CS522 - Programming Language Semantics

Simply Typed Lambda Calculus

Grigore Roşu

Department of Computer Science

University of Illinois at Urbana-Champaign

We now discuss a non-trivial extension of λ-calculus with *types*. The idea is that each variable binding is assigned a type, which will allow one to calculate a unique type for each well-formed λ-expression or λ-term.

As we know from our experience with programming languages, the addition of types will allow one to reject "programs" that are not well-typed, with the intuition that those programs are most likely wrong (with respect to what the programmer meant).

Typing comes at a price: sometimes correct programs are rejected. One can, of course, argue that those programs are not correct (by definition, because they do not type). All in all, practice has shown that typing is overall useful in programming languages. Simply typed λ-calculus is perhaps the simplest typed language.

Syntax

The BNF syntax of simply-typed λ-calculus is

$$Var ::= x \mid y \mid \ldots$$

$$Type ::= \circ \mid Type \rightarrow Type$$

$$Exp ::= Var \mid \lambda Var : Type.Exp \mid Exp\,Exp.$$

To keep the presentation simple, for the time being we assume only one constant type, $\circ$, and only one type constructor, $\rightarrow$. Thus $(\circ \rightarrow \circ) \rightarrow (\circ \rightarrow (\circ \rightarrow \circ))$ is a well-formed type. To simplify writing, we assume that $\rightarrow$ is *right-associative*; the type above can then be written $(\circ \rightarrow \circ) \rightarrow \circ \rightarrow \circ \rightarrow \circ$. As in the case of untyped λ-calculus, the λ-application is still assumed to be *left-associative*.

Exercise 1 *Define the syntax above in a Maude module, using the alternative mix-fix notation.*

Terms

Using the syntax above, one can naturally generate simply-typed terms or "programs", such as, for example, $\lambda x \colon \circ.\lambda f \colon \circ \to \circ.fx$. The intuition for this λ-abstraction is that it takes some λ-expressions of types $\circ$ and $\circ \to \circ$, respectively, and applies the latter on the former. Naturally, the type of the result is expected to be $\circ$.

Unlike in the case of untyped λ-calculus, the BNF (or, equivalently, the mix-fix notation) is not powerful enough to express all the intended well-formed, or better say well-typed, terms. Indeed, for example the term $\lambda x \colon \circ.xx$ does not make any sense, because x, in order to be applied on an expression of type $\circ$, in particular on itself, must have the type $\circ \to s$ for some type s; however, x is declared of type $\circ$. Moreover, one can formally show that $\lambda x \colon s.xx$ is not well-formed for any type s.

Even more, it can be shown that there is *no context free grammar* (CFG) whose language consists of all well-typed λ-expressions. This is perhaps the simplest language supporting the "folklore" claim that "programs do not form a context-free language".

Now the natural question is how to characterize, or how to "parse", simply-typed λ-expressions. There are three equivalent approaches to do this, all being easily adaptable to other typed frameworks. Let us first introduce some important notation.

A *type assignment* is a finite set $X = \{x_1 : s_1, \ldots, x_n : s_n\}$ of pairs $x : s$, where x is a variable and s a type, with the property that each variable occurs at most once in X: it is not possible to have $x : s_1, x : s_2 \in X$ for different types s_1, s_2. One of the main reasons for this limitation is that well-formed λ-expressions, including those which are just variables, are desired to have unique types. Then if

x occurs in X, written (admittedly ambiguously) $x \in X$, we may let $X(x)$ denote the type s such that $x\!:\!s \in X$. Often the curly brackets "{" and "}" are omitted from the notation of X and its elements are permuted conveniently; thus, if $x \notin X$, then $X, x\!:\!s$ is a type assignment containing $x\!:\!s$. We let *TypeAssignment* denote the set of type assignments.

For the time being, let us introduce the notation $X \, \triangleright \, E : t$, typically called a *type judgment*, or a *well-typed λ-expression* or *λ-term* (sometimes "well-typed" may be dropped if understood), with the intuition that under the type assignment X, the λ-expression E is well-typed and has the type t. For example, one can write $x\!:\!\circ, f\!:\!\circ \to \circ \, \triangleright \, fx\!:\!\circ$. We will shortly see three different ways to define this intuition precisely.

Alternative notations for $X \, \triangleright \, E\!:\!t$ could be $(\forall X)E\!:\!t$ or $(E\!:\!t).X$, or just $E\!:\!t$ when X is understood from context, or even simply E if both X and t are understood. Let us next discuss the three

different (but related) formal approaches to define this.

