CIS 705 — Programming Languages — Spring 2010

Assignment 6

Due by 2:30 p.m. on Thursday, May 6

The context for this assignment is Chapter 11 of *TAPL*.
In particular, look at Exercise 11.4.1 and its solution.

1 The Simply Typed Lambda Calculus with Wildcard Abstractions and Booleans

If f is a function and x, y are elements of our universe, we define the function $f[x \mapsto y]$ from $\mathsf{dom}(f) \cup \{x\}$ to $\mathsf{ran}(f) \cup \{y\}$ by, for all $z \in \mathsf{dom}(f) \cup \{x\}$,

$$f[x \mapsto y](z) = \begin{cases} y, & \text{if } z = x, \\ f(z), & \text{if } z \neq x. \end{cases}$$

In the following, variables x are as usual, and we assume that the set of all variables is infinite and comes with a total ordering with the property that every nonempty set of variables has a least element.

Our *types* are inductively defined by:

$T ::=$	types:
Bool	type of booleans
$T \to T$	type of functions

As usual, $\to$ associates to the right. Our *terms* are inductively defined by:

$t ::=$	terms:
true	true constant
false	false constant
$\mathsf{if}\ t\ \mathsf{then}\ t\ \mathsf{else}\ t$	conditional
x	variable
$\lambda x : T.\, t$	abstraction
$\lambda - : T.\, t$	wildcard abstraction
$t\, t$	application

And our *values* (a subset of the terms) are:

$v ::=$	values:
true	true constant
false	false constant
$\lambda x : T.\, t$	abstraction
$\lambda \_ : T.\, t$	wildcard abstraction

As usual, application associates to the left and abstractions extend as far as possible.

In contrast to TAPL's approach, we do *not* identify abstractions up to the renaming of bound variables, so that, e.g., $\lambda x : T.x = \lambda y : T.y$ iff $x = y$. The *free variables* of a term t ($\mathsf{FV}(t)$) is defined recursively as follows:

$$\begin{aligned}
\mathsf{FV}(\mathsf{true}) &= \emptyset, \\
\mathsf{FV}(\mathsf{false}) &= \emptyset, \\
\mathsf{FV}(\mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3) &= \mathsf{FV}(t_1) \cup \mathsf{FV}(t_2) \cup \mathsf{FV}(t_3), \\
\mathsf{FV}(x) &= \{x\}, \\
\mathsf{FV}(\lambda x : T.\, t) &= \mathsf{FV}(t) \setminus \{x\}, \\
\mathsf{FV}(\lambda _ : T.\, t) &= \mathsf{FV}(t), \\
\mathsf{FV}(t_1\, t_2) &= \mathsf{FV}(t_1) \cup \mathsf{FV}(t_2).
\end{aligned}$$

A term is *closed* iff it has no free variables; otherwise it is *open*. The *substitution of* a *closed* term s *for the free occurrences of* a variable x *in* a term t ($[x \mapsto s]t$) is defined recursively by:

$$\begin{aligned}
[x \mapsto s]\mathsf{true} &= \mathsf{true}, \\
[x \mapsto s]\mathsf{false} &= \mathsf{false}, \\
[x \mapsto s](\mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3) &= \mathsf{if}\ [x \mapsto s]t_1\ \mathsf{then}\ [x \mapsto s]t_2\ \mathsf{else}\ [x \mapsto s]t_3, \\
[x \mapsto s]y &= \begin{cases} s, & \text{if } y = x, \\ y, & \text{if } y \neq x, \end{cases} \\
[x \mapsto s](\lambda y : T.\, t) &= \begin{cases} \lambda y : T.\, t, & \text{if } y = x, \\ \lambda y : T.\, [x \mapsto s]t, & \text{if } y \neq x, \end{cases} \\
[x \mapsto s](\lambda _ : T.\, t) &= \lambda _ : T.\, [x \mapsto s]t, \\
[x \mapsto s](t_1\, t_2) &= [x \mapsto s]t_1\, [x \mapsto s]t_2.
\end{aligned}$$

