &= a x \\
  () &= () \\
  (t_1, t_2) &= (t_1', t_2') \\
  \text{fst } t &= a \text{fst } t' \\
  t &= a t' \\
  t_1 &= a t_1' \\
  t_2 &= a t_2' \\
  t_1 t_2 &= a t_1' t_2' \\
  (y x) \cdot t &= a (y x') \cdot t' \\
  \lambda x : A. t &= a \lambda x' : A. t' \\
\end{align*} \]

In the last rule \((y x) \cdot t\) indicates the syntax tree obtained from \(t\) by swapping occurrences of \(y\) and \(x\); given the condition that \(y\) does not occur in \(t\), this is the same as replacing all occurrences of \(x\) in \(t\) by \(y\). Thus the last rule says that \(\lambda x : A. t\) and \(\lambda x' : A. t'\) are α-equivalent if \(t\) and \(t'\) become α-equivalent once we replace all occurrences of \(x\) in \(t\) and all occurrences of \(x'\) in \(t'\) by some common “fresh” variable \(y\).

It is conventional to not make a notational distinction between a tree \(t\) and the α-equivalence class that it determines. That convention can be made mathematically precise via the use of nominal sets; see for example Pitts [2013, Chapter 8]. An alternative to working with λ-terms as α-equivalence classes of abstract syntax trees is to use a nameless representation due to de Bruijn [1972] instead of explicitly named bound variables. For typed λ-calculi, especially when using systems like Agda [wiki.portal.chalmers.se/agda/agda.php] or Coq [coq.inria.fr], so-called well-scoped de Bruijn indices are very convenient (if not very human-readable); see for example Keller and Altenkirch [2010, Section 2].

\section{Typing relation}

We assume that the set \(\text{Con}\) comes with a function mapping each constant \(c \in \text{Con}\) to its type \(A \in ST(\text{Gnd})\). We sometimes write \(c\) as \(c^A\) to indicate that \(A\) is its type.

In order to extend this typing function from constants to compound simply typed λ-terms we have to assign types to (free) variables. We do so via typing environments \(\Gamma\):

\[ \Gamma ::= \diamond | \Gamma, x : A \quad (\text{where } x \in V, A \in ST(\text{Gnd})) \]

Thus the set of typing environments is in bijection with \((V \times ST(\text{Gnd}))^*\), the set of finite lists of (variable,type)-pairs. The domain \(\text{dom } \Gamma\) of a typing environment \(\Gamma\) is the finite set of variables occurring in it:

\[ \begin{align*}
  \text{dom} \diamond &= \emptyset \\
  \text{dom}(\Gamma, x : A) &= \text{dom } \Gamma \cup \{x\} \\
\end{align*} \]

We only use the \(\Gamma\) that are well-formed \(\Gamma \text{ok}\) in the sense that no variable occurs more than once in the list:

\[ \begin{align*}
  \diamond \text{ok} \\
  \Gamma \text{ok} \quad x \notin \text{dom } \Gamma \\
  \Gamma, x : A \text{ ok} \\
\end{align*} \]
Then the typing relation $\Gamma \vdash t : A$ for assigning types $A$ to terms $t$ in a given typing environment $\Gamma$ is inductively defined by:

- $\Gamma \vdash x : A \quad \var{x \notin \text{dom} \Gamma}$ (VAR)
- $\Gamma \vdash x' : A' \quad \var{x' \notin \text{dom} \Gamma}$ (VAR')
- $\Gamma \vdash c^A : A \quad \text{(CONST)}$
- $\Gamma \vdash () : \text{unit} \quad \text{(UNIT)}$
- $\Gamma \vdash t : A \quad \Gamma \vdash t' : A' \quad \Gamma \vdash (t, t') : A \times A' \quad \text{(PAIR)}$
- $\Gamma \vdash \text{fst} t : A \quad \Gamma \vdash \text{snd} t : A' \quad \text{(SND)}$
- $\Gamma \vdash \lambda x : A. t : A \to A' \quad \text{(}\lambda\text{)}$
- $\Gamma \vdash t t' : A' \quad \Gamma \vdash t' : A \quad \text{(APP)}$

Here are some simple properties of the typing relation $\Gamma \vdash t : A$, proved by induction on its derivation. The second property makes use of the finite set $\text{fv} t$ of free variables of a term $t$, which is well-defined by:

- $\text{fv} c = \text{fv} () = \emptyset$
- $\text{fv} x = \{x\}$
- $\text{fv} (t, t') = \text{fv} t t' = \text{fv} t \cup \text{fv} t'$
- $\text{fv} \lambda x : A. t = \{x' \in \text{fv} t \mid x' \neq x\}$

**Lemma 2.1.**

1. If $\Gamma \vdash t : A$, then $\Gamma \text{ ok}$.
2. If $\Gamma \vdash t : A$, then $\text{fv} t \subseteq \text{dom} \Gamma$.
3. If $\Gamma \vdash t : A$ and $\Gamma \vdash t : A'$, then $A = A'$.