1. Proof system

We can define e *proof system* that can derive precisely the well-typed λ-expressions. The following three rules do this:

$$X, x\!:\!s \; \triangleright \; x\!:\!s \qquad\qquad \text{for any type assignment } X, x\!:\!s;$$

$$\frac{X[s/x] \; \triangleright \; E\!:\!t}{X \; \triangleright \; \lambda x\!:\!s.E\!:\!s \rightarrow t} \qquad \text{for any type assignment } X, \text{ any } \textit{Exp}\text{-term } E, \text{ and any type } t;$$

$$\frac{X \; \triangleright \; E\!:\!s \rightarrow t \qquad X \; \triangleright \; E'\!:\!s}{X \; \triangleright \; EE'\!:\!t} \qquad \text{for any type assignment } X, \textit{Exp}\text{-terms } E, E', \text{ and types } s, t.$$

As usual, $X \vartriangleright E\!:\!t$ is called *derivable* if there is some sequence $X_1 \vartriangleright E_1\!:\!t_1, \ldots, X_n \vartriangleright E_n\!:\!t_n$ such that $X_n \vartriangleright E_n\!:\!t_n$ is $X \vartriangleright E\!:\!t$ and each $X_i \vartriangleright E_i\!:\!t_i$ "follows" by one of the three rules above from previously $(< i)$ derived well-typed terms. We may write $\vdash X \vartriangleright E\!:\!t$, or sometimes even just $X \vartriangleright E\!:\!t$, whenever $X \vartriangleright E\!:\!t$ is derivable.

Exercise 2 *Derive* $x\!:\!\circ \vartriangleright \lambda f\!:\!\circ \to \circ.fx : (\circ \to \circ) \to \circ$. *Also, find the type "?" and derive* $\emptyset \vartriangleright \lambda x\!:\!\circ.\lambda f\!:\!\circ \to \circ \to \circ \to \circ.fxx : ?$

When X is empty, we write $E : t$ instead of $\emptyset \vartriangleright E : t$. We are not going to (re)define the notions of *free variable*, with the corresponding operator FV, and *substitution*. They have precisely the same meaning as in untyped λ-calculus.

Properties about well-typed λ-expressions are typically proved by induction on the length of derivation.

Proposition 1 *The following hold:*

- *If $X \rhd E:t$ then $FV(E) \subseteq X$;*

- *If $X \rhd \lambda x:s.E:s \to t$ then $X, y:s \rhd (E[x \leftarrow y]):t$ for any $y \notin X$;*

- *If $X \rhd E:s$ and $X \rhd E:t$ then $s = t$;*

- *If $X[s/x] \rhd E:t$ and $X \rhd E':s$ then $X \rhd E[x \leftarrow E']:t$;*

- *If X, X' are type assignments and E is a λ-expression such that for all $x \in FV(E)$, $x:s \in X$ iff $x:s \in X'$, then $X \rhd E:t$ iff $X' \rhd E:t$;*

- *$\lambda x:s.xx$ does not type.*

2. Typing Algorithm

We can also define a relatively trivial *typing algorithm* that takes a type assignment X together with a λ-expression E, and tries to calculate a type t for E. The algorithm traverses E recursively:

Algorithm $\mathcal{A}(X, E)$

- if E is x and $x : s \in X$ then return s;

- if E is $\lambda x : s.E'$ and $\mathcal{A}(X[s/x], E')$ returns t then return $t \to s$;

- if E is $E_1 E_2$ and $\mathcal{A}(X, E_1)$ returns $s \to t$ and $\mathcal{A}(X, E_2)$ returns s then return t;

- otherwise return error

Exercise 3 *Prove that $X \rhd E : t$ is derivable if and only if $\mathcal{A}(X, E)$ returns t. (Hint: By structural induction on E.)*

3. Sets of Terms

Let us next give another characterization of the well-typed λ-terms. We define the family of sets $\{\mathcal{T}_t(X)\}_{X \in TypeAssignment, t \in Type}$ as the (componentwise) smallest set $\{W_{X,t}\}_{X \in TypeAssignment, t \in Type}$ of words in the (CF) language of λ-calculus such that:

- $x \in W_{X,s}$ if $x : s \in X$

- $\lambda x : s.E \in W_{X,s \to t}$ if $E \in W_{X[s/x],t}$

- $E_1 E_2 \in W_{X,t}$ if $E_1 \in W_{X,s \to t}$ and $E_2 \in W_{X,s}$ for some $s \in Type$.

Exercise 4 *Prove that $X \rhd E : t$ is derivable iff $E \in \mathcal{T}_t(X)$. (Hint: By structural induction on E.)*

Equational Rules

We have discussed so far techniques to check that λ-expressions are well-typed. From now on we assume that all the λ-expressions that occur in any context are well-typed. More precisely, whenever we write $X \vartriangleright E:t$, we assume that E is well-typed under the type assignment X and that it has the type t. We now focus on equational properties of simply-typed λ-calculus. These equations play a dual role: on the one hand they give means to show "programs" equivalent, while on the other hand underlay the infrastructure necessary to define a canonical model of λ-calculus.

An *equation* is a 4-tuple consisting of a type assignment X, two λ-expressions E and E', and a type t, such that $X \vartriangleright E : t$ and $X \vartriangleright E' : t$. To simplify notation, we write such equations as $(\forall X)\ E =_t E'$, with the intuition that for any interpretation of the variables in X (i.e., any assignment of values of corresponding type to variables in X), the expressions E and E' evaluate to the same value, which has the expected type t.

