Harvard School of Engineering and Applied Sciences — CS 152: Programming Languages

Assignment 4

Name: Sharanya Pulapura
HUID: 50983277
Collaborators: None

Make sure that the remaining pages of this assignment do not contain any identifying information.

1 Recursive types (25 points)

(a)

$$\textbf{intlist} = \mu X.(\textbf{unit} + (\textbf{int} \times X))$$

$$\begin{aligned}\mathsf{nil} &= \lambda x\!:\!\textbf{unit}.\ \mathsf{roll}_{\mu X.(\textbf{unit}+(\textbf{int}\times X))}\ \mathsf{inl}_{\textbf{unit}+\textbf{int}\times\mu X.(\textbf{unit}+(\textbf{int}\times X)}\ x\\ \mathsf{cons} &= \lambda x\!:\!\textbf{int}.\ \lambda l\!:\!\textbf{intlist}.\ \mathsf{roll}_{\mu X.(\textbf{unit}+(\textbf{int}\times X))}\ \mathsf{inr}_{\textbf{unit}+\textbf{int}\times\mu X.(\textbf{unit}+(\textbf{int}\times X)}\ (x, l)\end{aligned}$$

In both of these cases, we roll up a projection of a sum type, where the first type in the sum is unit, and the second type is a product type of an int and an intlist (where the intlist is typed as $\mu X.(\textbf{unit} + (\textbf{int} \times X))$. When we take the inl or inr of our input which is either unit or an (int, intlist) pair, it then has the sum type $\textbf{unit} + (\textbf{int} \times \mu X.(\textbf{unit} + (\textbf{int} \times X)))$, or $\textbf{unit} + (\textbf{int} \times \textbf{intlist})$. We then roll that into the recursive type $\mu X.(\textbf{unit} + (\textbf{int} \times X))$ which is the given type of intlist.

(b)

$$\begin{aligned}\mathsf{fix}_{\tau_1,\tau_2} = {} & \lambda f\!:\!(\tau_1 \to \tau_2) \to (\tau_1 \to \tau_2).\\ & (\lambda x\!:\!\mu X.X \to \tau_1 \to \tau_2.\ \lambda a\!:\!\tau_1.\\ & \quad f\ (\mathsf{unroll}_{\mu X.X\to\tau_1\to\tau_2}\ x\ x)\ a)\\ & \qquad (\mathsf{roll}_{\mu X.X\to\tau_1\to\tau_2}\ \lambda x\!:\!\mu X.X \to \tau_1 \to \tau_2.\ \lambda a\!:\!\tau_1.\ f\ (\mathsf{unroll}_{\mu X.X\to\tau_1\to\tau_2}\ x\ x)\ a).\end{aligned}$$

If x has type $\mu X.X \to \tau_1 \to \tau_2$, then if we unroll it has type $\mu X.X \to \tau_1 \to \tau_2 \to \tau_1 \to \tau_2$. If we apply x, then we get a return type of $\tau_1 \to \tau_2$. Then if we do $f(\mathsf{unroll}\ \ x\ x)a$, we have a function that takes in $\tau_1 \to \tau_2 \to \tau_1$ and returns τ_2. Therefore the overall type of $\lambda x : \mu X.X \to \tau_1 \to \tau_2.\ \lambda a : \tau_1.\ f\ (\mathsf{unroll}_{\mu X.X\to\tau_1\to\tau_2}\ x\ x)\ a)$ is $\mu X.X \to \tau_1 \to \tau_2 \to \tau_1 \to \tau_2$. This is the type of the expression in lines 2-3 above. If we roll that expression (i.e. as in line 4) we get a type of $\mu X.X \to \tau_1 \to \tau_2$. Applying that to the result of lines 2-3, we get a return type of $\tau_1 \to \tau_2$. So overall our function takes in f of type $(\tau_1 \to \tau_2) \to (\tau_1 \to \tau_2)$ and returns $\tau_1 \to \tau_2$ as desired.

(c) We can define the function F_{sum} as $\lambda f : \textbf{intlist} \to \textbf{int}.\ \lambda x : \textbf{intlist}.\ \mathsf{case}\ \ x\ \mathsf{of}\ \ 0 \mid (\#1\ x) + f\ (\#2\ x)$. Given a function and an intlist, if the intlist has type nil then we return 0, otherwise, if it has type (int x intlist) we take the first projection (the int) and add it to the function applied to the rest of the list (the intlist). sum is the fixed point of F_{sum}. To find the fixed point of this function, then, we just apply the fixpoint function from the previous question as fix F_{sum}.

2 Existential and universal (25 points)

(a) $$\frac{}{\mathsf{hd}\ (\mathsf{nil}\ [\tau]) \longrightarrow v}\ \text{v is type } \tau$$

$$\frac{}{\mathsf{tl}\ (\mathsf{nil}\ [\tau]) \longrightarrow \mathsf{nil}\ [\tau]}$$

This means that any value of type τ is acceptable as the head of an empty τ list, and an empty τ list is the tail of an empty τ list. This ensures that the head of a τ list is always type τ, and the tail of a τ list is always type τ list.

