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

$\mathbf{I} \triangleq \lambda x.x$ $\mathbf{K} \triangleq \lambda xy.x$ $\mathbf{S} \triangleq \lambda fgx.(fx)(gx)$ $\mathbf{B} \triangleq \lambda fgx.f(gx)$ $\mathbf{C} \triangleq \lambda fgx.fxg$

$\omega \triangleq \lambda x.xx$ $\Omega \triangleq \omega\omega$ $\mathbf{Y} \triangleq \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$

$\mathbf{true} \triangleq \lambda xy.x$ $\mathbf{false} \triangleq \lambda xy.y$ $\mathbf{not} \triangleq \lambda t.t\ \mathbf{false}\ \mathbf{true}$ $\mathbf{cond} \triangleq \lambda ee_1e_2.ee_1e_2$

$\mathbf{pair} \triangleq \lambda e_1e_2f.fe_1e_2$ $\mathbf{fst} \triangleq \lambda p.p\ \mathbf{true}$ $\mathbf{snd} \triangleq \lambda p.p\ \mathbf{false}$

$\mathbf{0} \triangleq \lambda fx.x$ $\mathbf{1} \triangleq \lambda fx.fx$ $\mathbf{2} \triangleq \lambda fx.f(fx)$ $\mathbf{succ} \triangleq \lambda nfx.nf(fx)$ $\mathbf{add} \triangleq \lambda mnfx.mf(nfx)$
$\mathbf{iszero} \triangleq \lambda n.n(\lambda x.\mathbf{false})\mathbf{true}$ $\mathbf{prefn} \triangleq \lambda fp.\mathbf{pair}\ \mathbf{false}(\mathbf{cond}(\mathbf{fst}\ p)(\mathbf{snd}\ p)(f(\mathbf{snd} p)))$
$\mathbf{pred} \triangleq \lambda nfx.\mathbf{snd}(n(\mathbf{prefn}\ f)(\mathbf{pair}\ \mathbf{true}\ x))$

$\mathbf{cons} \triangleq \lambda hts.sht$ $\mathbf{hd} \triangleq \lambda L.L\ \mathbf{true}$ $\mathbf{tl} \triangleq \lambda L.L\ \mathbf{false}$ $\mathbf{nil} \triangleq \lambda x.\mathbf{true}$ $\mathbf{isempty} \triangleq \lambda L.L(\lambda ht.\mathbf{false})$

Normal forms of some λ-terms

$\mathbf{SKK} \twoheadrightarrow \lambda x.x$ $\mathbf{K}(\mathbf{SII}) \twoheadrightarrow \lambda ab.bb$ $\mathbf{S}(\mathbf{S}(\mathbf{KS})(\mathbf{KI}))(\mathbf{KI}) \twoheadrightarrow \lambda ab.bb$
$\mathbf{SSSSSSS} \twoheadrightarrow \lambda ab.(ab(ab(ab\lambda c.ac(bc))))$

6.1.1

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

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

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. $\mathbf{Y}$
4. $\mathbf{Y}(\lambda y.y)$

6.1.3

(Turing) Let $A \triangleq \lambda xy.y(xxy)$. Let $\Theta \triangleq AA$. Show that Θ 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 λ-calculus that will automate reductions. This program will follow literally the rules for β-conversion and the rules for substitution. The internal representation of λ-terms is essentially the same as the textual representation, though the data type makes the bracketting structure apparent, and pattern-matching easier.

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

```
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 λ-terms, with type `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 `freeVariables: expr -> var list`.

6.2.3

Implement a function `isFreeVariable: var * expr -> bool`.

6.2.4

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

```
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 β redex, if any, and performs β conversion.

6.2.8

Implement a function `reduce: expr -> expr` that applies β conversion steps in normal order until a normal form is found.

6.2.9

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

6.2.10

Implement the factorial function over Church numerals. (Use the $\mathbf{Y}$ 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.