A set of equations $\mathcal{E}$ is also called an *equational theory* (in λ-calculus). Given an equational theory $\mathcal{E}$ and an equation e, we call the syntactic construct $\mathcal{E} \vdash e$ an *equational judgment*. We next give a set of derivation rules for equational judgments:

(axiom) $\quad \mathcal{E} \vdash (\forall X)\ E =_t E' \qquad$ if $(\forall X)\ E =_t E'$ is in $\mathcal{E}$

(add) $\quad \dfrac{\mathcal{E} \vdash (\forall X)\ E =_t E'}{\mathcal{E} \vdash (\forall X, x : s)\ E =_t E'} \qquad$ if $x \notin X$

(reflexivity) $\quad \mathcal{E} \vdash (\forall X)\ E =_t E \qquad$ if $X \vartriangleright E : t$

(symmetry) $\quad \dfrac{\mathcal{E} \vdash (\forall X)\ E =_t E'}{\mathcal{E} \vdash (\forall X)\ E' =_t E}$

(transitivity) $\quad \dfrac{\mathcal{E} \vdash (\forall X)\ E =_t E' \quad \mathcal{E} \vdash (\forall X)\ E' =_t E''}{\mathcal{E} \vdash (\forall X)\ E =_t E''}$

$$\text{(application)} \quad \frac{\mathcal{E} \vdash (\forall X)\ E_1 =_{s \to t} E_1' \qquad \mathcal{E} \vdash (\forall X)\ E_2 =_s E_2'}{\mathcal{E} \vdash (\forall X)\ E_1 E_2 =_t E_1' E_2'}$$

$$(\xi) \quad \frac{\mathcal{E} \vdash (\forall X[s/x])\ E =_t E'}{\mathcal{E} \vdash (\forall X)\ \lambda x\!:\!s.E =_{s \to t} \lambda x\!:\!s.E'}$$

$$(\beta) \quad \mathcal{E} \vdash (\forall X)\ (\lambda x\!:\!s.E)E' =_t E[x \leftarrow E'] \qquad \begin{array}{l} \text{if } X[s/x] \rhd E\!:\!t \text{ and} \\ X \rhd E'\!:\!s \end{array}$$

$$(\eta) \quad \mathcal{E} \vdash (\forall X)\ \lambda x\!:\!s.Ex =_{s \to t} E \qquad \text{if } x \notin FV(E)$$

The rule (axiom) says that any equation already existing in $\mathcal{E}$ is derivable from $\mathcal{E}$. The rule (add) allows one to add typed variables to the type assignment of a derived equation; this is necessary for several technical reasons, such as, for example, to bring the two equations to the same type assignment in order to apply the (transitivity) or the (application) rule. The next four rules, (reflexivity), (symmetry), (transitivity), and (application), are self explanatory and are instances of general equational reasoning in algebraic specifications to the signature of λ-calculus; in particular, (application) is an instance of the congruence deduction rule. The rule (ξ) is "almost" an instance of the equational congruence rule to the λ-abstraction construct; note, however, that the type assignment needs to be changed appropriately. The last two rules, (β) and (η) are nothing but the equational rule and typed versions of the (β) and (η) equations from untyped λ-calculus.

Proposition 2 *If $\mathcal{E} \vdash (\forall X)\ E =_t E'$ then $X \rhd E\!:\!t$ and $X \rhd E'\!:\!t$.*

Proposition 3 *If E, E' are two λ-expressions and $x \notin X$ then $\mathcal{E} \vdash (\forall X)\ E =_t E'$ iff $\mathcal{E} \vdash (\forall X, x : s)\ E =_t E'$.*

Corollary 1 *If E, E' are λ-expressions and X, X' are type assignments such that $x\!:\!s \in X$ iff $x\!:\!s \in X'$ for any $x \in FV(EE')$, then $\mathcal{E} \vdash (\forall X)\ E =_t E'$ iff $\mathcal{E} \vdash (\forall X')\ E =_t E'$.*

Proposition 4 *If $\mathcal{E} \vdash (\forall X)\ E =_s E'$ and $Y[s/y] \rhd F : t$ such that $X \cup Y$ is a proper type assignment, then $\mathcal{E} \vdash (\forall X \cup Y)\ F[y \leftarrow E] =_t F[y \leftarrow E']$.*
(Proof hint: "Eliminate" the substitution by applying the rule (β) twice backwards.)

Models

1. A *Type-indexed set* $M = \{M_t\}_{t \in Type}$ is an infinite collection of sets, one for each type; there is no relationship required among the sets M_s, M_t, and $M_{s \to t}$ for any types s and t. Note that type assignments can be regarded as *Type*-indexed sets with only a finite number of sets non-empty. For example, the type assignment $\{x : s, y : s, z : t\}$ can be regarded as the *Type*-indexed set whose s-component is $\{x, y\}$, whose t-component is $\{z\}$, and whose other components are all empty.