The *evaluation relation* $\boxed{t \to t'}$ between *closed* terms is inductively defined by:

$$\mathsf{if\ true\ then}\ t_2\ \mathsf{else}\ t_3 \to t_2 \qquad \text{(E-IfTrue)}$$

$$\mathsf{if\ false\ then}\ t_2\ \mathsf{else}\ t_3 \to t_3 \qquad \text{(E-IfFalse)}$$

$$\frac{t_1 \to t_1'}{\mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3 \to \mathsf{if}\ t_1'\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3} \qquad \text{(E-If)}$$

$$\frac{t_1 \to t_1'}{t_1\, t_2 \to t_1'\, t_2} \qquad \text{(E-App1)}$$

$$\frac{t_2 \to t_2'}{v_1\, t_2 \to v_1\, t_2'} \qquad \text{(E-App2)}$$

$$(\lambda x : T.\, t)v \to [x \mapsto v]t \qquad \text{(E-AppAbs)}$$

$$(\lambda\_ : T.\, t')v \to t' \qquad \text{(E-AppWildcardAbs)}$$

So, in the above, t_1, t_1', t_2, t_2', t_3 and t' range over *closed* terms, v and v_1 range over *closed* values, and $\mathsf{FV}(t) \subseteq \{x\}$.

A *typing context* (or just *context*) Γ is a function such that $\mathsf{dom}(\Gamma)$ is a finite subset of the variables, and $\mathsf{ran}(\Gamma)$ is a subset of the types. The *typing relation* $\boxed{\Gamma \vdash t : T}$ between typing contexts, terms and types is inductively defined by:

$$\Gamma \vdash \mathsf{true} : \mathsf{Bool} \qquad \text{(T-True)}$$

$$\Gamma \vdash \mathsf{false} : \mathsf{Bool} \qquad \text{(T-False)}$$

$$\frac{\Gamma \vdash t_1 : \mathsf{Bool} \qquad \Gamma \vdash t_2 : T \qquad \Gamma \vdash t_3 : T}{\Gamma \vdash \mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3 : T} \qquad \text{(T-If)}$$

$$\frac{(x, T) \in \Gamma}{\Gamma \vdash x : T} \qquad \text{(T-Var)}$$

$$\frac{\Gamma[x \mapsto T_1] \vdash t : T_2}{\Gamma \vdash \lambda x : T_1.\, t : T_1 \to T_2} \qquad \text{(T-Abs)}$$

$$\frac{\Gamma \vdash t : T_2}{\Gamma \vdash \lambda\_ : T_1.\, t : T_1 \to T_2} \qquad \text{(T-WildcardAbs)}$$

$$\frac{\Gamma \vdash t_1 : T_1 \to T_2 \qquad \Gamma \vdash t_2 : T_1}{\Gamma \vdash t_1\, t_2 : T_2} \qquad \text{(T-App)}$$

2 Adding Eager Ascription to Our Language

We refer to the simply typed lambda calculus of the preceding section as the *internal language*, λ_I. The *external language*, λ_E, consists of the result of making the following

additions to the internal language, adding eager ascription:

Terms To the definition of terms, we add:

$$\begin{array}{rr} t ::= \ldots & \text{terms:} \\ t \text{ as } T & \text{ascription} \end{array}$$

Values The definition of values is unchanged, except that in a value $\lambda x : T.\, t$, t is a term of the external language (may contain ascriptions).

Free Variables To the definition of $\mathsf{FV}(t)$ we add:

$$\mathsf{FV}(t \text{ as } T) = \mathsf{FV}(t).$$

Substitution To the definition of $[x \mapsto s]t$, we add:

$$[x \mapsto s](t \text{ as } T) = [x \mapsto s]t \text{ as } T.$$

Evaluation To the definition of the evaluation relation, we add:

$$t \text{ as } T \to t \quad \text{(E-AscribeEager)}$$

Typing To the definition of the typing relation, we add:

$$\frac{\Gamma \vdash t : T}{\Gamma \vdash t \text{ as } T : T} \quad \text{(T-Ascribe)}$$

We write $\to_I$ and $\to_E$ for the evaluation relations for λ_I and λ_E, respectively. And, we write $\vdash_I$ and $\vdash_E$ for the typing relations for λ_I and λ_E, respectively.

