Homework 6 — ML module language — assigned Monday 17 April — due Tuesday 25 April

All code in this homework assignment must use the SML module language, and it must be organized using the SML/NJ Compilation Manager. Place all functor applications together in a file link.sml. Put all tests into a structure Tests in the file tests.sml.

6.1 Lambda-calculus (30pts)

A library of λ-terms

\[ \begin{align*}
I & \triangleq \lambda x.x \\
K & \triangleq \lambda xy.x \\
S & \triangleq \lambda fgx.(fx)(gx) \\
B & \triangleq \lambda gxy.fx(gx) \\
C & \triangleq \lambda gxy.fgx \\
\omega & \triangleq \lambda xx \\
\Omega & \triangleq \omega \omega \\
\text{true} & \triangleq \lambda xy.x \\
\text{false} & \triangleq \lambda xy.y \\
\text{not} & \triangleq \lambda ttf \text{true} \\
\text{cond} & \triangleq \lambda ee_{1}ee_{2}ee_{1}e_{2} \\
pair & \triangleq \lambda e_{1}e_{2}.f.e_{1}e_{2} \\
fst & \triangleq \lambda p.p \text{true} \\
snd & \triangleq \lambda p.p \text{false} \\
0 & \triangleq \lambda fx.x \\
1 & \triangleq \lambda fx.fx \\
2 & \triangleq \lambda fx.f(fx) \\
succ & \triangleq \lambda nfx.nf(fx) \\
add & \triangleq \lambda mnfx.mf(nfx) \\
iszero & \triangleq \lambda n.n(\lambda x.\text{false})true \\
\text{prefn} & \triangleq \lambda fp.pair \text{false}(\text{cond}(\text{fst}p)(\text{snd}p)(f(\text{snd}p))) \\
pred & \triangleq \lambda nfx.snd(\text{pair}(\text{false})(\text{true}x)) \\
\text{cons} & \triangleq \lambda hts.sht \\
\text{hd} & \triangleq \lambda LL \text{true} \\
\text{tl} & \triangleq \lambda LL \text{false} \\
\text{nil} & \triangleq \lambda x.\text{true} \\
\text{isempty} & \triangleq \lambda LL(\lambda ht.\text{false}) \\
\end{align*} \]

Normal forms of some λ-terms

\[ \begin{align*}
\text{SKK} & \rightarrow \lambda x.x \\
\text{K(SII)} & \rightarrow \lambda ab.bb \\
\text{SSSSSS} & \rightarrow \lambda ab.(ab(ab(ab\lambda ac(bc)))) \\
\end{align*} \]

6.1.1

Show that the following λ-terms have a normal form:

1. \((\lambda y.yyy)((\lambda ab.a)I(\text{SS}))\)
2. \((\lambda yz.zy)((\lambda x.xxx)(\lambda x.xxx))(\lambda w.I)\)

6.1.2

For each of the following λ-terms either find its normal form or show that it has no normal form:

1. \((\lambda x.xx)(\lambda x.x)\)
2. \((\lambda x.xx)(\lambda x.xx)\)
3. \text{Y}
4. \text{Y}(\lambda y.y)\]
6.1.3

(Turing) Let $A \triangleq \lambda xy.y(xxy)$. Let $\Theta \triangleq AA$. Show that $\Theta$ is a fixed-point operator.

6.2 Lambda-calculus Interpreter (70pts) [K.1.1; K.3.1; K.3.2]

Develop an interpreter for the $\lambda$-calculus that will automate reductions. This program will follow literally the rules for $\beta$-conversion and the rules for substitution. The internal representation of $\lambda$-terms is essentially the same as the textual representation, though the data type makes the bracketing structure apparent, and pattern-matching easier.

We must first specify the internal representation for $\lambda$-terms. The following type must be used:

```ml
type var = string
datatype expr = Var of var
  | Abs of var * expr
  | App of expr * expr
```

The following tasks build the interpreter bottom-up.

6.2.1

Implement an environment mapping identifiers to $\lambda$-terms, with type $\text{string -> expr}$. There should be a mechanism to build new environments out of old ones by introducing a new definition for an identifier.

6.2.2

Implement a function $\text{freeVariables: expr -> var list.}$

6.2.3

Implement a function $\text{isFreeVariable: var * expr -> bool.}$

6.2.4

Implement a function $\text{substitute: expr * var * expr -> expr,}$ such that $\text{substitute (e, x, t)}$ substitutes $t$ for $x$ in $e$. To generate fresh variable symbols, use the following code:

```ml
local
  val counter = ref 0
in
  fun genSym () =
    let
      val x = !counter
```
val _ = counter := x+1
in
   "_" ^ Int.toString x
end
end

6.2.5

Implement a function isBetaRedex: expr -> bool.

6.2.6

Implement a function convertBetaRedex: expr -> expr.

6.2.7

Implement a function convert: expr -> expr option which finds a leftmost outermost \( \beta \) redex, if any, and performs \( \beta \) conversion.

6.2.8

Implement a function reduce: expr -> expr that applies \( \beta \) conversion steps in normal order until a normal form is found.

6.2.9

Test your program by reducing various \( \lambda \)-terms, such as: \( \texttt{SKK; K(SII); S(S(KS)(KI))(KI); SSSSSSS} \).

6.2.10

Implement the factorial function over Church numerals. (Use the \( \texttt{Y} \) combinator.) Test your program by having it compute \( n! \) for various \( n \). Report how fast the evaluator works for different inputs or input sizes. (Take into account that with a unary representation, different numbers have different sizes.)

**How to turn in**

Submission instructions have been posted to the course mailing list.

Include the following statement with your submission, signed and dated:

*I pledge my honor that in the preparation of this assignment I have complied with the University of New Mexico Board of Regents' Policy Manual.*