Computer Science 131, Spring 2002
Assignment 10: λ-Calculus Implementation (Part 2)

Out: Wednesday, April 3

Due: Wednesday, April 10, at the beginning of class

Your code should be placed in a file named `assign10.sml` and handed in using `cs131submit`.

1 One-step β-Reduction (80%)

Because (as you saw in class) there may be many ways to β-reduce a single term, people have come up with various "strategies" which, given any term, specify what the next reduction step should be. Here are four (the names are historical).

- **Normal-Order Reduction**: reduce the application (of a function to an argument) whose λ appears leftmost in (i.e., closest to the beginning of) the term.

  $$\Big(\lambda x.(\lambda z.((\lambda w.w)(\lambda v.v)))\Big)\Big((\lambda u.u)(\lambda y.y)\Big) \rightarrow_\beta \lambda z.((\lambda w.w)(\lambda v.v)) \rightarrow_\beta \lambda z.(\lambda v.v)$$

  Normal-order reduction is interesting because it has the following guarantee: if there is any possible sequence of reductions to a term which has no occurrences of a function applied to an argument, then normal-order reduction will eventually produce this term.

- **Applicative-Order Reduction**: reduce the application whose λ occurs leftmost if the argument of this application cannot be further reduced; otherwise, first reduce the argument by one step (using applicative-order reduction).

  $$\begin{aligned}&\Big(\lambda x.(\lambda z.((\lambda w.w)(\lambda v.v)))\Big)\Big((\lambda u.u)(\lambda y.y)\Big)\\ \rightarrow_\beta\ &\Big(\lambda x.(\lambda z.((\lambda w.w)(\lambda v.v)))\Big)\Big(\lambda y.y\Big) \rightarrow_\beta \lambda z.((\lambda w.w)(\lambda v.v)) \rightarrow_\beta \lambda z.(\lambda v.v)\end{aligned}$$

  Applicative reduction is does not have the same termination guarantee as normal-order reduction, but in many cases it is more efficient.

- **Call-by-Name Evaluation**: The same as normal-order reduction, except we never reduce applications inside the body of a function. (This strategy may stop before all reductions are gone.)

  $$\Big(\lambda x.(\lambda z.((\lambda w.w)(\lambda v.v)))\Big)\Big((\lambda u.u)(\lambda y.y)\Big) \rightarrow_\beta \lambda z.((\lambda w.w)(\lambda v.v))$$

This is a less aggressive version of normal-order reduction. If the result is a function that is not being applied to anything, why bother working on the function?

- **Call-by-Value Evaluation**: The same as applicative-order reduction, except that we never reduce applications inside the body of a function. (This strategy may stop before all reductions are gone.)

$$\Big(\lambda x.(\lambda z.((\lambda w.w)(\lambda v.v)))\Big)\Big((\lambda u.u)(\lambda y.y)\Big)$$
$$\rightarrow_\beta \Big(\lambda x.(\lambda z.((\lambda w.w)(\lambda v.v)))\Big)\Big(\lambda y.y\Big) \rightarrow_\beta \lambda z.((\lambda w.w)(\lambda v.v))$$

  This is the corresponding less-aggressive version of applicative reduction.

Write four functions corresponding to the above four strategies:

```
normal      : lam -> lam option
applicative : lam -> lam option
cbn         : lam -> lam option
cbv         : lam -> lam option
```

These functions should return `NONE` if the argument cannot be reduced according to the given strategy, and `SOME` e' if the argument can be reduced one step to get the new term e'.

You may find the following pattern to be useful for many cases within your functions:

```
(case ... recursive call ... of
   NONE => ... do something...
 | SOME e' => ... do something else...
)
```

It is always wise to put parentheses around `case` statements.

2 Multistep β-Reduction (20%)

Write a function

```
reduce : (lam -> lam option)  ->  lam  ->  lam
```

which takes a reduction strategy (i.e., one of the four functions above) and a lambda expression as sequential arguments. The `reduce` function should then repeatedly do one-step reduction using the given strategy until no further reductions are possible; if this occurs, return the final λ-expression. Note that there are inputs such as

$$(\lambda x.(x\ x))(\lambda y.(y\ y))$$

for which `reduce` will not terminate for any reduction strategy, and there are other inputs such as

$$(\lambda z.w)((\lambda x.(x\ x))(\lambda y.(y\ y)))$$

or

$$\lambda z.((\lambda x.(x\ x))(\lambda y.(y\ y)))$$

which terminate under some strategies but not others.