For a given $X \in TypeAssignment$, we let $\mathcal{T}(X)$ denote the *Type*-indexed set $\{\mathcal{T}_t(X)\}_{t \in Type}$.

2. Given *Type*-indexed sets $M = \{M_t\}_{t \in Type}$ and $N = \{N_t\}_{t \in Type}$, a *Type-indexed function* $h : M \to N$ is a collection of functions $\{h_t : M_t \to N_t\}_{t \in Type}$ defined on the corresponding components of the *Type*-indexed sets, one for each type t. If X is a type assignment and M is a *Type*-indexed set, then we call the *Type*-indexed functions $\rho : X \to M$ *M-environments*. As usual, we let $[X \to M]$ denote the set of all M-environments over the assignment X. If $x{:}s \in X$ and $v \in M_s$, then we let $\rho[x \leftarrow v] : X \to M$ denote the M-environment ρ' with $\rho'(y) = \rho(y)$ for all $y \neq x$ and $\rho'(x) = v$.

A *pre-frame* or *pre-model* is a pair $(\{M_t\}_{t \in Type}, \{M^{s,t} : M_{s \to t} \times M_s \to M_t\}_{s,t \in Type})$ consisting of a *Type*-indexed set and a (*Type*$\times$*Type*)-indexed collection of functions, such that $M^{s,t}$ is *extensional* for any s, t: for any $f, g \in M_{s \to t}$, if $M^{s,t}(f, v) = M^{s,t}(g, v)$ for all $v \in M_s$, then $f = g$.

A pre-frame or pre-model
$\mathcal{M} = (\{M_t\}_{t \in \mathit{Type}}, \{M^{s,t} : M_{s \to t} \times M_s \to M_t\}_{s,t \in \mathit{Type}})$ is called a
frame or *model* of simply-typed λ-calculus iff there is a
Type-indexed mapping, say $M\_$, taking well-typed λ-expressions
$X \rhd E{:}t$ to mappings $M_{X \rhd E:t} : [X \to M] \to M_t$ with the
following properties for any M-environment $\rho : X \to M$:

1) $M_{X \rhd x:s}(\rho) = \rho(x : s) \in M_s$;

2) $M^{s,t}(M_{X \rhd \lambda x:s.E:s \to t}(\rho), v) = M_{X[s/x] \rhd E:t}(\rho[x \leftarrow v])$ for any
$v \in M_s$;

3) $M_{X \rhd E_1 E_2:t}(\rho) = M^{s,t}(M_{X \rhd E_1:s \to t}(\rho), M_{X \rhd E_2:s}(\rho))$.

When such a mapping exists, we say, by a slight language abuse,
that the model $\mathcal{M}$ *extends* the pre-model
$(\{M_t\}_{t \in \mathit{Type}}, \{M^{s,t} : M_{s \to t} \times M_s \to M_t\}_{s,t \in \mathit{Type}})$.

Exercise 5 *Show that there is* at most one extension *of any pre-model to a model.*
(Hint: by induction on λ-expressions, using extensionality.)

Therefore, if a pre-frame can be extended to a frame, than that extension is unique. Given a model $\mathcal{M}$ and an M-environment $\rho : X \to M$, we let $\rho^{\#} : \mathcal{T}(X) \to M$ denote the *Type*-indexed map defined as $\rho^{\#}(X \,\triangleright\, E\!:\!t) = M_{X \,\triangleright\, E\!:\!t}(\rho)$.

Definition 1 *A model $\mathcal{M}$* satisfies *an equation $(\forall X)\ E =_t E'$, written $\mathcal{M} \models (\forall X)\ E =_t E'$, iff $\rho^{\#}(X \,\triangleright\, E\!:\!t) = \rho^{\#}(X \,\triangleright\, E'\!:\!t)$ for any $\rho : X \to M$.*

Given a set of equations $\mathcal{E}$ and an equation e, we extend our satisfaction relation to $\mathcal{M} \models \mathcal{E}$ iff $\mathcal{M}$ satisfies all equations in $\mathcal{E}$, and $\mathcal{E} \models e$ iff for any model $\mathcal{M}$, if $\mathcal{M} \models \mathcal{E}$ then $\mathcal{M} \models e$.

