1 Hereditary Substitution [30 points]

The operation of hereditary substitution is like ordinary (capture-avoiding) substitution, but it does some \( \beta \)-reduction as it goes. The notation is \([t/x]^\phi t'\), where \( t \) is the term to substitute for variable \( x \) in term \( t' \), and \( \phi \) is a type which limits the \( \beta \)-reduction which is allowed to happen. The definition is as follows (where I write \( b \) for a sole base type):

\[
\begin{align*}
[t/x]^T_x &= t \\
[t/x]^T_y &= y, \text{ if } y \in \text{Var}, \ y \neq x \\
[t/x]^T \lambda y.t' &= \lambda y.([t/x]^T t') \\
[t/x]^T (t_1 t_2) &= ([t/x]^T t_1) ([t/x]^T t_2) \\
[t/x]^{T ightarrow T'} (t_1 t_2) &= \text{let } s_1 = ([t/x]^{T ightarrow T'} t_1) \text{ in} \\
&\quad \text{let } s_2 = ([t/x]^{T ightarrow T'} t_2) \text{ in} \\
&\quad \text{if } s_1 \equiv \lambda y.s'_1 \text{ for some } y, s'_1, \text{ then} \\
&\quad \quad \quad \quad [s_2/y]^T s'_1 \\
&\quad \quad \quad \quad \text{else} \\
&\quad \quad \quad \quad (s_1 \ s_2)
\end{align*}
\]

1. Using the above definition, compute the values of the following [5 points each]:

a. \([(\lambda x. (z (x x)))/y]^b(y y)\)
   
   \((\lambda x. (z (x x))) \ (\lambda x. (z (x x)))\)

b. \([(\lambda x. (z (x x)))/y]^b\rightarrow b(y y)\)
   
   \((z ((\lambda x. (z (x x))) \ (\lambda x. (z (x x)))))\)

c. \([(\lambda x. (z (x x)))/y]^0\rightarrow b(y y)\)
   
   \((z (z ((\lambda x. (z (x x))) \ (\lambda x. (z (x x)))))\))

2. Let us call a measure of the inputs to hereditary substitution sometimes decreasing iff there is at least one equation in the definition above where that measure decreases from the left hand side to some recursive call on the right hand side. List two measures that are sometimes decreasing. [5 points]

1. The term \( t' \).
2. The type \( T \).
3. Prove that hereditary substitution always terminates, by defining a measure which is always decreasing from each left hand side above to each recursive call on the corresponding right hand side. [10 points]

The measure is the pair \((T, t')\), in lexicographic combination of the strict-subexpression orderings for \(T\) and \(t'\).

2 Constraint-based typing [25 points]

1. Using the constraint-generating typing rules, compute the set of constraints for the term \(\lambda n. \lambda s. \lambda z. (s\ (n\ s\ z))\). Show the typing derivation that generates the constraints. [10 points].

\[
\begin{align*}
\frac{n : N, s : S, z : Z \vdash n : N > \cdot}{n : N, s : S, z : Z \vdash n : S > \cdot} & \frac{n : N, s : S, z : Z \vdash n : S > X_1}{n : N, s : S, z : Z \vdash n : S > X_1} & \frac{n : N, s : S, z : Z \vdash n : S > X_2}\end{align*}
\]

\[
\frac{n : N, s : S, z : Z \vdash n : S > X_1}{n : N, s : S, z : Z \vdash (n\ s\ z) : X_2 > N = S \rightarrow X_1, X_1 = Z \rightarrow X_2, S = X_2 \rightarrow X_3}\]

\[
\frac{n : N, s : S, z : Z \vdash n : S > X_1}{n : N, s : S, z : Z \vdash (n\ s\ z) : X_2 > N = S \rightarrow X_1, X_1 = Z \rightarrow X_2, S = X_2 \rightarrow X_3}\]

\[
\frac{n : N, s : S, z : Z \vdash n : S > X_1}{n : N, s : S, z : Z \vdash (n\ s\ z) : X_2 > N = S \rightarrow X_1, X_1 = Z \rightarrow X_2, S = X_2 \rightarrow X_3}\]