Property 3 says that terms have at most one type in any (well-formed) typing environment. Of course some terms have no type; for example $\Diamond \vdash () () : A$ is not derivable from the rules for any type $A$ (why?).

Because we have formulated typing environments as ordered lists (rather than, say, finite maps from variables to types), the important property of the typing relation that it is preserved under weakening typing environments (that is, adding extra (variable, type)-pairs while preserving the property of being well-formed) has to be formulated carefully. Here is a particular inductive definition of a weakening relation $[w : \Gamma' \triangleright \Gamma]$ (where $w ::= t \mid w \pi \mid w x$), inspired by Chapman [2009, Section 4.5], that interacts well with the typing relation:

- $\Gamma \text{ ok} \quad w : \Gamma' \triangleright \Gamma \quad x \notin \text{dom} \Gamma' \quad w : \Gamma' \triangleright \Gamma \quad x \notin \text{dom} \Gamma'$
- $\Gamma \vdash \pi : (\Gamma', x : A) \triangleright \Gamma$
- $\Gamma \vdash x : (\Gamma', x : A) \triangleright \Gamma, x : A$

**Lemma 2.2.**

1. If $w : \Gamma' \triangleright \Gamma$ and $\Gamma \text{ ok}$, then $\Gamma' \text{ ok}$.
2. If $\Gamma \vdash t : A$ and $w : \Gamma' \triangleright \Gamma$, then $\Gamma' \vdash t : A$. 


Proof. Property 1 is proved by induction on the derivation of \( w : \Gamma' \triangleright \Gamma \).

For property 2, which is the desired weakening property of the typing relation, one proceeds by induction on the derivation of \( \Gamma \vdash t : A \). For the base case when \( t \) is a variable, one proves

\[
\Gamma \vdash x : A \quad \text{and} \quad w : \Gamma' \triangleright \Gamma \quad \text{implies} \quad \Gamma' \vdash x : A
\]

by induction on the derivation of \( w : \Gamma' \triangleright \Gamma \), using part 1; for the induction step when \( t \) is a \( \lambda \)-abstraction one uses the fact that \( \lambda \)-terms are \( \alpha \)-equivalence classes of syntax trees, so that a representative \( \lambda \)-bound variable can chosen to not be in \( \text{dom} \, \Gamma' \), allowing the third rule for the \( w : \Gamma' \triangleright \Gamma \) relation to be applied. 

\[
\square
\]

3 Cartesian closed categories

Recall that a category \( C \) is \textbf{cartesian closed} if it has

\textbf{A terminal object:} a \( C \)-object \( \top \) with the property that for every \( Z \in \text{obj} \, C \) there is a unique morphism \( \langle \rangle \in C(Z, \top) \). The uniqueness part of this property is:

\[
f \in C(Z, \top) \Rightarrow f = \langle \rangle
\]

\textbf{Binary products:} for all \( X, Y \in \text{obj} \, C \) there is a \( C \)-object \( X \times Y \) and morphisms \( \pi_1 \in C(X \times Y, X), \pi_2 \in C(X \times Y, Y) \) with the property that for every \( Z \in \text{obj} \, C, f \in C(Z, X) \) and \( g \in C(Z, Y) \), there is a unique morphism \( \langle f, g \rangle \in C(Z, X \times Y) \) satisfying \( \pi_1 \circ \langle f, g \rangle = f \) and \( \pi_2 \circ \langle f, g \rangle = g \). The uniqueness part of this property is equivalent to requiring:

\[
h \in C(Z, X \times Y) \Rightarrow h = \langle \pi_1 \circ h, \pi_2 \circ h \rangle
\]

As a matter of notation, if \( f \in C(Z, X) \) and \( g \in C(W, Y) \) we define \( f \times g \in C(Z \times W, X \times Y) \) to be \( f \times g \triangleq \langle f \circ \pi_1, g \circ \pi_2 \rangle \).

