Assignment on Types

For your reference, we give the formalization of the untyped λ-calculus, the simply-typed λ-calculus (STLC) and System F in the appendix at the end of this document.

1. In this problem we add the option types to the simply-typed λ-calculus. We can use None and Some to construct terms of the option type, just like `None` and `Some` in Coq. You can also think about `std::optional` in C++. Intuitively, None represents a dummy element (i.e. there is no meaningful element), and $\mathsf{Some}\ M$ means that there is a meaningful element M. For M which has the option type, $\mathsf{get}\ M$ gives us the meaningful element contained in M.

   Syntax:

   $$\begin{array}{llll} (\mathsf{Types}) & \tau & ::= & \ldots \mid \mathsf{option}\ \tau \\ (\mathsf{Terms}) & M & ::= & \ldots \mid \mathsf{None} \mid \mathsf{Some}\ M \mid \mathsf{get}\ M \\ (\mathsf{Values}) & v & ::= & \ldots \mid \mathsf{None} \mid \mathsf{Some}\ v \end{array}$$

   New reduction rules:

   $$\frac{M \to M'}{\mathsf{Some}\ M \to \mathsf{Some}\ M'}\ (\textsc{some}) \qquad \frac{M \to M'}{\mathsf{get}\ M \to \mathsf{get}\ M'}\ (\textsc{get-m})$$

   $$\frac{}{\mathsf{get}\ (\mathsf{Some}\ M) \to M}\ (\textsc{get-some}) \qquad \frac{}{\mathsf{get\ None} \to \mathsf{get\ None}}\ (\textsc{get-none})$$

   (a) Give 3 appropriate new typing rules, one for each new form of term. Note that your rules should ensure the preservation and progress theorems (though you don't need to show their proofs).

   (b) Consider each of the following questions in isolation. Answer yes or no.

      i. Suppose we remove the above (SOME) rule.
      Does the preservation theorem still hold?
      Does the progress theorem still hold?

      ii. Suppose we add the following rule.

      $$\frac{}{\mathsf{get}\ v \to \mathsf{get}\ v}\ (\textsc{get-v})$$

Does the preservation theorem still hold?
Does the progress theorem still hold?

iii. Suppose we change the above (GET-SOME) rule to the following (GET-SOME') rule.

$$\frac{}{\mathsf{get}\ (\mathsf{Some}\ v) \to v}\ (\text{GET-SOME'})$$

Does the preservation theorem still hold?
Does the progress theorem still hold?

iv. Suppose we change the above (GET-NONE) rule to the following (GET-NONE') rule.

$$\frac{}{\mathsf{get}\ \mathsf{None} \to \mathsf{None}}\ (\text{GET-NONE'})$$

Does the preservation theorem still hold?
Does the progress theorem still hold?

2. In this problem we consider System F. Show that the type

$$\mathsf{PairNat} \stackrel{\text{def}}{=} \forall\alpha.\ (\mathsf{Nat} \to \mathsf{Nat} \to \alpha) \to \alpha$$

can be used to represent pairs of numbers, by writing functions

$$\begin{aligned}&\mathsf{mkPairNat} : \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{PairNat}\\ &\mathsf{fstNat} : \mathsf{PairNat} \to \mathsf{Nat}\\ &\mathsf{sndNat} : \mathsf{PairNat} \to \mathsf{Nat}\end{aligned}$$

for constructing elements of this type from pairs of numbers and for accessing their first and second components. Here Nat is the type of Church numerals.

Hint: In the untyped λ-calculus, we can encode mkPair, fst and snd as follows:

$$\begin{aligned}\mathsf{mkPair} &\stackrel{\text{def}}{=} \lambda x.\ \lambda y.\ \lambda z.\ z\,x\,y\\ \mathsf{fst} &\stackrel{\text{def}}{=} \lambda p.\ p\,(\lambda x.\ \lambda y.\ x)\\ \mathsf{snd} &\stackrel{\text{def}}{=} \lambda p.\ p\,(\lambda x.\ \lambda y.\ y)\end{aligned}$$

3. Use the functions defined in Problem 2 to write a function pred in System F that computes the predecessor of a Church numeral (returning 0 if its input is 0).

