1 Let-polymorphism [30 points]

The following shows the abstract syntax for the let-polymorphism system:

monotype\[A ::= A \to A | \alpha\]
polytype\[U ::= A | \forall \alpha. U\]
expression\[e ::= x | \lambda x : A. e | e e | \Lambda \alpha. e | e \llbracket A \rrbracket | \text{let } x : U = e \text{ in } e\]

Question 1. [30 points] Give a well-typed closed expression \(e\) in the let-polymorphism system such that \(\text{erase}(e) = \text{let } f = \lambda x. x \text{ in } (f \, \text{true}, f \, \text{0})\). Assume two monotypes \text{bool} for boolean values and \text{int} for integers.

\[
\text{let } f : \forall \alpha. \alpha \to \alpha = \Lambda \alpha. \lambda x : \alpha. x \text{ in } (f \llbracket \text{bool} \rrbracket \, \text{true}, f \llbracket \text{int} \rrbracket \, 0)
\]

2 Type reconstruction algorithm \(W\) [70 points]

In this problem, we examine the type reconstruction algorithm \(W\) for the implicitly let-polymorphic \(\lambda\)-calculus:

monotype\[A ::= A \to A | \alpha\]
polytype\[U ::= A | \forall \alpha. U\]
expression\[e ::= x | \lambda x : A. e | e e | \Lambda \alpha. e | e \llbracket A \rrbracket | \text{let } x : U = e \text{ in } e\]
typing context\[\Gamma ::= \cdot | \Gamma, x : U\]
type substitution\[S ::= \text{id} | \{A/\alpha\} | S \circ S\]
type equations\[E ::= \cdot | E, A = A\]

The type reconstruction algorithm, called \(W\), takes a typing context \(\Gamma\) and an expression \(e\) as input, and returns a pair of a type substitution \(S\) and a monotype \(A\) as output:

\(W(\Gamma, e) = (S, A)\)

A type substitution is a mapping from type variables to monotypes. Two auxiliary functions \(ftv(A)\) and \(ftv(\Gamma)\) denote the set of free type variables in \(A\) and \(\Gamma\), respectively:

\[
\begin{align*}
ftv(\cdot) &= \emptyset \\
ftv(\Gamma, x : U) &= ftv(\Gamma) \cup ftv(U)
\end{align*}
\]

Typing rules are:

\[
\begin{align*}
\text{Var:} & \quad \Gamma \vdash x : U \\
\text{App:} & \quad \Gamma \vdash e : A \to B, \Gamma \vdash \lambda x : A. e : A \to B \to \Gamma \vdash e : A \\
\text{Let:} & \quad \Gamma \vdash e : U, \Gamma \vdash x : U \vdash e' : A \\
\text{Gen:} & \quad \Gamma \vdash e : \forall \alpha. U \\
\text{Spec:} & \quad \Gamma \vdash e : \llbracket A/\alpha \rrbracket U
\end{align*}
\]
**Question 2. [30 pts]** Complete the definition of the application of a type substitution on types:

\[
\begin{align*}
\text{id} \cdot U &= U \\
\{A/\alpha\} \cdot \alpha &= A \\
\{A/\alpha\} \cdot \beta &= \beta \quad \text{where } \alpha \neq \beta \\
\{A/\alpha\} \cdot B_1 \rightarrow B_2 &= \{A/\alpha\} B_1 \rightarrow \{A/\alpha\} B_2 \\
\{A/\alpha\} \cdot \forall \alpha. U &= \forall \alpha. U \\
\{A/\alpha\} \cdot \forall \beta. U &= \forall \beta. \{A/\alpha\} U \quad \text{where } \alpha \neq \beta \text{ and } \beta \notin \text{ftv}(A) \\
S_1 \cdot S_2 \cdot U &= S_1 \cdot (S_2 \cdot U)
\end{align*}
\]

**Question 3. [20 pts]** Here are a few examples of the type reconstruction. Fill in the blanks.

\[
\begin{align*}
W(x : \alpha, x + 0) &= ((\text{int}/\alpha), \text{int}) \\
W(\_, \lambda x. x + 0) &= ((\text{int}/\alpha), \text{int} \rightarrow \text{int}) \\
W(\_, \lambda x. x) &= (\text{id}, \alpha \rightarrow \alpha)
\end{align*}
\]

**Question 4. [20 pts]** If the algorithm \(W\) is correct, the result of \(W(\Gamma, e)\) should be related with the above typing rules. We refer to this property as **soundness**. State the soundness theorem of the algorithm \(W\). Use the typing judgment of the form \(\Gamma \vdash x : U\) in your statement.

(Soundness of \(W\)). If \(W(\Gamma, e) = (S, A)\),

then \(S \cdot \Gamma \vdash e : A\)