\textbf{Exponentials:} for all \( X, Y \in \text{obj} \, C \) there is a \( C \)-object \( Y^X \) and a morphism \( \text{app} \in C(Y^X \times X, Y) \) with the property that for every \( Z \in \text{obj} \, C \) and \( f \in C(Z \times X, Y) \) there is a unique morphism \( \text{cur} \, f \in C(Z, Y^X) \) satisfying \( \text{app} \circ (\text{cur} \, f \times \text{id}_X) = f \). The uniqueness part of this property is equivalent to requiring:

\[
h \in C(Z, Y^X) \Rightarrow h = \text{cur}(\text{app} \circ (h \times \text{id}_X))
\]

4 Semantics in a cartesian closed category

Let \( C \) be a cartesian closed category. Any function \( M : \text{Gnd} \to \text{obj} \, C \) assigning \( C \)-objects to ground types can be extended to a function mapping types \( A \in \text{ST}(\text{Gnd}) \) to objects
$M[A] \in \text{obj } C$, by recursion over the structure of $A$:

- $M[G] = M(G)$
- $M[\text{unit}] = 1$ (terminal object in $C$)
- $M[A \times A'] = M[A] \times M[A']$ (product in $C$)
- $M[A \to A'] = M[A']^{M[A]}$ (exponential in $C$)

Typing environments also denote $C$-objects, by recursion over the length of the list $\Gamma$:

- $M[\emptyset] = 1$
- $M[\Gamma, x : A] = M[\Gamma] \times M[A]$ 

Finally, if in addition to $M : \text{Gnd} \to \text{obj } C$ we also have a function assigning to each constant $c \in \text{Con}$, of type $A$ say, a global section\(^1\) $M(c) \in C(1, M[A])$, then for each derivable instance of the typing relation $\Gamma \vdash t : A$ we define a $C$-morphism $M[\Gamma \vdash t : A] \in C(M[\Gamma], M[A])$

as follows:

- $M[\Gamma, x : A \vdash x : A] = M[\Gamma] \times M[A] \xrightarrow{\pi_2} M[A]$
- $M[\Gamma, x' : A' \vdash x : A] = M[\Gamma] \times M[A'] \xrightarrow{\pi_2} M[\Gamma] \xrightarrow{M[\Gamma \vdash x : A]} M[A]$ if $x' \notin \text{dom } \Gamma$
- $M[\Gamma \vdash c^A : A] = M[\Gamma] \xrightarrow{\langle \rangle} 1 \xrightarrow{M(c)} M[A]$
- $M[\Gamma \vdash () : \text{unit}] = M[\Gamma] \xrightarrow{\langle \rangle} 1$
- $M[\Gamma \vdash (t, t') : A \times A'] = M[\Gamma] \xrightarrow{\langle M[\Gamma \vdash t : A], M[\Gamma \vdash t' : A'] \rangle} M[A] \times M[A']$
- $M[\Gamma \vdash \text{fst } t : A] = M[\Gamma] \xrightarrow{M[\Gamma \vdash t : A \times A']} M[A] \times M[A'] \xrightarrow{\pi_2} M[A']$
- $M[\Gamma \vdash \text{snd } t : A'] = M[\Gamma] \xrightarrow{M[\Gamma \vdash t : A \times A']} M[A] \times M[A'] \xrightarrow{\pi_2} M[A']$
- $M[\Gamma \vdash \lambda x : A. t : A \to A'] = \text{cur} \left( M[\Gamma] \times M[A] \xrightarrow{M[\Gamma \vdash x : A], M[\Gamma \vdash t : A']} M[A'] \right)$
- $M[\Gamma \vdash t t' : A'] = M[\Gamma] \xrightarrow{\langle t, t' \rangle} M[A']^{M[A]} \times M[A] \xrightarrow{\text{app}} M[A']$

where $A'$ is the unique type for which $\Gamma \vdash t : A \times A'$ holds

and where $f = M[\Gamma \vdash t : A \to A']$ and $f' = M[\Gamma \vdash t' : A]$.

**Summary:** given an interpretation of ground types as objects of $C$ and constants as global sections of objects in $C$, we give meaning to simple types as $C$-objects and meaning to simply-typed $\lambda$ terms (in a given typing environment) as $C$-morphisms.

We will need the following property of this semantics with respect to weakening typing environments:

\(^1\)In a category $C$ with terminal object $1$, morphisms $f \in C(1, X)$ are called global sections of the $C$-object $X$. 

