Cornell University
Computer Science 6110

Homework 1 DUE: Monday, February 10, 11:59PM

What to turn in

Turn in the written part of the assignment by midnight on the due date. You can turn it in electronically to CMSX. Try to avoid having a huge file to submit; if you take pictures, increasing the contrast to the maximum tends to reduce file size dramatically.

1. Warmup [25 pts]

   (a) Write the following λ-calculus terms in their fully-parenthesized, curried forms. Change all bound variable names to names of the form $a_0, a_1, a_2, \ldots$ where the first λ binds a_0, the second a_1, and so on.

   i. $\lambda xy.\, z\ \lambda yz.\, z\ y\ x$
   ii. $\lambda x.\, (\lambda y.\, y\ x)\ \lambda x.\, y\ x$
   iii. $(\lambda x.\, y\ z\ \lambda y.\, x\ y)\ \lambda y.\, x\ y$

   (b) We defined capture-avoiding substitution into a lambda term using the following three rules:

$$
\begin{array}{lcll}
(\lambda x.\, e_0)\{e_1/x\} & = & \lambda x.\, e_0 & \\
(\lambda y.\, e_0)\{e_1/x\} & = & \lambda y.\, e_0\{e_1/x\} & (\text{where } y \neq x \wedge y \notin FV(e_1)) \\
(\lambda y.\, e_0)\{e_1/x\} & = & (\lambda y'.\, e_0\{y'/y\}\{e_1/x\}) & (\text{where } y' \neq x \wedge y' \notin FV(e_0) \wedge y' \notin FV(e_1))
\end{array}
$$

   In these rules, there are a number of conjuncts in the side conditions whose purpose is perhaps not immediately apparent. Show by counterexample that each of the above conjuncts of the form $x \notin FV(e)$ is independently necessary.

2. Equivalence and normal forms [15 pts]

   Two terms are observationally equivalent if there is no context in which they behave differently. Here a context $C[\,\cdot\,]$ is simply a term with a "hole" in it which a subterm could be plugged. For example, given the context $C = \lambda x.\, x\,[\,\cdot\,]$, the term $C[\lambda z.\, z\ x] = \lambda x.\, x\ (\lambda z.\, z\ x)$. Two lambda calculus terms are observationally equivalent if there is no context in which one diverges and the other does not.

   For each of the following pairs of λ-calculus terms, show either that the two terms are observationally equivalent or that they are not.

   (a) $(\mathit{SUCC}\ 0)$ and 1.

   (b) $\lambda x.\, x\ y$ and $\lambda x.\, y\ x$

3. Encoding lists [15 pts]

   We saw how to implement pairs in class with a pair constructor `PAIR`, defined as $\texttt{PAIR} = \lambda x.\, \lambda y.\, \lambda f.\, f\ x\ y$. Or equivalently, we could define `PAIR` by writing $\texttt{PAIR}\ x\ y = \lambda f.\, f\ x\ y$. In most languages, lists are implemented as pairs. We can do the same with our implementation.

   (a) Show how to implement values `NULL` and `ISNULL` with the property that $\texttt{ISNULL}\ \texttt{NULL} = \texttt{TRUE}$ and $\texttt{ISNULL}\ (\texttt{PAIR}\ x\ y) = \texttt{FALSE}$ for any x, y.

   (b) Show how to implement `List.map` in OCaml.

   (c) Show how to implement in lambda calculus a function `MAP` that behaves like `List.map` in OCaml.
   Here we need to select an appropriate fixed-point combinator `Y` for the reduction strategy we are using.

4. Encoding arithmetic [25 pts]

In class we saw that natural numbers could be implemented as Church numerals. But we didn't implement all the usual operations on natural numbers.

(a) Show how to write a λ-term `ISZERO` that determines whether a number is zero or not. It should return `TRUE` when the number is zero, and `FALSE` otherwise. Use the definitions of `TRUE` and `FALSE` given in class.

(b) Show how to write the `MUL` (multiply) operation for this number representation.

(c) Implement the `PRED` (predecessor) operation for this number representation. It should invert the successor operation `SUCC`, but need not do anything sensible when applied to 0. (Hint: start by constructing a function that maps $(\texttt{PAIR}\ n\ (n+1))$ to $(\texttt{PAIR}\ (n+1)\ (n+2))$.)

(d) Show how to write a λ-term `EVEN?` that determines whether a number is even or not. It should reduce to `TRUE` when the number is even, and `FALSE` otherwise.

In your answers to any of these problems, feel free to define new terms and give them abbreviated names.

5. S and K Combinators [20 pts]

Consider the following definitions:

$$\begin{aligned} S &\triangleq \lambda xyz.\,(x\ z)(y\ z) \\ K &\triangleq \lambda xy.\,x \end{aligned}$$

In this problem you will show that any λ-calculus expression can be expressed as a series of applications of the S and K combinators. In particular, if we think of S and K as part of the syntax, we can remove all of the λ's from the lambda calculus!

(a) Show that the S and K combinators can be used to construct an expression with the same normal form (under β and η reductions) as the identity expression $\lambda x.\,x$.

(b) Now consider the following target language, which we might call "the λ-less calculus":

$$\epsilon ::= \mathtt{S} \mid \mathtt{K} \mid x \mid \epsilon\ \epsilon$$

Write an *abstraction* function $\mathcal{A}$ such that $\mathcal{A}[\![x, \epsilon]\!]$ is extensionally equivalent to $\lambda x.\epsilon$ (with the interpretations of `S` and `K` given above). For example,

$$\mathcal{A}[\![x, x']\!] = (\mathtt{K}\ x') \text{ where } x \neq x'$$

because for all z,

$$(K\ x')\ z = x' = (\lambda x.\,x')\ z$$

Note that the definition of $\mathcal{A}$ may be recursive as long as the right-hand side of the definition only uses $\mathcal{A}$ on terms that are in some sense strictly smaller.

(c) Use $\mathcal{A}$ to construct a translation $\mathcal{C}$ from the complete λ-calculus to the λ-less calculus. Is your translation the most compact encoding possible?

Bonus factoid: We can define another combinator

$$X \triangleq \lambda x.\,x\ K\ S\ K$$

which can represent all closed λ-calculus expressions, because K has the same normal form as $(XX)X$ and S has the same normal form as $X(XX)$. So any lambda calculus term can be represented as a tree of applications of just this term!