CS520, Handout 19, 2 Nov 2006

Type Reconstruction (Continued)

Reading: [Pierce, Chapter 22]

Let-Polymorphism [Pierce, Sect 22.7]

Using the rule T-LET from [Pierce, Fig 11-4]:

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \texttt{let}\ x = t_1\ \texttt{in}\ \ t_2 : T_2} \qquad \text{(T-LET)}$$

each of the following is a well-typed program fragment:

```
(* fragment 1 *)
  let double = lambda f:Nat->Nat. lambda a:Nat. f(f a) in
  double (lambda x:Nat. succ (succ x)) 2;
(* fragment 2 *)
  let double = lambda f:Bool->Bool. lambda a:Bool. f(f a) in
  double (lambda x:Bool. x) false;
(* fragment 3 *)
  let doubleNat = lambda f:Nat->Nat. lambda a:Nat. f(f a) in
  let doubleBool = lambda f:Bool->Bool. lambda a:Bool. f(f a) in
  let p = doubleNat (lambda x:Nat. succ (succ x)) 2 in
  let q = doubleBool (lambda x:Bool. x) false in ...
```

But we can not write the following, if we want to avoid specializing the function `double` twice, once for `Nat` and once for `Bool`:

```
let double = lambda f:X->X. lambda a:X. f(f a) in
let p = double (lambda x:Nat. succ (succ x)) 2 in
let q = double (lambda x:Bool. x) false in ...
```

Even though this is a perfectly safe code.

Constraint typing rule corresponding to T-LET:

$$\frac{\Gamma \vdash t_1 : T_1 \mid C_1 \qquad \Gamma, x : T_1' \vdash t_2 : T_2 \mid C_2}{\Gamma \vdash \texttt{let}\ x = t_1\ \texttt{in}\ \ t_2 : T_2 \mid (C_1 \cup C_2 \cup \{\, T_1 = T_1' \})} \qquad \text{(CT-LET)}$$

Exercise

Using the constraint typing rule CT-LET, show that the program fragment on page 3 is not well-typed.

We can view **let** $x = t_1$ **in** t_2 as sugar for $(\lambda x.\, t_2\,)t_1$ in which case there is no need to use T-LET and CT-LET. We can use instead the following typing rule and constraint typing rule:

$$\frac{\Gamma \vdash (\lambda x : T_1.\, t_2\,) : T_1 \to T_2 \qquad \Gamma \vdash t_1 : T_1}{\Gamma \vdash (\lambda x : T_1.\, t_2\,)t_1 : T_2}$$

$$\frac{\Gamma \vdash (\lambda x : T_1'.\, t_2\,) : T_1' \to T_2 \mid C_1 \qquad \Gamma \vdash t_1 : T_1 \mid C_2 \qquad X \text{ fresh}}{\Gamma \vdash (\lambda x : T_1'.\, t_2\,)t_1 : T_2 \mid (C_1 \cup C_2 \cup \{\, T_1' \to T_2 = T_1 \to X \})}$$

The two preceding rules are not really new. They are derived rules, because they can be expressed using rules introduced earlier. In fact, the first above is just a special of T-APP [Handout 07, p. 3], and the second is a special case of CT-APP [Handout 18, p. 7].

Exercise

Viewing **let** $x = t_1$ **in** t_2 as sugar for $(\lambda x.\, t_2\,)t_1$ and using the constraint typing rule above, show once more that the program fragment on page 3 is not well-typed.

Introduce a typing rule for un-annotated abstraction:

$$\frac{\Gamma,\, x : T \vdash t : T'}{\Gamma \vdash (\lambda x.\, t\,) : T \to T'} \qquad (\text{T-ABS}')$$

and the following adjusted rule for `let`-expressions:

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma \vdash [x \mapsto t_1]t_2 : T_2}{\Gamma \vdash \textbf{let}\ x = t_1\ \textbf{in}\ t_2 : T_2} \qquad (\text{T-LETPOLY})$$

With the two preceding typing rules the following program phrase is well-typed:

```
let double = lambda f. lambda a. f(f a) in
let p = double (lambda x:Nat. succ (succ x)) 2 in
let q = double (lambda x:Bool. x) false in ...
```

Exercise Prove this last assertion, by writing an appropriate typing derivation.

Exercise What goes wrong if we omit the premise "$\Gamma \vdash t_1 : T_1$" in the typing rule (T-LETPOLY)? Justify your answer carefully.

Corresponding to the typing rules (T-ABS′) and (T-LETPOLY) we have the following constraint typing rules:

$$\frac{\Gamma,\, x : T \vdash t : T' \mid C}{\Gamma \vdash (\lambda x.\, t) : T \to T' \mid C} \quad \text{(CT-ABS}'\text{)}$$

$$\frac{\Gamma \vdash t_1 : T_1 \mid C_1 \qquad \Gamma \vdash [x \mapsto t_1]t_2 : T_2 \mid C_2}{\Gamma \vdash \mathtt{let}\ x = t_1\ \mathtt{in}\ t_2 : T_2 \mid C_1 \cup C_2} \quad \text{(CT-LETPOLY)}$$