5
Lemma 4.1 (Semantics of weakening). For each instance of the weakening relation $w : \Gamma' \triangleright \Gamma$ we get a $C$-morphism

$M[w : \Gamma' \triangleright \Gamma] : M[\Gamma'] \rightarrow M[\Gamma]$

by defining:

$M[i : \Gamma \triangleright \Gamma] = M[\Gamma] \xrightarrow{id} M[\Gamma]$

$M[w \pi : (\Gamma', x : A) \triangleright \Gamma] = M[\Gamma'] \times M[A] \xrightarrow{\pi_1} M[\Gamma'] \xrightarrow{M[w : \Gamma' \triangleright \Gamma]} M[\Gamma]$

$M[w x : (\Gamma', x : A) \triangleright \Gamma, x : A] = M[\Gamma'] \times M[A] \xrightarrow{M[w : \Gamma' \triangleright \Gamma] \times id} M[\Gamma] \times M[A]$

If $w : \Gamma' \triangleright \Gamma$ holds, then for all derivable $\Gamma \vdash t : A$, the meaning of $\Gamma' \vdash t : A$ (valid by Lemma 2.2(2)) in $C$ is the morphism $M[\Gamma'] \rightarrow M[A]$ equal to the morphism given by composing $M[w : \Gamma' \triangleright \Gamma]$ with $M[\Gamma \vdash t : A]$.

Proof. By induction on the derivation of $\Gamma \vdash t : A$, following the proof of Lemma 2.2(2). For the induction step for $\lambda$-abstractions, one uses the fact that in a cartesian closed category the Currying operation satisfies $\text{cur}(f \circ (g \times \text{id})) = (\text{cur } f) \circ g$. □

When $M$ is understood from the context one sometimes just writes $[A]$ for $M[A]$ and similarly for $[\Gamma]$ and $[\Gamma \vdash t : A]$. Also, since the type $A$ in $\Gamma \vdash t : A$ is uniquely determined (Lemma 2.1(3)), it is common to just write $[\Gamma \vdash t]$ for $[\Gamma \vdash t : A]$.

If $\Gamma \vdash t : A$ and $\Gamma \vdash t' : A$, then a typed equation

$\Gamma \vdash t = t' : A$

is satisfied by this semantics if $M[\Gamma \vdash t : A]$ and $M[\Gamma \vdash t' : A]$ are equal morphisms from $M[\Gamma]$ to $M[A]$ in $C$. It is natural to ask which typed equations are always satisfied, whatever the ccc $C$. This turns out to be the notion of $\beta\eta$-equality given in Section 6. To describe it we first have to define (capture-avoiding) substitution of terms for free variables and its semantics.

5 Substitution

Substitutions $\sigma$ are finite lists of (variable, term)-pairs, written with the following notation:

$\sigma ::= \emptyset | \sigma, x := t$

The domain $\text{dom } \sigma$ of a substitution is given by

$\text{dom } \emptyset = \emptyset$

$\text{dom}(\sigma, x := t) = \text{dom } \sigma \cup \{x\}$

and its set of free variables $\text{fv } \sigma$ by

$\text{fv } \emptyset = \emptyset$

$\text{fv}(\sigma, x := t) = \text{fv } \sigma \cup \text{fv } t$
Write $x \not# \sigma$ to mean that $x \notin \text{dom } \sigma \cup \text{fv } \sigma$.

Then the simply-typed $\lambda$-term $t[\sigma]$ resulting from applying the substitution $\sigma$ to the simply-typed $\lambda$-term $t$ is well-defined by:

\[
\begin{align*}
x[\circ] &= x \\
x[\sigma, x := t] &= t \\
x[\sigma, x' := t] &= x[\sigma] & \text{if } x \neq x' \\
c[\sigma] &= c \\
(t, t')[\sigma] &= (t[\sigma], t'[\sigma]) \\
\text{fst } t[\sigma] &= \text{fst } (t[\sigma]) \\
\text{snd } t[\sigma] &= \text{snd } (t[\sigma]) \\
(\lambda x : A. t)[\sigma] &= \lambda x : A. (t[\sigma]) & \text{if } x \not# \sigma \\
(t t')[\sigma] &= (t[\sigma])(t'[\sigma])
\end{align*}
\]

Recall that simply-typed $\lambda$-terms are $\alpha$-equivalence classes of syntax trees. One has to check that not only does the above definition respect $\alpha$-equivalence, but also it gives a totally defined function; it does so because in the penultimate clause, modulo $\alpha$-equivalence we can always choose the $\lambda$-bound variable $x$ so that $x \not# \sigma$ holds.