Theorem *(Soundness) If $\mathcal{E} \vdash e$ then $\mathcal{E} \models e$.*

 Proof Sketch. By induction on the length of the derivation. All

one needs to prove is that each derivation rule is sound. For example, in the case of the (ξ) rule, we should show that if $\mathcal{E} \models (\forall X, x\!:\!s)\ E =_t E'$ then $\mathcal{E} \models (\forall X)\ \lambda x\!:\!s.E =_{s \to t} \lambda x\!:\!s.E'$. Let $\mathcal{M}$ be a model such that $\mathcal{M} \models \mathcal{E}$, and let $\rho : X \to M$ be an M-environment. Then note that

$$
\begin{aligned}
M^{s,t}(\rho^{\#}(X \ \triangleright\ \lambda x\!:\!s.E\!:\!s \to t), v) \ &=\ M_{X, x:s \ \triangleright\ E:t}(\rho[x \leftarrow v]) \\
&=\ M_{X, x:s \ \triangleright\ E':t}(\rho[x \leftarrow v]) \\
&=\ M^{s,t}(\rho^{\#}(X \ \triangleright\ \lambda x\!:\!s.E'\!:\!s \to t), v).
\end{aligned}
$$

Then by extensionality we get

$$
\rho^{\#}(X \ \triangleright\ \lambda x\!:\!s.E\!:\!s \to t) = \rho^{\#}(X \ \triangleright\ \lambda x\!:\!s.E'\!:\!s \to t).
$$

Exercise 6 *Show that all the equational inference rules of simply-typed λ-calculus are sound.*

Full Type Frame

We next define a special (but very important) type frame, or model, of λ-calculus, called the *full type frame*. It consists of the most intuitive interpretation of λ-expressions, namely as values and functions.

Let us fix a set T and let us define inductively the following set $\mathcal{HO}$ of sets of "high-order" functions starting with T:

- $T \in \mathcal{HO}$;

- $[A \to B] \in \mathcal{HO}$ whenever $A, B \in \mathcal{HO}$.

Recall that for any two sets A and B, $[A \to B]$ is the *set* of all functions of domain A and codomain B. In other words, $\mathcal{HO}$ is defined as the smallest set of sets that is closed under the

operations above; that is, it contains T and whenever it contains the sets A, B, it also contains the set of functions between them.

We can now define a unique function $[\![\_]\!] : \textit{Type} \to \mathcal{HO}$ with the property that $[\![\circ]\!] = T$ and $[\![s \to t]\!] = [\![[\![s]\!] \to [\![t]\!]]\!]$ for any $s, t \in \textit{Type}$. Note that this function actually organizes $\mathcal{HO}$ as a $\textit{Type}$-indexed set: $\mathcal{HO} = \{[\![t]\!]\}_{t \in \textit{Type}}$.

From now on we regard $\mathcal{HO}$ as a $\textit{Type}$-indexed set and organize it into a model of simply-typed λ-calculus. To make it a pre-model, let us define $\mathcal{HO}^{s,t} : [\![s \to t]\!] \times [\![s]\!] \to [\![t]\!]$ as expected: $\mathcal{HO}^{s,t}(f, x) = f(x)$ for any $s, t \in \textit{Type}$ and any $f : [\![s]\!] \to [\![t]\!]$ and $x \in [\![s]\!]$; note that x can be itself a function if s is a function type. One can immediately see that $\mathcal{HO}^{s,t}$ are extensional: indeed, if $f(x) = g(x)$ for any x then $f = g$ (by the definition of function equality). Therefore, $\mathcal{HO}$ is a pre-model.

To make $\mathcal{HO}$ a model, we need to define appropriate interpretations of well-typed λ-expressions. For simplicity, we use the same notation $[\![\_\,]\!]$ as for the interpretation of types. For a given $X \,\triangleright\, E\!:\!t$, we define $[\![X \,\triangleright\, E\!:\!t]\!] : [X \to \mathcal{HO}] \to [\![t]\!]$ by induction as follows:

- $[\![X, x\!:\!s \,\triangleright\, x\!:\!s]\!](\rho) \overset{def}{=} \rho(x\!:\!s) \in [\![s]\!]$ for any $\mathcal{HO}$-environment $\rho : X \to \mathcal{HO}$;

- $[\![X \,\triangleright\, \lambda x\!:\!s.E\!:\!s \to t]\!](\rho)(v) \overset{def}{=} [\![X, x\!:\!s \,\triangleright\, E\!:\!t]\!](\rho[x \leftarrow v])$ for any $\rho : X \to \mathcal{HO}$ and $v \in [\![s]\!]$;

- $[\![X \,\triangleright\, EE'\!:\!t]\!] \overset{def}{=} ([\![X \,\triangleright\, E\!:\!s \to t]\!](\rho))([\![X \,\triangleright\, EE'\!:\!s]\!](\rho))$ for any $\mathcal{HO}$-environment ρ.

Exercise 7 *Prove that $\mathcal{HO}$ defined above is a model of λ-calculus.*

$\mathcal{HO}$ is perhaps the most natural model of simply-typed λ-calculus, in which types are interpreted as sets of their corresponding values,

λ-abstractions as functions on appropriate domains and co-domains, and λ-applications as function applications.

Term model

Let us now fix a *Type*-indexed set $\mathcal{X} = \{\mathcal{X}_t\}_{t \in \textit{Type}}$ such that $\mathcal{X}_t$ is *infinite* for any $t \in \textit{Type}$ and $\mathcal{X}_s \cap \mathcal{X}_t = \emptyset$ for $s, t \in \textit{Type}$. From now on we consider only (well-typed) λ-experiments over variables in $\mathcal{X}$, i.e., of the form $X \vartriangleright E : t$ with $X \subseteq \mathcal{X}$.

