Exercise Set 10 — 15 November

10.1 Lambda-calculus

This exercise concerns the untyped lambda-calculus from Chapter 5 of our textbook, *Types and Programming Languages* by Benjamin Pierce. We assume full beta reduction.

A library of λ-terms

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

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

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

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

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

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

Normal forms of some λ-terms

Verify that you are able to obtain these normal forms.

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

10.1.1

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

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

10.1.2

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

1. $(\lambda x.x\,x)(\lambda x.x)$
2. $(\lambda x.x\,x)(\lambda x.x\,x)$
3. $\mathbf{Y}$
4. $\mathbf{Y}(\lambda y.y)$

10.1.3

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

10.2 Lambda-calculus Interpreter

Develop an interpreter for the lambda-calculus that will automate reductions. This program will follow literally the rules for β-conversion and the rules for substitution.

The internal representation for λ-terms will be:

```
type Var = String
data Term = Var Var
          | Abs Var Term
          | App Term Term
```

The following tasks build the interpreter bottom-up.

10.2.1

Implement an environment mapping variables to terms, with type `Var -> Term`. There should be a mechanism to build new environments out of old ones by introducing a new definition for a variable.

10.2.2

Implement a function `freeVariables :: Term -> [Var]`.

10.2.3

Implement a function `isFreeVariable :: Var -> Term -> Bool`.

10.2.4

Implement a function `substitute :: Term -> Var -> Term -> Term`, such that `substitute e x t` substitutes t for free occurrences of x in e. In some cases you will need to rename variables; see Chapter 5 of TAPL.

10.2.5

Implement a function `isBetaRedex :: Term -> Bool`.

10.2.6

Implement a function `convertBetaRedex :: Term -> Term`.

10.2.7

Implement a function `convert :: Term -> Maybe Term` which finds a leftmost outermost β-redex, if any, and performs β conversion.

10.2.8

Implement a function `reduce :: Term -> Term` that applies β-conversion steps in normal order until a normal form is reached (if ever).

10.2.9

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

10.2.10

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.)