Note that $t[\circ, x_1 := t_1, \ldots, x_n := t_n]$ is a simultaneous substitution of $t_i$ for free occurrences of $x_i$ in $t$ for all $i = 1, \ldots, n$ and that may be different from an iterated single-substitution.

For example $x[\circ, x := y, y := z] = y$, whereas $(x[\circ, x := y])[\circ, y := z] = z$. We write $t'[t/x]$ for the single-substitution.

The relation $\Gamma' \vdash \sigma : \Gamma$ that $\sigma$ is a well-formed substitution between the typing environments $\Gamma'$ and $\Gamma$ is inductively defined by:

\[
\begin{align*}
\Gamma' \text{ ok} \quad & \quad \Gamma' \vdash \circ : \circ \\
\Gamma' \vdash \sigma : \Gamma \\
x \notin \text{dom } \Gamma \\
\Gamma' \vdash t : A \quad & \quad \Gamma' \vdash \sigma, x := t : (\Gamma, x : A)
\end{align*}
\]

Here are some simple properties of this relation that we need, and that can be proved by induction on its derivation:

**Lemma 5.1.** If $\Gamma' \vdash \sigma : \Gamma$, then

1. $\Gamma$ ok and $\Gamma'$ ok
2. $w : \Gamma'' \triangleright \Gamma'$ implies $\Gamma'' \vdash \sigma : \Gamma$
3. $x \notin \text{dom } \Gamma \cup \text{dom } \Gamma'$ implies $\Gamma', x : A \vdash (\sigma, x := x) : (\Gamma, x : A)$

**Lemma 5.2.** If $\Gamma \vdash t : A$ and $\Gamma' \vdash \sigma : \Gamma$, then $\Gamma' \vdash t[\sigma] : A$.

**Proof.** By induction on the derivation of $\Gamma \vdash t : A$. The induction step for $\lambda$-abstractions uses Lemma 5.1(3) together with the easily proved property of substitution that $x \not# \sigma$ implies $x[\sigma] = x$ and $t[\sigma, x := x] = t[\sigma]$. 

\[\square\]
Given a function \( M \) mapping ground types and constants to objects and global sections in a ccc \( C \), we can interpret substitutions \( \Gamma' \vdash \sigma : \Gamma \) as morphisms like so:

\[
M[\Gamma'] \vdash \sigma : \Gamma \\
\Rightarrow M[\Gamma'] \rightarrow 1
\]

\[
M[\Gamma'] \vdash (\sigma, x := t) : (\Gamma, x : A) \\
\Rightarrow M[\Gamma'] \langle M[\Gamma'] \vdash \sigma : \Gamma, M[\Gamma'] \vdash t : A \rangle \\
\rightarrow M[\Gamma] \times M[A]
\]

**Lemma 5.3.** If \( \Gamma' \vdash \sigma : \Gamma \) and \( x \notin \text{dom} \Gamma \cup \text{dom} \Gamma' \), then the meaning of \( \Gamma', x : A \vdash (\sigma, x := x) : (\Gamma, x : A) \) (which is valid by Lemma 5.1(3)) is \( M[\Gamma'] \vdash \sigma : \Gamma \times \text{id} : M[\Gamma'] \times M[A] \rightarrow M[\Gamma] \times M[A] \).

**Proof.** By the definition of \( M[\Gamma', x : A \vdash (\sigma, x := x) : (\Gamma, x : A)] \), Lemma 4.1 and the fact that in a cartesian category one always has \( f \times \text{id} = \langle f \circ \pi_1, \pi_2 \rangle \).

**Theorem 5.4 (Semantics of simultaneous substitution).** If \( \Gamma \vdash t : A \) and \( \Gamma' \vdash \sigma : \Gamma \), then the following diagram commutes in \( C \):

\[
\begin{array}{ccc}
M[\Gamma'] & \xrightarrow{M[\Gamma'] \vdash \sigma : \Gamma} & M[\Gamma] \\
\downarrow{M[\Gamma'] \vdash t : A} & & \downarrow{M[\Gamma] \vdash t : A} \\
M[A] & & M[A]
\end{array}
\]

**Proof.** By induction on the derivation of \( \Gamma \vdash t : A \). For the induction step for \( \lambda \)-abstractions one uses Lemma 5.3 and the fact that in a cartesian closed category the Currying operation satisfies \( \text{cur}(f \circ (g \times \text{id})) = (\text{cur} f) \circ g \).

