$\boxed{\text{CIS 500 Software Foundations}}$

Homework Assignment 6

Extensions of simply-typed λ-calculus, Derived forms

Due: Monday, November 7, 2005, by noon

Submission instructions:

You must submit your solutions electronically (in ascii, postscript, or PDF format). Electronic solutions should be submitted following the same instructions as last time; these can be found at `http://www.seas.upenn.edu/~cis500/homework.html`. Do **not** email your solutions to us.

1 Exercise A derived form is a function e from an external language λ^E to an internal language λ^I, such that the following conditions hold (note that this definition is slightly different than the one given in the book, in Theorem 11.3.1):

1. if $t \rightarrow_E t'$ then $e(t) \rightarrow_I e(t')$.
2. if $e(t) \rightarrow_I u$ then there exists a t' such that $u = e(t')$ and $t \rightarrow_E t'$.
3. if $\Gamma \vdash^E t : T$ then $\Gamma \vdash^I e(t) : T$.
4. if $\Gamma \vdash^I e(t) : T$ then $\Gamma \vdash^E t : T$.
5. if t is a value then $e(t)$ is a value.
6. if $e(t)$ is a value then t is a value.

Now, given the above properties about some derived form e and a theorem that states that the internal language is type sound,

$$\text{if } \Gamma \vdash^I e(t) : T,\ e(t) \rightarrow_I^* e(t'), \text{ and } e(t') \not\rightarrow_I, \text{ then } e(t') \text{ is a value},$$

prove the following theorem that states that the external language is type sound:

$$\text{If } \Gamma \vdash^E t : T,\ t \rightarrow_E^* t', \text{ and } t' \not\rightarrow_E, \text{ then } t' \text{ is a value.}$$

2 Exercise Prove that ascription (as defined in Figure 11-3) is a derived form for the simply-typed lambda calculus, using the definition of derived form from the previous exercise (i.e. show that the above six conditions hold.) You may prove each result by induction on the structure of external language terms, and you need only show the case for the new syntactic form t *as* T. Note that the function e can be defined as follows, by induction on the syntax of terms:

$$\begin{array}{ll}
e(t \textit{ as } T) & = (\lambda x : T.x)\ e(t) \\
e(x) & = x \\
e(\lambda x : T.t) & = \lambda x : T.e(t) \\
e(t_1\ t_2) & = e(t_1)\ e(t_2)
\end{array}$$

3 Exercise Consider the lambda calculus extended with lists as in Section 11.12 of TAPL. The type soundness theorem doesn't hold for this language.

1. Give an example of a stuck term.
2. How should the canonical forms lemma (9.3.4) be extended for lists?

3. Which of the lemmas necessary to prove type soundness (i.e. progress (9.3.5), substitution (9.3.8), and preservation (9.3.9)) break for this language?

4 Exercise For this exercise, you should first read TAPL Chapter 10. Then, complete an implementation of the language of TAPL Section 11.12, the simply-typed lambda calculus with lists.

To do this exercise, download `lists.tar.gz` from the homeworks page. This tarball can be unpacked using `tar zxvf lists.tar.gz`, and you should see a new directory called `lists` after doing this. You only need to change the definitions of the functions `eval` and `typeof` in the file `core.ml`; there are `TODO` comments in the appropriate places. The datatype that implements the syntax of the language is called `term`; it is defined in `syntax.ml` and you may find it useful to refer to the definition. Note: the typing annotations on cons, isnil, head and tail have been omitted from the syntax (cf. TAPL exercise 11.12.2).

To compile your interpreter, simply run `make` in the `lists` directory. The final executable will be called `f`. If you're in the `lists` directory, you can run it by executing `./f some_file`. We have provided a file `test.f` with some simple examples. You will need to add test cases to this file to debug your code.

For this exercise, please submit your version of the file `core.ml`. You do not need to submit any other files.

5 Exercise Implement the following ML program in the language of Section 11.12. You may check your work using the interpreter from the previous exercise.

```
let rec eot l = match l with
  (hd :: tl) -> if hd then next tl else false
  | [] -> true
and next l = match l with
  (hd :: tl) -> eot tl
  | [] -> true
in
  eot [true; false; true; true]
```

6 Debriefing

1. How many hours did each person in your group spend on this assignment, including time taken to read TAPL?

2. Would you rate it as easy, moderate, or difficult?

3. Did everyone in your study group participate?

4. How deeply do you feel you understand the material it covers (0%–100%)?

If you have any other comments, we would like to hear them; please send them to `cis500@cis.upenn.edu`.