(b)

$$\begin{aligned}\mathsf{let}\ PolyStack = \Lambda X.\,\mathsf{pack}\ \{X\ \textbf{list}, \{&\mathsf{empty}\!:\!\mathsf{nil}\ [X],\\ &\mathsf{push}\!:\!\lambda x\!:\!X.\ \lambda y\!:\!X\ \textbf{list}.\ \mathsf{cons}\ x\ y,\\ &\mathsf{peek}\!:\!\lambda x\!:\!X\ \textbf{list}.\ \mathsf{hd}\ x,\\ &\mathsf{pop}\!:\!\lambda x\!:\!X\ \textbf{list}.\ \mathsf{tl}\ x,\\ &\mathsf{isempty}\!:\!\lambda x\!:\!X\ \textbf{list}.\ \mathsf{isnil}\ x\\ \}\}\ &\mathsf{as}\\ \exists Y.\ \{&\mathsf{empty}\!:\!Y, \mathsf{push}\!:\!X \to Y \to Y, \mathsf{peek}\!:\!Y \to X, \mathsf{pop}\!:\!Y \to Y, \mathsf{isempty}\!:\!Y \to \textbf{bool}\}\end{aligned}$$

(c) This recursive function loops and keeps removing the head of the input intlist and pushing that int to the stack until the intlist is empty. Then it returns the top element of the stack. For the given list, the output is 3.

(d) This recursive function loops and keeps removing the head of the input intlist and pushing that into to the stack until the intlist is empty, just like in the previous question. However, it instead returns a stack with everything but the top element, which is (cons 2, (cons 1 (nil[int]))).

3 Type Inference (15 points)

$$
\begin{aligned}
\tau &::= X \mid \tau_1 \to \tau_2 \mid \tau_1 \times \tau_2 \mid \textbf{int} \mid \textbf{bool} \\
e &::= x \mid \lambda x\!:\!\tau.\, e \mid e_1\ e_2 \\
&\mid (e_1, e_2) \mid \#1\ e \mid \#2\ e \\
&\mid n \mid e_1 + e_2 \mid e_1 * e_2 \mid e_1 - e_2 \\
&\mid \mathsf{true} \mid \mathsf{false} \mid e_1 = e_2 \mid \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \\
&\mid \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 \mid \mathsf{letrec}\ f = \lambda x\!:\!\tau.\, e_1\ \mathsf{in}\ e_2
\end{aligned}
$$

(a)

$$\text{CT-PAIR}\ \frac{\Gamma \vdash e_1\!:\!\tau_1 \triangleright C_1 \quad \Gamma \vdash e_2\!:\!\tau_2 \triangleright C_2}{\Gamma \vdash (e_1, e_2)\!:\!\tau_1 \times \tau_2 \triangleright C_1 \cup C_2}$$

$$\text{CT-LPROJ}\ \frac{\Gamma \vdash e\!:\!\tau \triangleright C \quad C' = C \cup \{\tau \equiv X \times Y\}}{\Gamma \vdash \#1\ e\!:\!X \triangleright C'}\ \text{X, Y are fresh}$$

$$\text{CT-RPROJ}\ \frac{\Gamma \vdash e\!:\!\tau \triangleright C \quad C' = C \cup \{\tau \equiv X \times Y\}}{\Gamma \vdash \#2\ e\!:\!Y \triangleright C'}\ \text{X, Y are fresh}$$

(b)

$$\text{CT-IF}\ \frac{\Gamma \vdash e_1\!:\!\tau_1 \triangleright C_1 \quad \Gamma \vdash e_2\!:\!\tau_2 \triangleright C_2 \quad \Gamma \vdash e_3\!:\!\tau_3 \triangleright C_3}{\Gamma \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3\!:\!\tau_2 \triangleright C_1 \cup C_2 \cup C_3 \cup \{\tau_1 \equiv \textbf{bool}, \tau_2 \equiv \tau_3\}}$$

$$\text{CT-EQUALS}\ \frac{\Gamma \vdash e_1\!:\!\tau_1 \triangleright C_1 \quad \Gamma \vdash e_2\!:\!\tau_2 \triangleright C_2}{\Gamma \vdash e_1 = e_2\!:\!\textbf{bool} \triangleright C_1 \cup C_2 \cup \{\tau_1 \equiv \tau_2\}}$$

(c)

$$\text{CT-LET}\ \frac{\Gamma \vdash e_1\!:\!\tau_1 \triangleright C_1 \quad \Gamma, x\!:\!\tau_1 \vdash e_2\!:\!\tau_2 \triangleright C_2}{\Gamma \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2\!:\!t_2 \triangleright C_1 \cup C_2}$$

$$\text{CT-LETREC}\ \frac{\Gamma, x\!:\!\tau \vdash e_1\!:\!\tau_1 \triangleright C_1 \quad \Gamma, f\!:\!\tau \to \tau_1 \vdash e_2\!:\!\tau_2 \triangleright C_2}{\Gamma \vdash \mathsf{letrec}\ f = \lambda x\!:\!\tau.\, e_1\ \mathsf{in}\ e_2\!:\!t_2 \triangleright C_1 \cup C_2}$$

4 Implementing Type Inference (35 points)

I followed instructions and submitted `check.ml`.