Hint: Define a function $f : \mathsf{PairNat} \to \mathsf{PairNat}$ that maps the pair (i, j) into $(i{+}1, i)$–that is, it throws away the second component of its argument, copies the first component to the second, and increments the first. Then n applications of f to the starting pair $(0, 0)$ yields the pair $(n, n-1)$, from which we extract the predecessor of n by projecting the second component.

Appendix

A The Untyped λ-Calculus

Syntax:

$$\begin{array}{rrcl} (\mathsf{Terms}) & M & ::= & x \mid \lambda x.\ M \mid M\ N \\ (\mathsf{Values}) & v & ::= & \lambda x.\ M \end{array}$$

Reduction rules:

$$\frac{}{(\lambda x.\ M)\ N\ \rightarrow\ M[N/x]} \qquad \frac{M \rightarrow M'}{\lambda x.\ M\ \rightarrow\ \lambda x.\ M'}$$

$$\frac{M \rightarrow M'}{M\ N\ \rightarrow\ M'\ N} \qquad \frac{N \rightarrow N'}{M\ N\ \rightarrow\ M\ N'}$$

Substitution:

$$\begin{array}{l} x[N/x] = N \\ y[N/x] = y \\ (M\,N)[N'/x] = (M[N'/x])\,(N[N'/x]) \\ (\lambda x.\,M)[N/x] = \lambda x.\,M \\ (\lambda y.\,M)[N/x] = \lambda y.\,(M[N/x]), \quad \text{where } y \notin \mathit{fv}(N) \\ (\lambda y.\,M)[N/x] = \lambda z.\,(M[z/y])[N/x], \quad \text{where } y \in \mathit{fv}(N) \text{ and } z \text{ fresh} \end{array}$$

Free variables:

$$\mathit{fv}(x) = \{x\} \qquad \mathit{fv}(M\,N) = \mathit{fv}(M) \cup \mathit{fv}(N) \qquad \mathit{fv}(\lambda x.\,M) = \mathit{fv}(M) - \{x\}$$

Zero-or-more steps:

$$\begin{array}{lll} M \rightarrow^0 M' & \text{iff} & M = M' \\ M \rightarrow^{k+1} M' & \text{iff} & \exists M''.\ M \rightarrow M'' \wedge M'' \rightarrow^k M' \\ M \rightarrow^* M' & \text{iff} & \exists k.\ M \rightarrow^k M' \end{array}$$

Normal form: a term containing no redex.
Closed term: a term containing no free variables.

B The Simply-Typed λ-Calculus

Syntax (here $\mathtt{T}$ denotes the base type):

$$\begin{array}{rrcl} (\mathsf{Types}) & \tau & ::= & \mathtt{T} \mid \tau \rightarrow \tau \\ (\mathsf{Terms}) & M & ::= & x \mid \lambda x : \tau.\ M \mid M\ N \\ (\mathsf{Values}) & v & ::= & \lambda x : \tau.\ M \\ (\mathsf{Contexts}) & \Gamma & ::= & \bullet \mid \Gamma, x : \tau \end{array}$$

Reduction rules:

$$\frac{}{(\lambda x : \tau.\ M)\ N\ \rightarrow\ M[N/x]} \qquad \frac{M \rightarrow M'}{\lambda x : \tau.\ M\ \rightarrow\ \lambda x : \tau.\ M'}$$

$$\frac{M \rightarrow M'}{M\ N\ \rightarrow\ M'\ N} \qquad \frac{N \rightarrow N'}{M\ N\ \rightarrow\ M\ N'}$$

Typing rules:

$$\frac{}{\Gamma, x : \tau \vdash x : \tau} \qquad \frac{\Gamma, x : \tau \vdash M : \tau'}{\Gamma \vdash (\lambda x : \tau.\ M) : \tau \rightarrow \tau'}$$

$$\frac{\Gamma \vdash M : \tau \rightarrow \tau' \qquad \Gamma \vdash N : \tau}{\Gamma \vdash M\ N : \tau'}$$

Preservation:

For any M, M' and τ, if $\bullet \vdash M : \tau$ and $M \rightarrow M'$, then $\bullet \vdash M' : \tau$.

Progress:

For any M and τ, if $\bullet \vdash M : \tau$, then either $M \in \mathsf{Values}$ or $\exists M'.\ M \rightarrow M'$.

