CMSC 336: Type Systems for Programming Languages (Winter 2008)

Homework 4

Out: Tuesday, February 26, 2008
Due: Tuesday, March 4, 2008

1 Polymorphism, Church Encodings

Recall from class System F, or the polymorphic λ-calculus.

$$\tau ::= \alpha \mid \tau_1 \to \tau_2 \mid \forall\alpha.\,\tau$$
$$e ::= x \mid \lambda x{:}\tau.\,e \mid e_1\ e_2 \mid \Lambda\alpha.\,e \mid e\,[\tau]$$

Despite its simplicity, System F is quite expressive. As discussed in class, it has sufficient expressive power to be able to encode many datatypes found in other programming languages, including products, sums, and inductive datatypes.

For example, recall that bool may be encoded as follows:

$$\mathsf{bool} := \forall\alpha.\,\alpha \to \alpha \to \alpha$$
$$\mathsf{true} := \Lambda\alpha.\,\lambda t{:}\alpha.\,\lambda f{:}\alpha.\,t$$
$$\mathsf{false} := \Lambda\alpha.\,\lambda t{:}\alpha.\,\lambda f{:}\alpha.\,f$$

$$\mathsf{if}_\tau\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 := e\,[\tau]\ e_1\ e_2 \qquad \text{(where } \tau \text{ indicates the type of } e_1 \text{ and } e_2\text{)}$$

Exercise 1. Show how to encode the following terms. Note that each of these terms, when applied to the appropriate arguments, return a result of type bool.

(a) the term not that takes an argument of type bool and computes its negation;

(b) the term and that takes two arguments of type bool and computes their conjunction;

(c) the term or that takes two arguments of type bool and computes their disjunction.

The type nat may be encoded as follows:

$$\mathsf{nat} := \forall\alpha.\,\alpha \to (\alpha \to \alpha) \to \alpha$$
$$\mathsf{zero} := \Lambda\alpha.\,\lambda z{:}\alpha.\,\lambda s{:}\alpha \to \alpha.\,z$$
$$\mathsf{succ}\ := \lambda n{:}\mathsf{nat}.\,\Lambda\alpha.\,\lambda z{:}\alpha.\,\lambda s{:}\alpha \to \alpha.\,s\ (n\,[\alpha]\ z\ s)$$

A nat $\overline{n}$ is defined by what it can do, which is to compute a function iterated n times. In the polymorphic encoding above, the result of that iteration can be any type α, as long as you have a base element $z : \alpha$ and a function $s : \alpha \to \alpha$.

Conveniently, this encoding "is" its own elimination form, in a sense:

$$\mathbf{rec}(e,\ e_0,\ x{:}\tau.\,e_1) := e\,[\tau]\ e_0\ (\lambda x{:}\tau.\,e_1)$$

The case analysis is baked into the very definition of the type.

Exercise 2. Verify that these encodings (zero, succ, $\mathbf{rec}$) typecheck in System F. Write down the typing derivations for the terms.

As mentioned in class, System F can express any inductive datatype. Consider the following list type:

```
datatype 'a list =
    Nil
  | Cons of 'a * 'a list
```

We can encode $\tau\,\mathsf{list}$s, lists of elements of type τ as follows:[1]

$$\begin{aligned}\tau\,\mathsf{list} &:= \forall\alpha.\,\alpha \to (\tau \to \alpha \to \alpha) \to \alpha\\ \mathsf{nil}_\tau &:= \Lambda\alpha.\,\lambda n{:}\alpha.\,\lambda c{:}\tau \to \alpha \to \alpha.\,n\\ \mathsf{cons}_\tau &:= \lambda h{:}\tau.\,\lambda t{:}\tau\,\mathsf{list}.\,\Lambda\alpha.\,\lambda n{:}\alpha.\,\lambda c{:}\tau \to \alpha \to \alpha.\,c\ h\ (t\,[\alpha]\ n\ c)\end{aligned}$$

