What to turn in

Turn in the written part of the assignment by 5PM on the due date in Upson 4119. The programming part should be submitted using CMS (`http://cms.csuglab.cornell.edu`) by the same time. Most of the assignment is to be done individually, except for the last problem as noted below.

1. Free and bound variables (10 pts.)
   Identify the free and bound variables in each of the following expressions, and for bound variables indicate which lambda term binds them.

   (a) $\lambda xyz.\, zx(\lambda x.\, x)$
   (b) $\lambda xy.\, (\lambda z.\, zy)\ (\lambda x.\, zx)$
   (c) $(\lambda x.\, (y\ (\lambda y.\, xy)))(\lambda y.\, xy)$

   We defined capture-free substitution into a lambda term using the following three rules:

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

   (d) 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. Encodings (25 pts.)
   We have seen in class one way to represent natural numbers in the λ-calculus. However, there are many other ways to encode numbers. Consider the following definitions:

$$\mathsf{TRUE} \triangleq \lambda xy.\, x$$
$$\mathsf{FALSE} \triangleq \lambda xy.\, y$$
$$0 \triangleq \lambda x.\, x$$
$$n+1 = \lambda x.\, (x\ \mathsf{FALSE})\ n$$

   (a) Show how to write the PRED (predecessor) operation for this number representation. Reduce (PRED (PRED 2)) to its $\beta\eta$ normal form, which should be the representation of 0, above. PRED need not do anything sensible when applied to ZERO.
   (b) Show how to write a λ-term ZERO? 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 above.
   (c) Show how to write the PLUS and TIMES operations for this number representation.

   If you get problem 4 working, you can use it to test your solution!

3. The S and K Combinators (30 pts.)
   Consider the following definitions of the S and K combinators:

$$S \triangleq \lambda xyz.\, (xz)(yz)$$
$$K \triangleq \lambda xy.\, x$$

   Any λ-calculus expression without free variables can be written using only applications of the S and K combinators; thus, the λ-calculus can be universal with only three distinct identifier names, since both combinators use no more than three identifiers.

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

(b) Now, we will construct a translation from λ-calculus expressions to expressions containing only applications of the S and K combinators. This translation will be defined in terms of two functions: $\mathcal{C}[\![e]\!]$, which converts an expression e into this form, and a function $\mathcal{A}[\![x, e]\!]$, which *abstracts* the variable x from the expression e. removing all uses of x within e.

The idea is that $\mathcal{A}[\![x, e]\!] = \lambda x.\, e$, in the sense that the two expressions have the same effect when applied to any argument (they are extensionally equal). Using the function $\mathcal{A}$, the function $\mathcal{C}$ can be defined simply by structural induction:

$$\begin{aligned} \mathcal{C}[\![x]\!] &= x \\ \mathcal{C}[\![e_0\ e_1]\!] &= (\mathcal{C}[\![e_0]\!]\ \mathcal{C}[\![e_1]\!]) \\ \mathcal{C}[\![\lambda x\ e]\!] &= \mathcal{A}[\![x, \mathcal{C}[\![e]\!]]\!] \end{aligned}$$

Because $\mathcal{A}$ is only applied to expressions produced by $\mathcal{C}$, it needs to be defined only for expressions that are identifiers and applications. For example, consider $\mathcal{A}[\![x, x']\!]$ where $x' \neq x$. We require $(\mathcal{A}[\![x, x']\!] e) = (\lambda x\ x')e$ for any e, so we obtain the right effect with the following definition:

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

Define the remainder of the translation to the S and K combinators. Does this translation result in the most compact equivalent expression using these combinators? Justify your answer.

Bonus factoid: We can define another combinator

$$X \triangleq \lambda x.\, xKSK$$

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!

4. Implementing lambda calculus (35 pts.)

The file lambda.sml contains a partial implementation of some useful lambda calculus mechanisms. In particular, it contains a correct implementation of call-by-value implementation in the function cbv, and you can use it to try out evaluation. The function print_exp can be used to print a human-readable representation of an expression.

(a) This file also includes most of the implementation of a function nf that reduces a term to $\beta\eta$-normal form, but it doesn't quite work because the substitution function subst is not correct. Fix the implementation of subst and make nf work correctly.

(b) There is also a very incomplete implementation of a function translate that translates from an extended language to simple lambda calculus. The extended language includes let expressions (like in ML), recursive functions, and pairs. Complete translate so that it faithfully translates extended terms into lambda calculus terms.

Complete the implementation of lambda.sml and submit the result through CMS. You may do this part of the assignment (and only this part of the assignment) with a partner. Both partners are expected to understand the solution. Make sure to add a comment to lambda.sml indicating who your partner is, if any.