Computer Science 131, Fall 2000
Assignment 7: Polymorphism, Subtyping, and λ-Calculus Sample Solution

December 7, 2000

1 Subtyping (20%)

1. t_1 `list` $\preceq$ t_2 `list` should hold if $t_1 \preceq t_2$. The only thing we can do with a value of type t_2 `list` is extract the elements and use them as values of type t_2. If every t_1 is a t_2, it suffices to supply a list of values of type t_1 instead.

2. t_1 `foo` $\preceq$ t_2 `foo` (defined by `type 'a foo = 'a -> 'a`) should hold if and only if $t_1 = t_2$. The argument for functions given in class shows that t_1`->`t_1 and t_2`->`t_2 must be unrelated unless $t_1 \preceq t_2$ *and* $t_2 \preceq t_1$.

3. t_1 `bar` $\preceq$ t_2 `bar` (defined by `type 'a bar = ('a -> int) -> int`) should hold when $t_1 \preceq t_2$. It follows from the rule for functions given in class that this is safe.

4. t_1 `cont` $\preceq$ t_2 `cont` if $t_2 \preceq t_1$. The only thing you can do with a value of type t_2 `cont` is to throw a value of type t_2 to it, which will be used to compute the rest of the program. If every t_2 is a t_1, then we can safely throw values of type t_2 to a t_1 `cont`.

2 Fixed Points (10%)

Put $D := (\lambda x.\lambda y.y(xxy))$, so that $\Theta = DD$. Then $\Theta M = (DD)M = ((\lambda x.\lambda y.y(xxy))D)(M) \rightarrow_\beta (\lambda y.y(DDy))(M) \rightarrow_\beta M(DDM) = M(\Theta M)$.

3 Predecessor (20%)

1. - $\overline{0}\ M_1\ M_2 = (\lambda b.\lambda f.b)\ M_1\ M_2 \rightarrow_\beta (\lambda f.M_1)M_2 \rightarrow_\beta M_1$
   - $\overline{n+1}\ M_1\ M_2 = (\lambda b.\lambda f.f^{(n+1)}(b))\ M_1\ M_2 \rightarrow_\beta (\lambda f.f^{(n+1)}(M_1))M_2 \rightarrow_\beta M_2^{(n+1)}(M_1) = M_2(M_2^{(n)}(M_1)) \leftarrow_\beta M_2((\lambda f.f^{(n)}(M_1))M_2) \leftarrow_\beta M_2((\lambda b.\lambda f.f^{(n)}(b))\ M_1\ M_2) = M_2(\overline{n}\ M_1\ M_2)$

2.

$$\begin{array}{lcl} pred & = & \lambda n.(pred'\ n).1 \\ pred' & = & \lambda m.m\ \langle\, \overline{0}\,,\, \overline{0}\,\rangle\ (\lambda p.\langle p.2, succ(p.2)\rangle) \end{array}$$

To show: $\forall m \geq 0.\ pred'\ \overline{m+1} \longleftrightarrow_\beta^* \langle\, \overline{m}\,,\, \overline{m+1}\,\rangle$.

By induction.

- Case: $m = 0$. Then $pred'\ \overline{1} = (\lambda m.m\ \langle\, \overline{0}\,,\, \overline{0}\,\rangle\ (\lambda p.\langle p.2, succ(p.2)\rangle))(\,\overline{1}\,) \rightarrow_\beta$ $\overline{1}\ \langle\, \overline{0}\,,\, \overline{0}\,\rangle\ (\lambda p.\langle p.2, succ(p.2)\rangle) \rightarrow_\beta{}^2\ (\lambda p.\langle p.2, succ(p.2)\rangle)\langle\, \overline{0}\,,\, \overline{0}\,\rangle \rightarrow_\beta{}^*\ \langle\, \overline{0}\,,\, \overline{1}\,\rangle$.
- Inductive step: Assume $pred'\ \overline{m+1} \longleftrightarrow_\beta^* \langle\, \overline{m}\,,\, \overline{m+1}\,\rangle$; we need to show that $pred'\ \overline{m+2} \longleftrightarrow_\beta^* \langle\, \overline{m+1}\,,\, \overline{m+2}\,\rangle$. Then $pred'\ \overline{m+2} \rightarrow_\beta \overline{m+2}\ \langle\, \overline{0}\,,\, \overline{0}\,\rangle\ (\lambda p.\langle p.2, succ(p.2)\rangle)$. By the previous part, this $\rightarrow_\beta{}^*\ (\lambda p.\langle p.2, succ(p.2)\rangle)(\,\overline{m+1}\ \langle\, \overline{0}\,,\, \overline{0}\,\rangle\ (\lambda p.\langle p.2, succ(p.2)\rangle)) \leftarrow_\beta$ $(\lambda p.\langle p.2, succ(p.2)\rangle)(pred'\ \overline{m+1}\,)$. By the inductive hypothesis, this $\longleftrightarrow_\beta\ (\lambda p.\langle p.2, succ(p.2)\rangle)\langle\, \overline{m}\,,\, \overline{m+1}\,\rangle \rightarrow_\beta{}^*\ \langle\, \overline{m+1}\,,\, \overline{m+2}\,\rangle$.

