Type Systems for Programming Languages

Assessed Course Work

Hand-out November 21 2011, Hand-in December 7 2011

Question Show that ML-substitution is a sound operation in the ML type assignment system. You can use $(S\phi)[(S\sigma)/\phi] = S(\phi[\sigma/\phi])$, and can assume that free and bound (by for-all) type-variables are distinct.

(This is the only time I will ask for a proof.)

Answer: To show: If $B \vdash_{\text{ML}} E : \psi$, then, for every substitution S, $SB \vdash_{\text{ML}} E : S\psi$. We prove this by induction on the structure of derivations:

(Ax) : Then $E \equiv x$, and $x{:}\psi \in B$. Then $x{:}S\psi \in SB$, so $SB \vdash_{\text{ML}} x : S\psi$, by rule (Ax).

$(\to I)$: Then $E \equiv \lambda x.E'$, there are $\sigma, \tau \in \mathcal{T}_{\text{C}}$ such that $\psi = A \to B$, and $B, x{:}\sigma \vdash_{\text{ML}} E' : \tau$. By induction, $S(B, x{:}\sigma) \vdash_{\text{ML}} E' : S\tau$. Since $S(B, x{:}\sigma) = SB, x{:}S\sigma$, also $SB, x{:}S\sigma \vdash_{\text{ML}} E' : S\tau$. Then, by rule $(\to I)$, we obtain $SB \vdash_{\text{ML}} \lambda x.E' : S\sigma \to S\tau$. Since $S\sigma \to S\tau = S(A \to B)$, and $S(A \to B) = S\psi$, we get $SB \vdash_{\text{ML}} \lambda x.E' : S\psi$.

$(\to E)$: Then $E \equiv E_1 E_2$, and there exists $\sigma \in \mathcal{T}_{\text{C}}$ such that $B \vdash_{\text{ML}} E_1 : \sigma \to \psi$, and $B \vdash_{\text{ML}} E_2 : \sigma$ (also $\psi \in \mathcal{T}_{\text{C}}$). Then, by induction, $SB \vdash_{\text{ML}} E_1 : S(\sigma \to \psi)$ and $SB \vdash_{\text{ML}} E_2 : S\sigma$. Since $S(\sigma \to \psi) = S\sigma \to S\psi$, so we also have $SB \vdash_{\text{ML}} E_1 : S\sigma \to S\psi$, and we can apply rule $(\to E)$ to obtain $SB \vdash_{\text{ML}} E_1 E_2 : S\psi$.

$(\texttt{Fix})$: Then $E \equiv \texttt{Fix}\ g.E$, $\psi \in \mathcal{T}_{\text{C}}$, and $B, g{:}\psi \vdash_{\text{ML}} E : \psi$. Then $S(B, g{:}\psi) \vdash_{\text{ML}} E : S\psi$ by induction, and since $S(B, g{:}\psi) = SB, g{:}S\psi$, we also have $SB, Sg{:}\psi \vdash_{\text{ML}} E : S\psi$. So, by rule $(\texttt{Fix})$, also $SB \vdash_{\text{ML}} \texttt{Fix}\ g.E : S\psi$.

$(\forall I)$: Then $\psi = \forall\phi.\chi$, and $B \vdash_{\text{ML}} E : \chi$, and we know that ϕ does not occur free in B. By induction, $SB \vdash_{\text{ML}} E : S\chi$. Without loss of generality, we assume that free and bound type variables are distinct, so can assume that ϕ does not occur in the range of S, so does not occur free in SB. So, by rule $(\forall I)$, $SB \vdash_{\text{ML}} E : \forall\phi.S\chi$. Since $\forall\phi.S\chi = S(\forall\phi.\chi) = S\psi$, we get $SB \vdash_{\text{ML}} E : S\psi$.