Technically speaking, since $\mathcal{X}$ is a partition of $\bigcup_{t \in \textit{Type}} \mathcal{X}_t$, each variable is now tagged automatically with its type. This means that one can simply ignore the type assignment X when writing well-typed terms $X \vartriangleright E : t$. However, for uniformity in notation, we prefer to keep the assignments in the notation of terms; we can think of them as the variables over which the corresponding

λ-expression was *intended* to be defined. For example, the right-hand side in the equation $(\forall a\!:\!s, b\!:\!s)\ (\lambda x\!:\!s.\lambda y\!:\!s.x)ab =_s a$ was intended to be $(a\!:\!s, b\!:\!s)\ \triangleright\ a\!:\!s$ in order for the equation to make sense, even though $b\!:\!s$ is not necessary in the type assignment.

Given a set of equations $\mathcal{E}$, we define the *$\mathcal{E}$-equivalence class* of the a λ-expression $X\ \triangleright\ E\!:\!t$ as the set

$$[X \triangleright E\!:\!t]_{\mathcal{E}} \overset{def}{=} \{X' \triangleright E'\!:\!t \mid \text{ there is some } Y \text{ such that } \mathcal{E} \vdash (\forall Y)\ E =_t E'\}.$$

Proposition 5 $[X\ \triangleright\ E\!:\!t]_{\mathcal{E}} = [X'\ \triangleright\ E'\!:\!t]_{\mathcal{E}}$ *iff there is some Y such that $\mathcal{E} \vdash (\forall Y)\ E =_t E'$.*

We can now define a *Type*-indexed set $\mathcal{T}_{\mathcal{E}} = \{\mathcal{T}_{\mathcal{E},t}\}_{t \in \mathit{Type}}$, by letting $\mathcal{T}_{\mathcal{E},t}$ be the set $\{[X\ \triangleright\ E\!:\!t]_{\mathcal{E}} \mid X \subseteq \mathcal{X}\}$ for any $t \in \mathit{Type}$.

Further, we can extend $\mathcal{T}_{\mathcal{E}}$ to a pre-model, by defining functions

$$\mathcal{T}_{\mathcal{E}}^{s,t} : \mathcal{T}_{\mathcal{E},s \to t} \times \mathcal{T}_{\mathcal{E},s} \to \mathcal{T}_{\mathcal{E},t} \text{ for any types } s,t \in \textit{Type} \text{ as follows:}$$

$$\mathcal{T}_{\mathcal{E}}^{s,t}([X \,\triangleright\, E\!:\!s \to t]_{\mathcal{E}}, [Y \,\triangleright\, F\!:\!s]_{\mathcal{E}}) \stackrel{def}{=} [X \cup Y \,\triangleright\, EF\!:\!t]_{\mathcal{E}} \ .$$

Proposition 6 $\mathcal{T}_{\mathcal{E}}$ *is a pre-model.*

Proof. All we need to show is that $\mathcal{T}_{\mathcal{E}}^{s,t}$ is well-defined and extensional.

For well-definedness, we need to prove that if
$[X' \,\triangleright\, E'\!:\!s \to t]_{\mathcal{E}} = [X \,\triangleright\, E\!:\!s \to t]_{\mathcal{E}}$ and
$[Y' \,\triangleright\, F'\!:\!s]_{\mathcal{E}} = [Y \,\triangleright\, F\!:\!s]_{\mathcal{E}}$ then
$[X' \cup Y' \,\triangleright\, E'F'\!:\!t]_{\mathcal{E}} = [X \cup Y \,\triangleright\, EF\!:\!t]_{\mathcal{E}}$. Since there are some $\bar{X}$
and $\bar{Y}$ such that $\mathcal{E} \vdash (\forall \bar{X}) \ E =_{s \to t} E'$ and $\mathcal{E} \vdash (\forall \bar{Y}) \ F =_s F'$, by
using the rule (add) a finite number of times we can derive
$\mathcal{E} \vdash (\forall \bar{X} \cup \bar{Y}) \ E =_{s \to t} E'$ and $\mathcal{E} \vdash (\forall \bar{X} \cup \bar{Y}) \ F =_s F'$; then by
(application) we can derive $\mathcal{E} \vdash (\forall \bar{X} \cup \bar{Y}) \ EF =_t E'F'$. By
Proposition 5, it follows that

$$[X' \cup Y' \; \triangleright \; E'F' : t]_{\mathcal{E}} = [X \cup Y \; \triangleright \; EF : t]_{\mathcal{E}}.$$