3. $pred\ \overline{m+1} \rightarrow_\beta (pred'\ \overline{m+1}\,).1 \longleftrightarrow_\beta{}^*\ \langle\, \overline{m}\,,\, \overline{m+1}\,\rangle.1 \longleftrightarrow_\beta{}^*\ \overline{m}\,.$

4 Lambda Calculus Encodings (30%)

For this problem you will devise an encoding for lists within the untyped λ-calculus. Recall that a list is either empty or it has a head (first element) and a tail (a list containing the rest of the elements, possibly empty).

1. There are many, many possible encodings; here's a fairly direct encoding using pairing and booleans as defined in class. A list a pair whose first element is a boolean saying whether it is nil or not; if not, the second element is a pair containing the head and tail of the list.

$$\begin{array}{l} nil := \langle tt, M_0\rangle \\ cons := \lambda h.\lambda t.\langle ff, \langle h, t\rangle\rangle \\ isnil := \lambda l.(l.1) \\ hd := \lambda l.((l.2).1) \\ tl := \lambda l.((l.2).2) \end{array}$$

   where M_0 is arbitrary.

   Here's another; the idea is that lists are "things that one can do `listcase`-like operation on; if M_1 is a term representing a list, then $M_1\ M_2\ (\lambda x.\lambda y.M_3)$ will act like `listcase` M_1 `of nil =>` M_2 `| x::y =>` M_3.

$$\begin{array}{l} nil := \lambda n.\lambda c.n \\ cons := \lambda h.\lambda t.\lambda n.\lambda c.c\ h\ t \\ isnil := \lambda l.l\ tt\ (\lambda h.\lambda t.ff) \\ hd := \lambda l.l\ M_0\ (\lambda h.\lambda t.h) \\ tl := \lambda l.l\ M_0\ (\lambda h.\lambda t.t) \end{array}$$

   where M_0 is arbitrary. Then

- $isnil\ \ nil \rightarrow_\beta{}^* (\lambda n.\lambda c.n)\ tt\ (\lambda h.\lambda t.ff) \rightarrow_\beta{}^2 tt$.
- $isnil\ (cons\ M_1\ M_2) \rightarrow_\beta{}^* isnil(\lambda n.\lambda c.c\ M_1\ M_2) \rightarrow_\beta (\lambda n.\lambda c.c\ M_1\ M_2)\ tt\ (\lambda h.\lambda t.ff) \rightarrow_\beta{}^2 (\lambda h.\lambda t.ff)\ M_1\ M_2 \rightarrow_\beta{}^2 ff$.
- $hd\ (cons\ M_1\ M_2) \rightarrow_\beta{}^* hd(\lambda n.\lambda c.c\ M_1\ M_2) \rightarrow_\beta (\lambda n.\lambda c.c\ M_1\ M_2)M_0(\lambda h.\lambda t.h) \rightarrow_\beta{}^2 (\lambda h.\lambda t.h)\ M_1\ M_2 \rightarrow_\beta{}^2 M_1$.
- $tl\ \ (cons\ M_1\ M_2) \rightarrow_\beta{}^* tl(\lambda n.\lambda c.c\ M_1\ M_2) \rightarrow_\beta (\lambda n.\lambda c.c\ M_1\ M_2)M_0(\lambda h.\lambda t.t) \rightarrow_\beta{}^2 (\lambda h.\lambda t.t)\ M_1\ M_2 \rightarrow_\beta{}^2 M_2$.

