Recitation 9: Polymorphism

15-312: Principles of Programming Languages

Wednesday, March 19, 2014

Arguably, the only benefit dynamic languages offer is a form of polymorphism that arises from an absence of a type for a function. For example the identity function $\texttt{fun}(\lambda(x.x))$, can be applied to numbers, functions, functions of functions etc. This comes however, at the great cost of the runtime messiness of class-checking. On the other hand in PCF, there is a *distinct* identity function for each type τ, namely $\lambda(x : \tau)x$, even though the behavior is same for each choice of τ. Polymorphic types allow one to write a general pattern once and for all, and instantiate the pattern with a particular type when needed. While polymorphism is motivated by this simple idea, of how to avoid writing redundant code, the concept adds a lot of power to a language, thereby allowing one to define products, sums, integers and much more.

1 System F

Sort			**Abstract Form**	**Concrete Form**
Type	τ	$::=$	t	t
			$\texttt{arr}(\tau_1; \tau_2)$	$\tau_1 \to \tau_2$
			$\texttt{all}(t.\tau)$	$\forall(t.\tau)$
Exp	e	$::=$	x	x
			$\lambda(x.e)$	$\lambda(x : \tau).e$
			$\texttt{ap}(e_1; e_2)$	$e_1\ (e_2)$
			$\texttt{Lam}(t.e)$	$\Lambda(t)\ e$
			$\texttt{App}[\tau](e)$	$e[\tau]$

1.1 Statics

Defining the statics first requires us to define a judgment for the well-formedness of types, given by $\Delta \vdash \tau$ type. These essentially capture scoping rules for type abstraction and are given by the following rules

$$\frac{}{\Delta, t\ \mathsf{type} \vdash t\ \mathsf{type}}(T_1) \qquad \frac{\Delta \vdash \tau_1\ \mathsf{type} \qquad \Delta \vdash \tau_2\ \mathsf{type}}{\Delta \vdash \tau_1 \to \tau_2\ \mathsf{type}}(T_2) \qquad \frac{\Delta, t\ \mathsf{type} \vdash \tau\ \mathsf{type}}{\Delta \vdash \forall(t.\tau)\ \mathsf{type}}(T_3)$$

The rules defining typing judgment carry two sets of hypothesis. The first one Δ, consists of judgments regarding well-formedness of types and the second one Γ is the usual typing context for variables.

$$\frac{}{\Delta\ \Gamma, x : \tau \vdash x : \tau}(T_4) \qquad \frac{\Delta \vdash \tau_1 \textsf{ type} \qquad \Delta\ \Gamma, x : \tau_1 \vdash e : \tau_2}{\Delta\ \Gamma \vdash \lambda(x : \tau_1)\ e : \tau_1 \to \tau_2}(T_5)$$

$$\frac{\Delta\ \Gamma \vdash e_1 : \tau_2 \to \tau \qquad \Delta\ \Gamma \vdash e_2 : \tau_2}{\Delta\ \Gamma \vdash e_1\ e_2 : \tau}(T_6) \qquad \frac{\Delta, t \textsf{ type}\ \ \Gamma \vdash e : \tau}{\Delta\ \Gamma \vdash \Lambda(t)\ e : \forall(t.\tau)}(T_7)$$

$$\frac{\Delta\ \Gamma \vdash e : \forall(t.\tau') \qquad \Delta \vdash \tau \textsf{ type}}{\Delta\ \Gamma \vdash e[\tau] : [\tau/t]\tau'}(T_8)$$

1.2 Examples

The polymorphic identity function I is written as

$$\Lambda(t)\ \lambda(x : \tau)\ e$$

it has the polymorphic type

$$\forall(t.t \to t)$$

Instances of the identity function are written as $I[\tau]$, where τ is some type, and have the type $\tau \to \tau$.

The polymorphic composition function C is written as

$$\Lambda(t_1)\ \Lambda(t_2)\ \Lambda(t_3)\ \lambda(f : t_1 \to t_2)\ \lambda(g : t_2 \to t_3)\ \lambda(x : t_1)\ g(f(x))$$

Task What is the type of C? What is the type of $C[\tau]$?

1.3 Dynamics

The dynamics of System F are given as follows

$$\frac{}{\lambda(x : \tau)\ e \textsf{ val}}(D_1) \qquad \frac{}{\Lambda(t)\ e \textsf{ val}}(D_2) \qquad \frac{e_2 \textsf{ val}}{\lambda(x : \tau_1)\ e\ e_2 \mapsto [e_2/x]e}(D_3) \qquad \frac{e_1 \mapsto e_1'}{e_1\ e_2 \mapsto e_1'\ e_2}(D_4)$$

$$\frac{e_1 \textsf{ val} \qquad e_2 \mapsto e_2'}{e_1\ e_2 \mapsto e_1\ e_2'}(D_5) \qquad \frac{}{\Lambda(t)\ e[\tau] \mapsto [\tau/t]e}(D_6) \qquad \frac{e \mapsto e'}{e[\tau] \mapsto e'[\tau]}(D_7)$$

2 Polymorphic Church Numerals

Back in Gödel's T the motivation behind numbers was a construct which would allow one to iterate up to the number. So, intuitively, a natural number n is a function that takes an expression which is the base case, a function which does one step of work and then applies the function n times to the base case. With this intuition, we can define natural numbers in System F by the following translation.

$$
\begin{aligned}
\texttt{nat} &\triangleq \forall(t.t \rightharpoonup (t \rightharpoonup t) \rightharpoonup t) \\
\texttt{z} &\triangleq \Lambda(t)\ \lambda(b:t)\ \lambda(s:t \rightharpoonup t)\ b \\
\texttt{s}(e) &\triangleq \Lambda(t)\ \lambda(b:t)\ \lambda(s:t \rightharpoonup t)\ s(e[t]\ b\ s) \\
\texttt{iter}(e;e_0;x.e_1) &\triangleq e[\tau]\ e_0\ \lambda(x:\tau)\ e_1
\end{aligned}
$$

We can verify here that

$$\texttt{iter}(\texttt{z};e_0;x.e_1) \equiv e_0$$

$$\texttt{iter}(\texttt{s}(e);e_0;x.e_1) \equiv [\texttt{iter}(e;e_0;x.e_1)/x]e_1$$

This is exactly what is dictated by the dynamics of `nat` in Gödel's T.

3 Lists

The definition of lists of element of type ρ is not very different from the definition of natural numbers. In fact, natural numbers can be seen as isomorphic to lists of `unit`. One implementation of lists in system F is as follows.

$$
\begin{aligned}
\texttt{list}\,\rho &\triangleq \forall(t.t \rightharpoonup (\rho \rightharpoonup t \rightharpoonup t) \rightharpoonup t) \\
\texttt{nil}_\rho &\triangleq \Lambda(t)\ \lambda(n:t)\ \lambda(c:\rho \rightharpoonup t \rightharpoonup t)\ n \\
\texttt{cons}(h;tl)_\rho &\triangleq \Lambda(t)\ \lambda(n:t)\ \lambda(c:\rho \rightharpoonup t \rightharpoonup t)\ c\ h\ (tl[t]\ n\ c) \\
\texttt{fold}(e;e_0;x.y.e_1)_\rho &\triangleq e[\tau]\ e_0\ \lambda(x:\rho)\ \lambda(y:\tau)\ e_1
\end{aligned}
$$