2. Using the constraint-solving rules (i.e., the unification rules), compute a solved form for those constraints (hint: they are solvable). Show the derivation you use. [10 points]

\[
\begin{align*}
N = S \rightarrow X_1, X_1 = Z \rightarrow X_2, S = X_2 \rightarrow X_3
\end{align*}
\]

\[
\begin{align*}
\frac{N = (X_2 \rightarrow X_3) \rightarrow X_1, X_1 = Z \rightarrow X_2, S = X_2 \rightarrow X_3}{N = (X_2 \rightarrow X_3) \rightarrow (Z \rightarrow X_2) X_1 = Z \rightarrow X_2, S = X_2 \rightarrow X_3}
\end{align*}
\]

3. Apply the substitution determined by the solved form of the constraints to the type (with meta-variables) computed by the derivation from part (1). [5 points]

\[
((X_2 \rightarrow X_3) \rightarrow (Z \rightarrow X_2)) \rightarrow (X_2 \rightarrow X_3) \rightarrow Z \rightarrow X_3
\]

3 Reducibility [15 points]

1. Argue that \(((\lambda x. x)\ (\lambda y. y))\) \(\in [b \rightarrow b]\) [5 points].

It suffices to assume an arbitrary \(t \in [b]\), and show \(((\lambda x. x)\ (\lambda y. y))\ t) \in [b].\ Since \([b] = SN\), this is equivalent to assuming arbitrary \(t \in SN\), and showing \(((\lambda x. x)\ (\lambda y. y))\ t) \in SN\). But that term is strongly normalizing if \(t\) is.
2. Give an example of a term which is in \([b \to b]\) but not \([b \to b \to b]\). Give an argument for why your term is in the first set but not the second [10 points].

An example is just \(\lambda x.x\). This is in \([b \to b]\), because given a strongly normalizing \(t\), the term \((\lambda x.x) t\) is also strongly normalizing. But \(\lambda x.x \notin [b \to b \to b]\), because if we take \(\lambda y.y\ y\), which is in \([b]\), then we should have \((\lambda x.x) (\lambda y.y\ y) \in [b]\), but this term diverges.

4 System F and Church-Encoded Data [30 points]

1. Give a Church-encoding in System F of the datatype of polymorphic binary trees – including a definition of \(\langle\text{tree}\ A\rangle\) parametric in the type \(A\) – with these term constructors [10 points]:

\[
\begin{align*}
\langle\text{tree}\ A\rangle & := \forall X. (A \to X) \to (A \to X \to X \to X) \to X \\
\text{leaf} & := \lambda A. \lambda a : A. \lambda X. \lambda l : (A \to X). \lambda n : (A \to X \to X \to X). (l a) \\
\text{node} & := \lambda A. \lambda a : A. \lambda L : \langle\text{tree}\ A\rangle. \lambda R : \langle\text{tree}\ A\rangle. \lambda X : (A \to X). \lambda n : (A \to X \to X \to X). (\text{cons}\ a (\text{append}\ A\ L\ R)\ n)
\end{align*}
\]

2. Write down the term in System F corresponding to the tree with 1 at the root, left subtree a leaf with 0, and right subtree a leaf with 0, where 1 and 0 are the Church-encoded versions of the corresponding natural numbers [10 points].

\[
\lambda X. \lambda l : (\text{nat} \to X). \lambda n : (\text{nat} \to X \to X \to X). (n\ 1\ (l\ 0)\ (l\ 0))
\]

3. Write down a term in System F which will compute the list (using the Church-encoded version of lists, which we saw in class) of elements stored in the tree, in prefix order [10 points].

\[
\begin{align*}
\lambda A. \lambda t : \langle\text{tree}\ A\rangle. \\
(t[(\text{list}\ A)] (\lambda a : A. (\text{cons}\ A\ a\ \text{nil}\ A)) (\lambda a : A. \lambda L : \langle\text{list}\ A\rangle. \lambda R : \langle\text{list}\ A\rangle. (\text{cons}\ a (\text{append}\ A\ L\ R)))
\end{align*}
\]

where \text{append} is as we defined in class.