**Lemma 5.5 (Identity substitution).** For each typing environment \( \Gamma \), define the substitution \( \text{id}_\Gamma \) by:

\[
\text{id}_\varnothing = \varnothing \\
\text{id}_{\Gamma, x : A} = (\text{id}_\Gamma, x := x)
\]

1. If \( \Gamma \) ok, then \( \Gamma \vdash \text{id}_\Gamma : \Gamma \).

2. If \( \Gamma \vdash t : A \) and \( \Gamma, x : A \vdash t' : A' \), then

\[
\Gamma \vdash (\text{id}_\Gamma, x := t) : (\Gamma, x : A), \\
t'[t/x] = t'[\text{id}_\Gamma, x := t]
\]

and

3. \( M[\Gamma \vdash \text{id}_\Gamma : \Gamma] \) is equal to the identity morphism on \( M[\Gamma] \).

**Proof.** By induction on the derivation of \( \Gamma \) ok, using Lemma 5.2 for part (2).
Corollary 5.6 (Semantics of single substitution). If \( \Gamma \vdash t : A \) and \( \Gamma, x : A \vdash t' : A' \), then the following diagram commutes in \( C \):

\[
\begin{array}{c}
\text{M}[\Gamma] \\
\downarrow \quad \downarrow
\end{array}
\begin{array}{c}
\text{M}[\Gamma] \times \text{M}[A]
\end{array}
\begin{array}{c}
\text{M}[\Gamma, x : A' \vdash t' : A']
\end{array}
\begin{array}{c}
\text{M}[A']
\end{array}
\]

Proof. The result is a special case of Theorem 5.4 for the simultaneous substitution \( \Gamma \vdash (\text{id}_\Gamma, x := t) : (\Gamma, x : A) \), using Lemma 5.5.

6 \( \beta \eta \)-Equality of simply-typed \( \lambda \)-terms

The relation \( \Gamma \vdash t =_{\beta \eta} t' : A \) is inductively defined by the following rules:

**equivalence relation**

\[
\begin{align*}
\Gamma \vdash t : A & \quad \Gamma \vdash t_1 =_{\beta \eta} t_2 : A \\
\Gamma \vdash t =_{\beta \eta} t : A & \quad \Gamma \vdash t_2 =_{\beta \eta} t_1 : A \\
\Gamma \vdash t_1 =_{\beta \eta} t_2 : A & \quad \Gamma \vdash t_2 =_{\beta \eta} t_3 : A \\
\end{align*}
\]

**\( \beta \)-conversions**

\[
\begin{align*}
\Gamma, x : A & \vdash t : A' \\
\Gamma \vdash (\lambda x : A. t) t' =_{\beta \eta} t [t'/x] : A' \\
\Gamma \vdash t : A & \quad \Gamma \vdash t' : A' \\
\Gamma \vdash \text{fst} (t, t') =_{\beta \eta} t : A \\
\Gamma \vdash t : A & \quad \Gamma \vdash t' : A' \\
\Gamma \vdash \text{snd} (t, t') =_{\beta \eta} t' : A'
\end{align*}
\]

**\( \eta \)-conversions**

\[
\begin{align*}
\Gamma \vdash t : A \rightarrow A' & \quad x \notin \text{fv} t \\
\Gamma \vdash t =_{\beta \eta} \lambda x : A. (t x) : A \rightarrow A'
\end{align*}
\]

**congruence rules**

\[
\begin{align*}
\Gamma \vdash t_1 =_{\beta \eta} t_2 : A & \quad \Gamma \vdash t' =_{\beta \eta} t'_2 : A' \\
\Gamma \vdash (t_1, t'_2) =_{\beta \eta} (t_2, t'_2) : A \times A' \\
\Gamma \vdash t_1 =_{\beta \eta} t_2 : A \times A' \\
\Gamma \vdash \text{fst} t_1 =_{\beta \eta} \text{fst} t_2 : A
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash t_1 =_{\beta \eta} t_2 : A \times A' \\
\Gamma \vdash \text{snd} t_1 =_{\beta \eta} \text{snd} t_2 : A' \\
\Gamma \vdash \lambda x : A. t_1 =_{\beta \eta} \lambda x : A. t_2 : A \rightarrow A' \\
\Gamma \vdash \text{fst} t_1 =_{\beta \eta} \text{fst} t_2 : A
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash t_1 =_{\beta \eta} t_2 : A \rightarrow A' \\
\Gamma \vdash t'_1 =_{\beta \eta} t'_2 : A \\
\Gamma \vdash t_1 t'_1 =_{\beta \eta} t_2 t'_2 : A'
\end{align*}
\]
Lemma 6.1. If $\Gamma \vdash t =_{\beta\eta} t' : A$, then $\Gamma \vdash t : A$ and $\Gamma \vdash t' : A$.

