CIS 500 Software Foundations

Homework Assignment 3

Untyped Lambda Calculus

Due: Monday, October 4, 2004, 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 solution to us.

1 Exercise This question is based on a compiler from the simple boolean language to the lambda calculus. The following recursive function (called *comp*, for compile) maps terms in the boolean language to lambda calculus terms:

$$
\begin{aligned}
&comp(\texttt{true}) && = \lambda \texttt{x}.\lambda \texttt{y}.(\texttt{x}(\lambda \texttt{z}.\texttt{z}))\\
&comp(\texttt{false}) && = \lambda \texttt{x}.\lambda \texttt{y}.(\texttt{y}(\lambda \texttt{z}.\texttt{z}))\\
&comp(\texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3) && = comp(\texttt{t1})(\lambda \texttt{x}.comp(\texttt{t2}))(\lambda \texttt{x}.comp(\texttt{t3}))
\end{aligned}
$$

1. What is *comp*(`if (if true then false else true) then false else false`) ?
2. What does the above lambda calculus term evaluate to? Show each step in the small-step evaluation.
3. Show the correctness of this compiler, using a simulation argument. In other words, prove that the execution of the compiled version of a term exactly simulates the original term. This idea is captured by the following theorem:

   If $t \to^* v$ (using the operational semantics of the boolean language) then $comp(t) \to^* comp(v)$ (using the operational semantics of the lambda calculus).

   The core of proving this theorem is the following lemma, which you should rigorously prove and turn in:

   If $t \to t'$ then $comp(t) \to^* comp(t')$
4. We cannot use the same simulation argument for the following encoding of booleans.

$$
\begin{aligned}
&comp(\texttt{true}) && = \lambda \texttt{x}.\lambda \texttt{y}.\texttt{x}\\
&comp(\texttt{false}) && = \lambda \texttt{x}.\lambda \texttt{y}.\texttt{y}\\
&comp(\texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3) && = comp(\texttt{t1})(comp(\texttt{t2}))(comp(\texttt{t3}))
\end{aligned}
$$

   Show a counter example to the above lemma using this encoding.

2 Exercise More proofs by induction

1. State the structural induction principle for terms in the lambda calculus.
2. State induction principle for the small-step evaluation relation for the lambda calculus.
3. Prove by induction over the structure of lambda calculus terms: *For all terms* `t`*, if* `t` *is closed, then either* `t` *is a value or there is some* `u` *such that* $\texttt{t} \to \texttt{u}$.
4. Prove that small-step evaluation is deterministic by induction over the evaluation relation : *If* $\texttt{t} \to \texttt{u}$ *and* $\texttt{t} \to \texttt{u}'$ *then* $\texttt{u} = \texttt{u}'$.

3 Exercise Scope

1. Which of these terms are closed?

(a) λx.λy.xy

(b) λx.y

(c) (λx.x)x

(d) λx.xx

2. Which of these pairs of terms are alpha-equivalent?

λx.λy.xy	λx.λx.xx
λw.λx.λy.w(xy)	λx.λy.λw.w(xy)
λw.λx.λy.w(xy)	λx.λy.λw.x(yw)
x	y
λw.(λx.x)x	λv.(λy.y)y

4 Exercise These problems involve programming in the untyped lambda calculus. You can test your solutions using the lambda calculus interpreter installed on `eniac-l.seas.upenn.edu`. The interpreter may be run with: `/usr/local/tapl/fulluntyped/f filename.f` where `filename.f` is a file containing lambda calculus expressions. The output is the value of evaluating those expressions (if any). The syntax can been seen by looking at the examples in the file `test.f` in the same directory.

1. TAPL 5.2.4
2. TAPL 5.2.7
3. TAPL 5.2.8
4. Use fix and the encoding of lists from 5.2.8 to write a function that determines if all boolean values in a list are `tru`.

5 Debriefing

1. How many hours did you spend on this assignment?
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 `cis500@cis.upenn.edu`.