C System F

Syntax (here $\mathtt{T}$ denotes the base type):

$$\begin{array}{rrcl} (\mathsf{Terms}) & M & ::= & x \mid \lambda x : \tau.\ M \mid M\,M \mid \Lambda\alpha.\ M \mid M\,\langle\tau\rangle \\ (\mathsf{Types}) & \tau & ::= & \alpha \mid \mathtt{T} \mid \tau \rightarrow \tau \mid \forall\alpha.\ \tau \\ (\mathsf{Values}) & v & ::= & \lambda x : \tau.\ M \mid \Lambda\alpha.\ M \end{array}$$

$$\begin{array}{rrcl} (\mathsf{Contexts}) & \Gamma & ::= & \bullet \mid \Gamma, x : \tau \\ (\mathsf{TypeVarContexts}) & \Delta & ::= & \bullet \mid \Delta, \alpha \end{array}$$

Reduction rules:

$$\frac{}{(\lambda x : \tau.\ M_1)\,M_2 \longrightarrow M_1[M_2/x]}$$

$$\frac{M_1 \longrightarrow M_1'}{M_1\,M_2 \longrightarrow M_1'\,M_2} \qquad \frac{M_2 \longrightarrow M_2'}{M_1\,M_2 \longrightarrow M_1\,M_2'}$$

$$\frac{M \longrightarrow M'}{\lambda x : \tau.\ M \longrightarrow \lambda x : \tau.\ M'}$$

$$\frac{}{(\Lambda\alpha.\ M_1)\,\langle\tau_2\rangle \longrightarrow M_1[\tau_2/\alpha]}$$

$$\frac{M_1 \longrightarrow M_1'}{M_1\,\langle\tau_2\rangle \longrightarrow M_1'\,\langle\tau_2\rangle} \qquad \frac{M \longrightarrow M'}{\Lambda\alpha.\ M \longrightarrow \Lambda\alpha.\ M'}$$

Type well-formedness rules:

$$\frac{}{\Delta, \alpha \vdash \alpha} \qquad \frac{}{\Delta \vdash \mathtt{T}} \qquad \frac{\Delta \vdash \tau_1 \qquad \Delta \vdash \tau_2}{\Delta \vdash \tau_1 \to \tau_2} \qquad \frac{\Delta, \alpha \vdash \tau}{\Delta \vdash \forall \alpha.\ \tau}$$

Typing rules:

$$\frac{}{\Delta; \Gamma, x : \tau \vdash x : \tau}\ (\text{T-Var})$$

$$\frac{\Delta \vdash \tau_1 \qquad \Delta; \Gamma, x : \tau_1 \vdash M : \tau_2}{\Delta; \Gamma \vdash (\lambda x : \tau_1.\ M) : \tau_1 \to \tau_2}\ (\text{T-Abs})$$

$$\frac{\Delta; \Gamma \vdash M_1 : \tau \to \tau' \qquad \Delta; \Gamma \vdash M_2 : \tau}{\Delta; \Gamma \vdash M_1\, M_2 : \tau'}\ (\text{T-App})$$

$$\frac{\Delta, \alpha; \Gamma \vdash M : \tau}{\Delta; \Gamma \vdash (\Lambda \alpha.\ M) : \forall \alpha . \tau}\ (\text{T-TAbs})$$

$$\frac{\Delta; \Gamma \vdash M_1 : \forall \alpha . \tau \qquad \Delta \vdash \tau_2}{\Delta; \Gamma \vdash M_1\, \langle \tau_2 \rangle : \tau[\tau_2/\alpha]}\ (\text{T-TApp})$$

Church numerals:

$$\begin{aligned} 0 &\stackrel{\text{def}}{=} \Lambda \alpha.\ \lambda f : \alpha \to \alpha.\ \lambda x : \alpha.\ x \\ 1 &\stackrel{\text{def}}{=} \Lambda \alpha.\ \lambda f : \alpha \to \alpha.\ \lambda x : \alpha.\ f\, x \\ 2 &\stackrel{\text{def}}{=} \Lambda \alpha.\ \lambda f : \alpha \to \alpha.\ \lambda x : \alpha.\ f\, (f\, x) \end{aligned}$$

The type of Church numerals:

$$\mathsf{Nat} \stackrel{\text{def}}{=} \forall \alpha.\ (\alpha \to \alpha) \to \alpha \to \alpha$$