$(\forall E)$: Then $\psi = \chi[\sigma/\phi]$, and $B \vdash_{\text{ML}} E : \forall\phi.\chi$, and, by induction, $SB \vdash_{\text{ML}} E : S\forall\phi.\chi$. Then, by rule $(\forall E)$, also $SB \vdash_{\text{ML}} E : (S\chi)[(S\sigma)/\phi]$. Without loss of generality, we can assume that ϕ does not occur in the domain of S, so if it occurs in χ, then it occurs in $S\chi$. So $(S\chi)[(S\sigma)/\phi] = S(\chi[\sigma/\phi]) = S\psi$, and we get $SB \vdash_{\text{ML}} E : S\psi$.

Question Extend the Lambda Calculus with the *conditional* language construct '$\texttt{if}\ M\ \texttt{then}\ P\ \texttt{else}\ Q$'. Add any other constant or construct that might be needed to make this work correctly and extend, if needed, the notion of reduction to deal with this addition.

Answer: We need to add the boolean expressions *True* and *False* to the set of terms, and the reduction rules

$$\begin{aligned} \texttt{if}\ \textit{True}\ \texttt{then}\ M\ \texttt{else}\ N &\to M \\ \texttt{if}\ \textit{False}\ \texttt{then}\ M\ \texttt{else}\ N &\to N \end{aligned}$$

as well as the contextual rules

$$P \to Q \ \Rightarrow\ \begin{cases} \texttt{if}\ P\ \texttt{then}\ M\ \texttt{else}\ N \to \texttt{if}\ Q\ \texttt{then}\ M\ \texttt{else}\ N \\ \texttt{if}\ M\ \texttt{then}\ P\ \texttt{else}\ N \to \texttt{if}\ M\ \texttt{then}\ Q\ \texttt{else}\ N \\ \texttt{if}\ M\ \texttt{then}\ N\ \texttt{else}\ P \to \texttt{if}\ M\ \texttt{then}\ N\ \texttt{else}\ Q \end{cases}$$

Give the natural extension to Curry's type assignment system, by presenting -if necessary- new types and rules, for the expressions that deal with this new construct.

Answer: We add the type constant *Bool*, as well as the type assignment rules:

$$\frac{}{\Gamma \vdash \mathit{True} : \mathit{Bool}} \qquad \frac{}{\Gamma \vdash \mathit{False} : \mathit{Bool}} \qquad \frac{\Gamma \vdash P : \mathit{Bool} \quad \Gamma \vdash M_1 : \sigma \quad \Gamma \vdash M_2 : \sigma}{\Gamma \vdash \texttt{if } P \texttt{ then } M_1 \texttt{ else } M_2 : \sigma}$$

Extend the principal context algorithm with a case for the new construct.

Answer: The extension to pp_{C} is:

$$\begin{array}{rcl} pp_{\mathrm{C}}\,(\texttt{if } M \texttt{ then } P \texttt{ else } Q) & = & S_4 \circ S_3 \circ S_2 \circ S_1\,\langle \Gamma_1 \cup \Gamma_2 \cup \Gamma_3, \sigma \rangle \\ \text{where } \langle \Gamma_1, \rho \rangle & = & pp_{\mathrm{C}}\ M \\ \langle \Gamma_2, \tau \rangle & = & pp_{\mathrm{C}}\ P \\ \langle \Gamma_3, \sigma \rangle & = & pp_{\mathrm{C}}\ Q \\ S_1 & = & \mathit{unify}\ \rho\ \mathit{Bool} \\ S_2 & = & \mathit{unify}\ (S_1\,\tau)\ (S_1\,\sigma) \\ S_3 & = & \mathit{UnifyContexts}\ (S_2 \circ S_1\,\Gamma_1)\ (S_2 \circ S_1\,\Gamma_2) \\ S_3 & = & \mathit{UnifyContexts}\ (S_3 \circ S_2 \circ S_1\,\Gamma_3)\ (S_3 \circ S_2 \circ S_1\,\Gamma_1 \cup \Gamma_2) \end{array}$$

Question Assume the existence of characters, numbers, test for zero and (pre-fix) addition, as well as the abstract data type *List* in the ML language. Let $[\phi] = \mathit{List}\ \phi$, *Tail* be a function of type $\forall \phi.[\phi] \to [\phi]$, and '$=[\,]\ l$' be a test if l is the empty list. Build the ML expression that calculates the length of a list, and build a derivation that types this ML expression.