For extensionality, we need to show that given $X \; \triangleright \; E : s \to t$ and $X' \; \triangleright \; E' : s \to t$ such that $[X \cup Y \; \triangleright \; EF : t]_{\mathcal{E}} = [X' \cup Y \; \triangleright \; E'F : t]_{\mathcal{E}}$ for any $Y \; \triangleright \; F : s$, it is indeed the case that $[X \; \triangleright \; E : t]_{\mathcal{E}} = [X' \; \triangleright \; E' : t]_{\mathcal{E}}$. Let us pick $Y = \{y : s\} \not\subseteq X \cup X'$ and $F = y$. Then $[X, y : s \; \triangleright \; Ey : t]_{\mathcal{E}} = [X', y : s \; \triangleright \; E'y : t]_{\mathcal{E}}$, so by Proposition 5, $\mathcal{E} \vdash (\forall Z) \; Ey =_t E'y$ for some $Z \subseteq \mathcal{X}$. Note that, in order for $Z \; \triangleright \; Ey : t$ and $Z \; \triangleright \; E'y : t$ to be well-typed, Z must contain the variable $y : s$. Let Z be $W, y : s$. By rule (ξ) we then derive $\mathcal{E} \vdash (\forall W) \; \lambda y : s.Ey =_{s \to t} \lambda y : s.E'y$. Finally, by applying the rule (η) twice we can derive $\mathcal{E} \vdash (\forall W) \; E =_{s \to t} E'$, which concludes our proof that $[X \; \triangleright \; E : t]_{\mathcal{E}} = [X' \; \triangleright \; E' : t]_{\mathcal{E}}$. Therefore, $\mathcal{T}_{\mathcal{E}}$ is a pre-model. $\qquad \square$

Our goal next is to organize $\mathcal{T}_{\mathcal{E}}$ as a model. To do it, we first need to define mappings $\mathcal{T}_{\mathcal{E}, X \triangleright E:t} : [X \to \mathcal{T}_{\mathcal{E}}] \to \mathcal{T}_{\mathcal{E},t}$ for all $X \triangleright E:t$. Note that $\mathcal{T}_{\mathcal{E}}$-environments map variables to $\mathcal{E}$-equivalence classes of λ-expressions. If $X = \{x_1:s_1, \ldots x_n:s_n\}$ and $\rho : X \to \mathcal{T}_{\mathcal{E}}$ is a $\mathcal{T}_{\mathcal{E}}$-environment taking x_i to, say $[X_i \triangleright E_i:s_i]_{\mathcal{E}}$, then we let $\mathcal{T}_{\mathcal{E}, X \triangleright E:t}(\rho)$ be defined as $[\bigcup_{i=1}^{n} X_i \triangleright E[x_1, \ldots, x_n \leftarrow E_1, \ldots E_n]]_{\mathcal{E}}$, where $E[x_1, \ldots, x_n \leftarrow E_1, \ldots E_n]$ is the term obtained by substituting $E_1, \ldots E_n$ for $x_1, \ldots x_n$ *in parallel*. One way to achieve this is to choose some fresh variables $z_1:s_1, \ldots, z_n:s_n$ in $\mathcal{X} \setminus (X \cup \bigcup_{i=1}^{n} X_i)$ and to let $E[x_1, \ldots, x_n \leftarrow E_1, \ldots E_n]$ be defined as $E[x_1 \leftarrow z_1] \ldots [x_n \leftarrow z_n][z_1 \leftarrow E_1] \ldots [z_n \leftarrow E_n]$.

Exercise 8 *Why would it* not *be correct to define parallel substitution as* $E[x_1 \leftarrow E_1] \ldots [x_n \leftarrow E_n]$?

Propositions 5 and 4 tell us that the maps $\mathcal{T}_{\mathcal{E}, X \triangleright E:t} : [X \to \mathcal{T}_{\mathcal{E}}] \to \mathcal{T}_{\mathcal{E},t}$ are indeed well defined.

Proposition 7 $\mathcal{T}_{\mathcal{E}}$ *is a model of simply-typed λ-calculus.*

Proof. We need to prove that the three conditions in the definition of a model hold.

1) Let $X, x\!:\!s$ be a type assignment and let $\rho : X, x\!:\!s \to \mathcal{T}_{\mathcal{E}}$ be a $\mathcal{T}_{\mathcal{E}}$-environment where $\rho(x_i\!:\!s_i) = [X_i \;\triangleright\; E_i\!:\!s_i]_{\mathcal{E}}$ for all $x_i\!:\!s_i \in X$ and $\rho(x\!:\!s) = [Y \;\triangleright\; F\!:\!s]$. Then $\mathcal{T}_{\mathcal{E},(X,x:s \,\triangleright\, x:s)}(\rho)$ is by definition $[Y \cup \bigcup_{i=1}^n X_i \;\triangleright\; F\!:\!s]_{\mathcal{E}}$, which is equal to $\rho(x\!:\!s)$ by Proposition 5, noting that $\mathcal{E} \vdash (\forall Y \cup \bigcup_{i=1}^n X_i)\, F =_s F$ by (reflexivity).

