1 Type Safety for System F

Recall the grammar for System F:

\[ \tau ::= t | \tau \rightarrow \tau | \forall t.\tau \]

\[ M ::= x | \lambda x : \tau. M | MN | \Lambda t. M | M[\tau] \]

and the statics and dynamics:

\[ \Delta, x : \tau \vdash x : \tau \]

\[ \Delta, \tau \vdash \lambda x : \tau. M : \tau \rightarrow \tau_2 \]

\[ \Delta, \tau \vdash M : \tau_1 \rightarrow \tau_2 \]

\[ \Delta, \tau \vdash N : \tau_1 \]

\[ \Delta ; \tau \vdash \tau \]

\[ \text{FV}(\tau) \subseteq \text{dom}(\Delta) \]

Before we can prove preservation and progress for System F, we need to know that substitution of types and terms preserves both typing and well-formedness of types. In the following lemma, \([\tau/t]\Gamma\) stands for the context obtained by substituting \(\tau\) for \(t\) in each type in \(\Gamma\).

**Lemma 1.1** (Substitution).

1. If \(\Delta, t \vdash \tau\) type and \(\Delta \vdash \tau'\) type, then \(\Delta \vdash [\tau'/t]\tau\) type.
2. If \(\Delta, t \vdash \tau\) type and \(\Gamma \vdash M : A\) and \(\Delta \vdash \tau\) type, then \(\Delta; [\tau/t]\Gamma \vdash [\tau/t]M : [\tau/t]A\).
3. If \(\Delta; x : A \vdash M : B\) and \(\Delta; \tau \vdash N : A\), then \(\Delta; \Gamma \vdash [N/x]M : B\).

Now we are prepared to prove preservation and progress. Recall from the previous homework that type safety is an easy consequence of these theorems.

**Task 1** (Preservation). If \(\cdot ; \vdash M : \tau\) and \(M \mapsto M'\) then \(\cdot ; \vdash M' : \tau\).

**Task 2** (Progress). If \(\cdot ; \vdash M : \tau\) then either \(M\) val or \(M \mapsto M'\), for some \(M'\).
2 Polymorphic Church Encodings

Polymorphism gives us the power to Church-encode data types like we could in the untyped λ-calculus, but now we have the power of types to ensure that we don’t pass illegal terms to our constructors and recursors.

In this task, we will Church-encode typed sums, where the left and right branches of the sum are required to contain a term of the correct type. In particular, for any two types \( \tau_1, \tau_2 \), we will define a type \( \text{Sum}(\tau_1, \tau_2) \) and terms \( \text{inl}, \text{inr}, \text{rec}_{\text{Sum}} \) with the following properties:

\[
\begin{align*}
\text{inl} : & \tau_1 \rightarrow \text{Sum}(\tau_1, \tau_2) \\
\text{inr} : & \tau_2 \rightarrow \text{Sum}(\tau_1, \tau_2) \\
\text{rec}_{\text{Sum}} : & \forall t. \text{Sum}(\tau_1, \tau_2) \rightarrow (\tau_1 \rightarrow t) \rightarrow (\tau_2 \rightarrow t) \rightarrow t
\end{align*}
\]

Task 3. Define \( \text{Sum}(\tau_1, \tau_2) \), \( \text{inl} \), \( \text{inr} \), and \( \text{rec}_{\text{Sum}} \).

3 Contextual and Logical Equivalence

Now, extend System F to include an observable type of booleans \( 2 \) with terms \( \texttt{t} \) and \( \texttt{f} \). In class, we defined the grammar and typing rules for expression contexts \( C \). Recall the following definitions:

Definition 3.1 (Kleene equivalence). Two closed terms \( \cdot \vdash M, N : 2 \) are Kleene equivalent, \( M \simeq N \), if \( (M \mapsto^* \texttt{t}) \iff (N \mapsto^* \texttt{t}) \), and \( (M \mapsto^* \texttt{f}) \iff (N \mapsto^* \texttt{f}) \).

Definition 3.2 (Contextual equivalence). Two terms \( \Delta; \Gamma \vdash M, N : \tau \) are contextually equivalent, \( M \bowtie N \), if, for any expression context \( C : (\Delta, \Gamma \triangleright \tau) \rightarrow (\cdot, \cdot \triangleright 2), C[M] \simeq C[N] \).

Because contextual equivalence quantifies over all expression contexts, it requires some machinery to prove terms are contextually equivalent. However, it is straightforward to show two terms are not contextually equivalent.

Task 4. Show that the closed terms \( \lambda x : 2. x \) and \( \lambda x : 2. ((\lambda y : 2. x) \texttt{t}) \) are not contextually equivalent.

A clever choice of expression contexts can also let us prove some basic results about contextually equivalent terms.

Task 5. For any two terms \( \cdot \vdash M, N : \tau \), if \( M \) terminates and \( M \bowtie N \), then \( N \) terminates.

To get a better handle on contextual equivalence, we then defined a type-directed notion of logical equivalence, which it turns out coincides with contextual equivalence.

Definition 3.3 (Logical equivalence of closed terms). To each closed type \( \tau \) we associate a binary relation \( \llbracket \tau \rrbracket \) on closed terms of that type:

- \( (M, N) \in \llbracket 2 \rrbracket \) if \( M, N \) terminate, and \( M \simeq N \).
- \( (M, N) \in \llbracket \tau_1 \rightarrow \tau_2 \rrbracket \) if \( M, N \) terminate, and for any \( (M', N') \in \llbracket \tau_1 \rrbracket \), \( (M M', N N') \in \llbracket \tau_2 \rrbracket \).

Task 6. Show that the two closed terms \( \lambda x : 2. x \) and \( \lambda x : 2. ((\lambda y : 2. x) \texttt{t}) \) are logically equivalent at type \( 2 \rightarrow 2 \).
Finally, we extended this definition to open terms $\Delta; \Gamma \vdash M, N : \tau$ by saying that these terms are logically equivalent if, for any assignment of splendid binary relations to each type variable in $\Delta$, and two closing substitutions $\gamma, \gamma'$ for $\Gamma$ which are elementwise related at their types—interpreting the type variables as the chosen relations—$M$ and $N$ are related by the semantics of $\tau$.

**Task 7.** *Show that the two open terms*

\[
\therefore f : 2 \to 2 \vdash f : 2 \to 2
\]

\[
\therefore f : 2 \to 2 \vdash \lambda x : 2. f x : 2 \to 2
\]

*are logically equivalent.*