The derived type for length might be small, but notice that you would, formally, need to repeat the WHOLE derivation when you use length. Instead, you can abbreviate that (and write just the conclusion in the bottom line) by D.

Answer:

$$\mathit{Length} \ = \ \texttt{Fix}\ g.\lambda l.\texttt{if}\ (=[\,]\ l)\ \texttt{then}\ 0\ \texttt{else} + 1\ (g\ (\mathit{Tail}\ l))$$

Let $\Gamma = g{:}[\phi] \to \mathit{Int}, l{:}[\phi]$, and take D to be

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash_{\mathrm{ML}} +{:}\,\mathit{Int} \to \mathit{Int} \to \mathit{Int}} \quad \dfrac{}{\Gamma \vdash_{\mathrm{ML}} 1{:}\,\mathit{Int}}}{\Gamma \vdash_{\mathrm{ML}} +\,1{:}\,\mathit{Int} \to \mathit{Int}} \quad \dfrac{\dfrac{}{\Gamma \vdash_{\mathrm{ML}} g{:}\,[\phi] \to \mathit{Int}} \quad \dfrac{\dfrac{}{\Gamma \vdash_{\mathrm{ML}} \mathit{Tail}{:}\,[\phi] \to [\phi]} \quad \dfrac{}{\Gamma \vdash_{\mathrm{ML}} l{:}\,[\phi]}}{\Gamma \vdash_{\mathrm{ML}} \mathit{Tail}\ l{:}\,[\phi]}}{\Gamma \vdash_{\mathrm{ML}} g\ (\mathit{Tail}\ l){:}\,\mathit{Int}}}{\Gamma \vdash_{\mathrm{ML}} +\,'a'\ (g\ (\mathit{Tail}\ l)){:}\,\mathit{Int}}$$

then

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{\Gamma \vdash_{\mathrm{ML}} =[\,] : [\phi]\to Bool} \quad \overline{\Gamma \vdash_{\mathrm{ML}} l : [\phi]}}{\Gamma \vdash_{\mathrm{ML}} =[\,]\ l : Bool} \quad \overline{\Gamma \vdash_{\mathrm{ML}} 0 : Int} \quad \dfrac{D}{\Gamma \vdash_{\mathrm{ML}} +\ {}'a'\ (g\ (Tail\ l)) : Int}}{\Gamma \vdash_{\mathrm{ML}} \mathtt{if}\ (=[\,]\ l)\ \mathtt{then}\ 0\ \mathtt{else} +\ {}'a'\ (g\ (Tail\ l)) : Int}}{g{:}([\phi]\to Int \vdash_{\mathrm{ML}} \lambda l.\mathtt{if}\ (=[\,]\ l)\ \mathtt{then}\ 0\ \mathtt{else} + 1\ (g\ (Tail\ l)) : [\phi]\to Int}}{\vdash_{\mathrm{ML}} \mathtt{Fix}\ g.\lambda l.\mathtt{if}\ (=[\,]\ l)\ \mathtt{then}\ 0\ \mathtt{else} + 1\ (g\ (Tail\ l)) : [\phi]\to Int}}{\vdash_{\mathrm{ML}} \mathtt{Fix}\ g.\lambda l.\mathtt{if}\ (=[\,]\ l)\ \mathtt{then}\ 0\ \mathtt{else} + 1\ (g\ (Tail\ l)) : \forall\phi.[\phi]\to Int}$$

Question Assume that *Cons* has type $\forall\phi.\phi\to[\phi]\to[\phi]$; using your answer to the previous question, give a derivation for the (translation of the) expression

$$Length\ (Cons('a', Cons('b', [\,]))) + Length\ (Cons(1, Cons(2, Cons(3, [\,]))))$$

that types this ML-expression (do not forget `let!`).