2) Let $X \;\triangleright\; \lambda x\!:\!s.E\!:\!s \to t$ be a well formed λ-expression, let $\rho : X \to \mathcal{T}_{\mathcal{E}}$ be a $\mathcal{T}_{\mathcal{E}}$-environment, and let $[Y \;\triangleright\; F\!:\!s]_{\mathcal{E}}$ be an $\mathcal{E}$-equivalence class in $\mathcal{T}_{\mathcal{E},s}$. We need to show that
$$\mathcal{T}_{\mathcal{E}}^{s,t}(\mathcal{T}_{\mathcal{E},(X \,\triangleright\, \lambda x:s.E:s \to t)}(\rho), [Y \;\triangleright\; F\!:\!s]_{\mathcal{E}}) =$$
$$\mathcal{T}_{\mathcal{E},(X,x:s \,\triangleright\, E:t)}(\rho[x \leftarrow [Y \;\triangleright\; F\!:\!s]_{\mathcal{E}}]).$$

If $\rho(x_i\!:\!s_i) = [X_i \; \triangleright \; E_i\!:\!s_i]_{\mathcal{E}}$ for each $x_i\!:\!s_i \in X$, then

$$\mathcal{T}_{\mathcal{E},(X \,\triangleright\, \lambda x:s.E:s\to t)}(\rho) = \rho^{\#}(X \; \triangleright \; \lambda x\!:\!s.E\!:\!s \to t) =$$

$$= [\bigcup_{i=1}^{n} X_i \; \triangleright \; \lambda x\!:\!s.E[x_1,\ldots,x_n \leftarrow E_1,\ldots,E_n] : s \to t]_{\mathcal{E}} \,,$$

so the left-hand side of the equality becomes, after an application of (β), $[Y \cup \bigcup_{i=1}^{n} X_i \; \triangleright \; E[x_1,\ldots,x_n, x \leftarrow E_1,\ldots,E_n,F]\!:\!t]_{\mathcal{E}}$, which is by definition equal to $[\mathcal{T}_{\mathcal{E},(X,x:s \,\triangleright\, E:t)}(\rho[x \leftarrow [Y \; \triangleright \; F\!:\!s]_{\mathcal{E}}])]$.

3) Easy.

Exercise 9 *Prove 3) above, thus completing the proof that $\mathcal{T}_{\mathcal{E}}$ is a model.*

Completeness

We are now ready to prove one of the most important results of simply-typed λ-calculus, namely the completeness of the equational deduction rules. In other words, we show that the equational rules are sufficient to derive any equation that is true in all models of λ-calculus.

Let us first investigate the satisfaction in $\mathcal{T}_\mathcal{E}$. By definition, $\mathcal{T}_\mathcal{E} \models (\forall X)\ E =_t E'$ iff for any $\mathcal{T}_\mathcal{E}$-environment $\rho : X \to \mathcal{T}_\mathcal{E}$, it is the case that $\rho^\#(X \vartriangleright E : t) = \rho^\#(X \vartriangleright E' : t)$. If $\rho(x_i : s_i) = [X_i \vartriangleright E_i : s_i]_\mathcal{E}$ for any $x_i : s_i \in X$, then the above says that $[\bigcup_{i=1}^n X_i \vartriangleright E[x_1, \ldots, x_n \leftarrow E_1, \ldots E_n] : t]_\mathcal{E} = [\bigcup_{i=1}^n X_i \vartriangleright E'[x_1, \ldots, x_n \leftarrow E_1, \ldots E_n] : t]_\mathcal{E}$, or by Proposition 5 that there is some $Y \subseteq \mathcal{X}$ such that $\mathcal{E} \vdash (\forall Y)\ E[x_1, \ldots, x_n \leftarrow E_1, \ldots E_n] =_t E'[x_1, \ldots, x_n \leftarrow E_1, \ldots E_n]$. Taking ρ to be the identity $\mathcal{T}_\mathcal{E}$-environment, that is,

$\rho(x_i : s_i) = [x_i : s_i \,\triangleright\, x_i : s_i]$, we obtain that $\mathcal{T}_{\mathcal{E}} \models (\forall X)E =_t E'$ implies $\mathcal{E} \vdash (\forall Y)E =_t E'$ for some $Y \subseteq \mathcal{X}$. By Proposition 3, we then get the following important result:

Proposition 8 $\mathcal{T}_{\mathcal{E}} \models (\forall X)\, E =_t E'$ *iff* $\mathcal{E} \vdash (\forall X)\, E =_t E'$.

Corollary 2 $\mathcal{T}_{\mathcal{E}} \models \mathcal{E}$.

Theorem *(Completeness) If* $\mathcal{E} \models e$ *then* $\mathcal{E} \vdash e$.

Proof. If $\mathcal{E} \models e$ then by Corollary 2 we get that $\mathcal{T}_{\mathcal{E}} \models e$, so by Proposition 8 we obtain that $\mathcal{E} \vdash e$. $\qquad\square$