Proof. By induction on the derivation of $\Gamma \vdash t =_{\beta\eta} t' : A$, using Lemma 5.2 for the first $\beta$-conversion rule and Lemma 2.2(2) for first $\eta$-conversion rule.

Theorem 6.2 (Soundness). For any function $M$ mapping ground types and constants to objects and global sections in a cartesian closed category $C$, the associated semantics of types and terms (Section 4) satisfies that if $\Gamma \vdash t =_{\beta\eta} t' : A$ is derivable, then $M[\Gamma \vdash t : A]$ and $M[\Gamma \vdash t' : A]$ are equal morphisms in $C(M[\Gamma], M[A])$.

Proof. One has to check that the relation $\Gamma \vdash t : A$ and $\Gamma \vdash t' : A$ and $M[\Gamma \vdash t : A] = M[\Gamma \vdash t' : A]$ is closed under the above rules inductively generating the relation $\beta\eta$-equality relation. Here is the argument for the $\beta$-conversion involving $\lambda$-abstraction

$$\frac{\Gamma, x : A \vdash t : A' \quad \Gamma \vdash t' : A}{\Gamma \vdash (\lambda x : A. t) t' =_{\beta\eta} t'[x/x] : A'}$$

Given $\Gamma, x : A \vdash t : A'$ and $\Gamma \vdash t' : A$, define

$$X = M[A]$$
$$Y = M[\Gamma]$$
$$Z = M[A']$$
$$f = M[\Gamma, x : A \vdash t : A']$$
$$g = M[\Gamma \vdash t' : A]$$

Thus $f : Y \times X \rightarrow Z$ and $g : Y \rightarrow X$ in the ccc $C$ and

$$M[\Gamma \vdash (\lambda x : A. t) t' : A'] = \text{app} \circ (\text{cur} f, g) : Y \rightarrow Z$$
(by definition of the semantics of terms)

$$M[\Gamma \vdash t'[x/x] : A'] = f \circ (\text{id}_Y, g) : Y \rightarrow Z$$
(by Corollary 5.6)

But in any ccc we have $\text{app} \circ (\text{cur} f, g) = \text{app} \circ (\text{cur} f \times \text{id}_X) \circ (\text{id}_Y, g) = f \circ (\text{id}_Y, g)$. Therefore $M[\Gamma \vdash (\lambda x : A. t) t' : A'] = M[\Gamma \vdash t'[x/x] : A']$, as required.

Here is the argument for the $\eta$-conversion involving $\lambda$-abstraction

$$\frac{\Gamma \vdash t : A \rightarrow A' \quad x \notin \text{fv } t}{\Gamma \vdash t =_{\beta\eta} \lambda x : A. (t x) : A \rightarrow A}$$

Given $\Gamma \vdash t : A \rightarrow A'$ and $x \notin \text{fv}(t)$, without loss of generality we may assume also that $x \notin \text{dom } \Gamma$ (since $\lambda x : A. (t x) =_{\alpha} \lambda x' : A. (t x')$ for any $x' \notin \text{fv } t \cup \text{dom } \Gamma$). Define

$$X = M[A]$$
$$Y = M[\Gamma]$$
$$Z = M[A']$$
$$h = M[\Gamma \vdash t : A \rightarrow A']$$
Thus \( h : Y \to Z^X \) in \( C \) and
\[
M[\Gamma, x : A \vdash t : A \to A'] = h \circ \pi_1 : Y \times X \to Z^X
\]
(by Lemma 4.1)
\[
M[\Gamma, x : A \vdash x : A] = \pi_2 : Y \times X \to X
\]
(by definition of the semantics of terms)

Hence \( M[\Gamma \vdash \lambda x : A.(t x) : A \to A'] = \text{cur}(\text{app} \circ \langle h \circ \pi_1, \pi_2 \rangle) \). But in any ccc we have \( \text{cur}(\text{app} \circ \langle h \circ \pi_1, \pi_2 \rangle) = \text{cur}(\text{app} \circ (h \times \text{id}_X)) = h \) and therefore \( M[\Gamma \vdash t : A \to A'] = M[\Gamma \vdash \lambda x : A.(t x) : A \to A'] \), as required.