Notice that length is used on objects of different type here: i.e. polymorphic, so this calls for the use of `let`. It is possible to type it without, but that would breach the idea of polymorphism, since you would be specifying length twice.

I want to see the whole derivation (just this once).

Answer: The expression translates to

$$\begin{array}{l}\mathtt{let}\ l\ =\ \mathtt{Fix}\ g.\lambda l.\mathtt{if}\ (=[\,]\ l)\ \mathtt{then}\ 0\ \mathtt{else}\ (+\ 1\ (g\ (Tail\ l)))\ \ \mathtt{in}\\ \qquad +\ (l\ (Cons\ {}'a'\ (Cons\ {}'b'\ [\,])))\ (l\ (Cons\ {}'a'\ (Cons\ 2\ (Cons\ 3\ [\,]))))\end{array}$$

Let D_1 be the derivation from the previous exercise, $D_2 =$

$$\dfrac{\dfrac{\dfrac{\overline{\vdash Cons : \forall\phi.\phi\to[\phi]\to[\phi]}}{\vdash Cons : Ch\to[Ch]\to[Ch]} \quad \overline{\vdash {}'a' : Ch}}{\vdash Cons\ {}'a' : [Ch]\to[Ch]} \quad \dfrac{\dfrac{\overline{\vdash Cons : \forall\phi.\phi\to[\phi]\to[\phi]}}{\vdash Cons : Ch\to[Ch]\to[Ch]} \quad \overline{\vdash {}'b' : Ch}}{\vdash Cons\ {}'b' : [Ch]\to[Ch]} \quad \dfrac{\overline{\vdash [\,] : \forall\phi.[\phi]}}{\vdash [\,] : [Ch]}}{\vdash Cons\ {}'b'\ [\,] : [Ch]}}{\vdash Cons\ {}'a'\ (Cons\ {}'b'\ [\,]) : [Ch]}$$

and $D_3 =$

$$\dfrac{\dfrac{\dfrac{\overline{\vdash Cons : \forall\phi.\phi\to[\phi]\to[\phi]}}{\vdash Cons : I\to[I]\to[I]} \quad \overline{\vdash 1 : I}}{\vdash Cons\ 1 : [I]\to[I]} \quad \dfrac{\dfrac{\overline{\vdash Cons : \forall\phi.\phi\to[\phi]\to[\phi]}}{\vdash Cons : I\to[I]\to[I]} \quad \overline{\vdash 2 : I}}{\vdash Cons\ 2 : [I]\to[I]} \quad \dfrac{\dfrac{\dfrac{\overline{\vdash Cons : \forall\phi.\phi\to[\phi]\to[\phi]}}{\vdash Cons : I\to[I]\to[I]} \quad \overline{\vdash 3 : I}}{\vdash Cons\ 3 : [I]\to[I]} \quad \dfrac{\overline{\vdash [\,] : \forall\phi.[\phi]}}{\vdash [\,] : [I]}}{\vdash Cons\ 3\ [\,] : [I]}}{\vdash Cons\ 2\ (Cons\ 3\ [\,]) : [I]}}{\vdash Cons\ 1\ (Cons\ 2\ \ (Cons\ 3\ [\,])) : [I]}$$

Take

$$\begin{array}{lcl} Length & = & \mathtt{Fix}\ g.\lambda l.\mathtt{if}\ (=[\,]\ l)\ \mathtt{then}\ 0\ \mathtt{else}\ (+\ 1\ (g\ (Tail\ l))),\\ L_1 & = & Cons\ {}'a'\ (Cons\ {}'b'\ [\,])\\ L_2 & = & Cons\ 1\ (Cons\ 2\ \ (Cons\ 3\ [\,]))\\ \psi & = & \forall\phi.[\phi]\to I \end{array}$$