The **Determinacy of Evaluation Proposition** holds for both λ_I and λ_E:

- for all closed λ_I terms t, t' and t'', if $t \to_I t'$ and $t \to_I t''$, then $t' = t''$;
- for all closed λ_E terms t, t' and t'', if $t \to_E t'$ and $t \to_E t''$, then $t' = t''$.

The **Free Variables Lemma** holds for both λ_I and λ_E:

- for all contexts Γ, λ_I terms t and types T, if $\Gamma \vdash_I t : T$, then $\mathsf{FV}(t) \subseteq \mathsf{dom}(\Gamma)$;
- for all contexts Γ, λ_E terms t and types T, if $\Gamma \vdash_E t : T$, then $\mathsf{FV}(t) \subseteq \mathsf{dom}(\Gamma)$.

The **Uniqueness of Typing Proposition** holds for both λ_I and λ_E:

- for all contexts Γ, λ_I terms t, and types T and T', if $\Gamma \vdash_I t : T$ and $\Gamma \vdash_I t : T'$, then $T = T'$;

- for all contexts Γ, λ_E terms t, and types T and T', if $\Gamma \vdash_E t : T$ and $\Gamma \vdash_E t : T'$, then $T = T'$.

The **Progress Theorem** holds for both λ_I and λ_E:

- for all closed λ_I terms t, if $\emptyset \vdash_I t : T$ for some type T, then t is not stuck, i.e., is either a λ_I value, or there is a closed λ_I term t' such that $t \rightarrow_I t'$;
- for all closed λ_E terms t, if $\emptyset \vdash_E t : T$ for some type T, then t is not stuck, i.e., is either a λ_E value, or there is a closed λ_E term t' such that $t \rightarrow_E t'$.

And, the **Preservation Theorem** holds for both λ_I and λ_E:

- for all closed λ_I terms t and t', and types T, if $\emptyset \vdash_I t : T$ and $t \rightarrow_I t'$, then $\emptyset \vdash_I t' : T$;
- for all closed λ_E terms t and t', and types T, if $\emptyset \vdash_E t : T$ and $t \rightarrow_E t'$, then $\emptyset \vdash_E t' : T$.

We define an *elaboration* function e from the set of λ_E terms to the set of λ_I terms by structural recursion:

$$\begin{aligned}
e(\mathsf{true}) &= \mathsf{true}, \\
e(\mathsf{false}) &= \mathsf{false}, \\
e(\mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3) &= \mathsf{if}\ e(t_1)\ \mathsf{then}\ e(t_2)\ \mathsf{else}\ e(t_3), \\
e(x) &= x, \\
e(\lambda x : T.\, t) &= \lambda x : T.\, e(t), \\
e(\lambda _ : T.\, t) &= \lambda _ : T.\, e(t), \\
e(t_1\, t_2) &= e(t_1)\, e(t_2), \\
e(t\ \mathsf{as}\ T) &= (\lambda x : \mathsf{Bool} \rightarrow T.\, x\, \mathsf{true})(\lambda _ : \mathsf{Bool}.\, e(t)), \text{ where} \\
&\qquad x \text{ is the least variable.}
\end{aligned}$$

It is easy to see that, for all λ_E terms t:

- t is a λ_E value iff $e(t)$ is a λ_I value;
- $\mathsf{FV}(t) = \mathsf{FV}(e(t))$;
- t is closed iff $e(t)$ is closed.

Furthermore, the following **Elaboration of Substitution Proposition** holds: for all λ_E terms t, closed λ_E terms s, and variables x,

$$e([x \mapsto s]t) = [x \mapsto e(s)](e(t)).$$

3 Exercises

When solving the following exercises, you may use—without proof—the facts and propositions/theorems stated above, as well as the inversion lemmas and principles of induction for the inductively defined sets and relations. When doing a proof by induction on a set or relation, you *must* write down the predicate P you are using, and then do your proof using P.

Exercise 1 (30 Points)

Prove that, for all λ_E terms t,

$$\emptyset \vdash_E t : \mathsf{Bool} \quad \text{iff} \quad \emptyset \vdash_I e(t) : \mathsf{Bool}.$$

Exercise 2 (70 Points)

Prove that, for all closed λ_E terms t and $b \in \{\mathsf{true}, \mathsf{false}\}$,

$$t \to_E^* b \quad \text{iff} \quad e(t) \to_I^* b.$$