We leave checking closure under the other rules of \( \beta\eta \)-equivalence as an exercise. \( \square \)

7 The internal language of a cartesian closed category

Given a particular cartesian closed category \( C \), we can take \( \text{obj} \ C \) to be the set of ground types and take each global element \( f \in C(1, X) \) (for any \( C \)-object \( X \)) to be a constant of type \( X \). Taking the interpretation \( M \) to be the identity function, then the simple types and the simply typed \( \lambda \)-terms over this collection of ground types and constants provides a convenient language for describing the objects and morphisms of \( C \) and their (equational) properties.

For example if \( X, Y \) and \( Z \) are three objects in a ccc \( C \), then there is always an isomorphism
\[
Z^{X \times Y} \cong (Z^Y)^X
\]

One can construct the morphisms that constitute this isomorphism and prove they are mutually inverse only using the universal properties of products and exponentials in \( C \). However, the internal language allows us describe the morphisms and prove that they are inverse via properties of \( \beta\eta \)-equivalence; furthermore these descriptions look like what one expect when \( C \) is the category of sets and functions:

\[
s \triangleq \lambda f : (X \times Y) \to Z. \lambda x : X. \lambda y : Y. f(x, y)
\]
\[
t \triangleq \lambda g : X \to (Y \to Z). \lambda z : X \times Y. g(\text{fst} z)(\text{snd} z)
\]
satisfy
\[
\circ \vdash s : ((X \times Y) \to Z) \to (X \to (Y \to Z))
\]
\[
\circ \vdash t : (X \to (Y \to Z)) \to ((X \times Y) \to Z)
\]
\[
\circ, f : (X \times Y) \to Z \vdash t(s f) = \beta\eta f : (X \times Y) \to Z
\]
\[
\circ, g : X \to (Y \to Z) \vdash s(t g) = \beta\eta g : X \to (Y \to Z)
\]

8 Free cartesian closed categories

Theorem 6.2 has a converse – a completeness theorem: given \( \Gamma \vdash t : A \) and \( \Gamma \vdash t' : A \), if \( M[\Gamma \vdash t : A] = M[\Gamma \vdash t' : A] \) holds for any interpretation \( M \) of the ground types and
constants in any ccc, then $\Gamma \vdash t =_{\beta\eta} t' : A$ is derivable. In fact for any set of ground types and constants, there is a particular freely generated ccc $F$ containing an interpretation $M$ of the ground types and constants satisfying

$$M[\Gamma \vdash t : A] = M[\Gamma \vdash t' : A] \iff \Gamma \vdash t =_{\beta\eta} t' : A \quad (1)$$

$F$ is constructed from the syntax of the simply typed $\lambda$-calculus quotiented by $\beta\eta$-equivalence. Specifically, one can take $\text{obj } F = \text{ST}(\text{Gnd})$. For two such objects $A, A' \in \text{ST}(\text{Gnd})$, we take $F(A, A')$ to be the quotient of the set $\{ t \mid \Diamond \vdash t : A \rightarrow A' \}$ of closed terms (i.e. those with no free variables) of type $A \rightarrow A'$ by the equivalence relation relating two such terms $t$ and $t'$ if $\Diamond \vdash t =_{\beta\eta} t' : A \rightarrow A'$. The identity morphism in $F$ on $A$ is the equivalence class of $\lambda x : A. x$. The composition of two morphisms represented by terms $\Diamond : t : A \rightarrow A'$ and $\Diamond : t' : A' \rightarrow A''$ is well-defined by taking the equivalence class of the term $\Diamond : \lambda x : A. t'(t x) : A \rightarrow A''$. One has to check that this recipe does give a category and that it is cartesian closed; unsurprisingly, the terminal object is unit, the product of objects $A, A' \in \text{ST}(\text{Gnd})$ is the simple type $A \times A'$ (equipped with the obvious projection morphisms) and their exponential is the simple type $A \rightarrow A'$ (equipped with the obvious application morphism).

Taking $M$ to map each ground type $G \in \text{Gnd}$ to $G \in \text{obj } F$ and each constant $c^A$ to the global element $M c \in F(\text{unit}, A)$ given by the equivalence class of the term $\Diamond : \lambda x : \text{unit}. c : \text{unit} \rightarrow A$, one can show that this interpretation has property (1).

$F$ is a free ccc in a similar sense to $\Sigma^n$ being the free monoid on a set $\Sigma$ – there is a universal property that characterises it, whose statement in terms of morphisms of cartesian closed categories is beyond the scope of these notes (see Crole [1993, Section 4.8]).

References