Then the derivation becomes:

$$\dfrac{\dfrac{}{l{:}\psi \vdash + : I \to I \to I} \quad \dfrac{\dfrac{\dfrac{}{l{:}\psi \vdash l : \psi}}{l{:}\psi \vdash l : [Ch] \to I} \quad \dfrac{}{l{:}\psi \vdash L_1 : [Ch]}}{l{:}\psi \vdash l\ L_1 : I}}{l{:}\psi \vdash +\ (l\ L_1) : I \to I} \quad \dfrac{\dfrac{\dfrac{}{l{:}\psi \vdash l : \psi}}{l{:}\psi \vdash l : [I] \to I} \quad \dfrac{D_3}{l{:}\psi \vdash L_2 : [I]}}{l{:}\psi \vdash l\ L_2 : I}}{l{:}\psi \vdash +\ (l\ L_1)\ (l\ L_2) : I} \quad \dfrac{D_1}{\vdash \mathit{Length} : \psi}$$
$$\vdash \mathtt{let}\ l = \mathit{Length}\ \mathtt{in} + (l\ L_1)\ (l\ L_2) : I$$

Question Show that $(\lambda xy.y(xxy))(\lambda xy.y(xxy))$ is a fixed-point constructor for the Lambda Calculus.

Answer: $$\begin{array}{rl}(\lambda xy.y(xxy))(\lambda xy.y(xxy))\,M & \to_\beta (\lambda y.y((\lambda xy'.y'(xxy'))(\lambda xy'.y'(xxy'))y))\,M \\ & \to_\beta M((\lambda xy.y(xxy))(\lambda xy.y(xxy))\,M)\end{array}$$

Extending ML with equi-recursive types, and using the recursive type $B = \mu X.X \to (\phi \to \phi) \to \phi$, show that $(\lambda xy.y(xxy))(\lambda xy.y(xxy))$ is typeable with $(A \to A) \to A$, for all A. (Hint: first derive $\vdash \lambda xy.y(xxy) : B \to (\phi \to \phi) \to \phi$).

Answer: Take $\Gamma = x{:}B, y{:}\phi \to \phi$, then we derive first

$$\mathcal{D} :: \dfrac{\dfrac{\dfrac{}{\Gamma \vdash y : \phi \to \phi} \quad \dfrac{\dfrac{\dfrac{}{\Gamma \vdash x : A}}{\Gamma \vdash x : A \to (\phi \to \phi) \to \phi} \quad \dfrac{}{\Gamma \vdash x : A}}{\Gamma \vdash xx : (\phi \to \phi) \to \phi} \quad \dfrac{}{\Gamma \vdash y : \phi \to \phi}}{\Gamma \vdash xxy : \phi}}{\Gamma \vdash y(xxy) : \phi}}{x{:}B \vdash \lambda y.y(xxy) : (\phi \to \phi) \to \phi}}{\vdash \lambda xy.y(xxy) : B \to (\phi \to \phi) \to \phi}$$

now using that $B = \mu X.X \to (\phi \to \phi) \to \phi = (\mu X.X \to (\phi \to \phi) \to \phi) \to (\phi \to \phi) \to \phi = B \to (\phi \to \phi) \to \phi$, also

$$\dfrac{\dfrac{\dfrac{\dfrac{\boxed{\mathcal{D}}}{\vdash \lambda xy.y(xxy) : B \to (\phi \to \phi) \to \phi} \quad \dfrac{\boxed{\mathcal{D}}}{\vdash \lambda xy.y(xxy) : B \to (\phi \to \phi) \to \phi}}{\vdash \lambda xy.y(xxy) : B}}{\vdash (\lambda xy.y(xxy))(\lambda xy.y(xxy)) : (\phi \to \phi) \to \phi}}{\vdash (\lambda xy.y(xxy))(\lambda xy.y(xxy)) : \forall \phi.(\phi \to \phi) \to \phi}}{\vdash (\lambda xy.y(xxy))(\lambda xy.y(xxy)) : (A \to A) \to A}$$