As with nats, The $\tau\,\mathsf{list}$ type's case analyzing elimination form is just application. We can write functions like map:

$$\begin{aligned}&\mathsf{map} : (\sigma \to \tau) \to \sigma\,\mathsf{list} \to \tau\,\mathsf{list}\\ &\quad := \lambda f{:}\sigma \to \tau.\,\lambda l{:}\sigma\,\mathsf{list}.\,l\,[\tau\,\mathsf{list}]\ \mathsf{nil}_\tau\ (\lambda x{:}\sigma.\,\lambda y{:}\tau\,\mathsf{list}.\,\mathsf{cons}_\tau\ (f\ x)\ y\end{aligned}$$

Exercise 3. Consider the following simple binary tree type:

```
datatype 'a tree =
    Leaf
  | Node of 'a tree * 'a * 'a tree
```

(a) Give a System F encoding of binary trees, including a definition of the type $\tau\,\mathsf{tree}$ and definitions of the constructors $\mathsf{leaf} : \tau\,\mathsf{tree}$ and $\mathsf{node} : \tau\,\mathsf{tree} \to \tau \to \tau\,\mathsf{tree} \to \tau\,\mathsf{tree}$.

(b) Write a function $\mathsf{height} : \tau\,\mathsf{tree} \to \mathsf{nat}$. You may assume the above encoding of nat as well as definitions of the functions $\mathsf{plus} : \mathsf{nat} \to \mathsf{nat} \to \mathsf{nat}$ and $\mathsf{max} : \mathsf{nat} \to \mathsf{nat} \to \mathsf{nat}$.

(c) Write a function $\mathsf{in\text{-}order} : \tau\,\mathsf{tree} \to \tau\,\mathsf{list}$ that computes the in-order traversal of a binary tree. You may assume the above encoding of lists; define any auxiliary functions you need.

2 Polymorphism, Existential Types: Type Safety

Consider the polymorphic λ-calculus extended with existential types.

$$\begin{aligned}\tau &::= \alpha \mid \tau_1 \to \tau_2 \mid \forall\alpha.\,\tau \mid \exists\alpha.\,\tau\\ e &::= x \mid \lambda x{:}\tau.\,e \mid e_1\ e_2 \mid \Lambda\alpha.\,e \mid e\,[\tau] \mid \mathsf{pack}\,\sigma, e\,\mathsf{as}\,\exists\alpha.\,\tau \mid \mathsf{unpack}\,e_1\,\mathsf{as}\,\alpha, x\,\mathsf{in}\,e_2\end{aligned}$$

In class we covered the typing rules and the small-step, call-by-value dynamic semantics for the polymorphic λ-calculus with existential types.

Exercise 4. Prove the substitution lemma. (You only need to show the cases of the proof specified below.)

Substitution Lemma

(a) If $\Delta; \Gamma, x : \tau_1 \vdash e : \tau$ and $\Gamma \vdash e_1 : \tau_1$ then $\Gamma \vdash [e_1/x]\ e : \tau$.
The proof is by induction on the typing derivation $\Delta; \Gamma, x : \tau_1 \vdash e : \tau$.
Prove the cases for type abstraction, type application, pack, and unpack.

(b) If $\Delta, \alpha \vdash \tau$ and $\Delta \vdash \tau_1$ then $\Delta \vdash [\tau_1/\alpha]\ \tau$.
The proof is by induction on the typing derivation $\Delta, \alpha \vdash \tau$.
Prove the cases for type variables, universal types, and existential types.

[1] Note that we simply specify an encoding that works at any type τ, instead of formally being generic in that type.

(c) If $\Delta, \alpha; \Gamma \vdash e : \tau$ and $\Delta \vdash \tau_1$ then $\Delta; [\tau_1/\alpha]\ \Gamma \vdash [\tau_1/\alpha]\ e : [\tau_1/\alpha]\ \tau$.
The proof is by induction on the typing derivation $\Delta, \alpha; \Gamma \vdash e : \tau$.
Prove the cases for type abstraction, type application, pack, and unpack.

Exercise 5. Preservation State the preservation theorem and prove the cases for universal and existential types (both introduction and elimination forms). You can assume the usual lemmas, such as canonical forms, inversion, and of course, substitution (which you proved above). Clearly state the induction hypothesis and be explicit about when you appeal to it.

Exercise 6. Progress State the progress theorem and prove the cases for universal and existential types (both introduction and elimination forms). You can assume the usual lemmas as above. Again, clearly state the induction hypothesis and be explicit about when you appeal to it.