2. Let $length := Y(\lambda f.\lambda l.(isnil\ \ l)\,\overline{0}\,(succ(f(tl\ \ l))))$.

3. $zeros := Y(\lambda l.\ cons\ \ \overline{0}\ \ l)$.

   First, note that $zeros \longleftrightarrow_\beta ((\lambda l.\ cons\ \ \overline{0}\ \ l)zeros) \longleftrightarrow_\beta cons\ \ \overline{0}\ \ zeros$. Thus $hd\ zeros \longleftrightarrow_\beta hd(cons\ \ \overline{0}\ \ zeros) \longleftrightarrow_\beta \overline{0}$ and $tl\ \ zeros \longleftrightarrow_\beta tl(cons\ \ \overline{0}\ \ zeros) \longleftrightarrow_\beta zeros$. Hence $hd(tl^{(n)}\ zeros) \longleftrightarrow_\beta{}^*\ \overline{0}$ for every $n \geq 0$.

   The definition $Y(cons\ \ \overline{0})$ is even simpler, while more convoluted definitions are possible as well.

5 Theorems for Free (20% or more)

1. Assume $f : \forall\alpha.\alpha\texttt{->}(\alpha\texttt{->}\alpha)$. By the Parametericity theorem, we have $(f, f) \in \mathtt{Rel_V}(\forall\alpha.\alpha\texttt{->}(\alpha\texttt{->}\alpha))$. Expanding out this definition, we have that for all types t_1 and t_2, and for all $\mathcal{R}$, and for all values v_1, v_2, w_1, w_2, if $(v_1, w_1) \in \mathcal{R}$ and $(v_2, w_2) \in \mathcal{R}$ then $(\lfloor f[t_1](v_1)(v_2)\rfloor, \lfloor f[t_2](w_1)(w_2)\rfloor) \in \mathcal{R}$.

   Let t, v_1, and v_2 be given. Let $\mathcal{R}$ be the relation which only relates v_1 to $\overline{\mathtt{tt}}$ and v_2 to $\overline{\mathtt{ff}}$. Then plugging in $t_1 = t$, $t_2 = \mathtt{Bool}$, $w_1 = \overline{\mathtt{tt}}$, and $w_2 = \overline{\mathtt{ff}}$, we have that $(\lfloor f[t](v_1)(v_2)\rfloor, \lfloor f[\mathtt{Bool}](\overline{\mathtt{tt}})(\overline{\mathtt{ff}})\rfloor) \in \mathcal{R}$. Now since $\mathcal{R}$ only relates values to $\overline{\mathtt{tt}}$ or $\overline{\mathtt{ff}}$, it must be that $b := \lfloor f[\mathtt{Bool}](\overline{\mathtt{tt}})(\overline{\mathtt{ff}})\rfloor$ is either $\overline{\mathtt{tt}}$ or $\overline{\mathtt{ff}}$. Furthermore, if $b = \overline{\mathtt{tt}}$ then $\lfloor f[t](v_1)(v_2)\rfloor = v_1$ and otherwise $\lfloor f[t](v_1)(v_2)\rfloor = v_2$.

   This shows that f always returns one of its two arguments, but are we sure that it always returns the same one (rather than returning the first or the second depending on what arguments are given)? Yes; since b does not depend on the choice of v_1 and v_2, it immediately follows either f always returns its first argument (if b is true) or else f always returns its second argument (if b is false).

2. **[10% EXTRA CREDIT]** Recall that the function `map` applies a given function to every element of a list and returns the list of results. Assume $f : \forall\alpha.(\alpha\ \mathtt{list})\texttt{->}(\alpha\ \mathtt{list})$. Show that for any types t and u, function $g : t\rightarrow u$, and list $v : t\ \mathtt{list}$, it must be the case that $\lfloor \mathtt{map}\ g\ (f[t](v))\rfloor = \lfloor f[u](\mathtt{map}\ g\ v)\rfloor$ evaluate to the same value. Hint: a function can be viewed as a relation between values.

3. **[10% EXTRA CREDIT]** Assume $\mathtt{sw} : \forall\alpha_1.\forall\alpha_2.(\alpha_1 \times \alpha_2)\texttt{->}(\alpha_2 \times \alpha_1)$. Show that if $v_1 : t_1$ and $v_2 : t_2$ then $\lfloor \mathtt{sw}[t_1][t_2]\,(v_1, v_2)\rfloor = (v_2, v_1)$