Assignment 2
Introduction to Computational Logic, SS 2005

Prof. Dr. Gert Smolka, Dr. Lutz Straßburger
http://www.ps.uni-sb.de/courses/cl-ss05/

see http://www.ps.uni-sb.de/courses/cl-ss05/script/term.sml

This week we are concerned with the implementation of S in Standard ML. Most of it has been explained in the lecture and you will find the source code in the Net. The assignment will help you to get a solid understanding of the data structure for terms and the algorithms for substitution, reduction and elimination.

Exercise 2.1 Declare representations of the following terms:

a) $\lambda x.fx$

b) $\lambda x.g(fxa)$

c) $\lambda yx.(\lambda zy.fyx)ay$

Use the nice syntax provided by *C*, *var*, *app* and *lam*.

Exercise 2.2 (Reduction) There is a theorem that says that the simplification of a term with the β- and the η-rule always terminates with a unique result. You are going to write a procedure that computes this result, which is called the $\beta\eta$-normal form of the initial term. First we need two auxiliary procedures.

a) Declare a procedure *bsubst* : *int* → *ter* → *ter* → *ter* such that *bsubst 0 t t′* yields the term obtained by applying the β-rule topmost to the term $(\lambda T.t)t'$. The first argument of *bsubst* provides the number of λ's on the path to the root.

b) Declare a procedure *down* : *int* → *ter* → *ter* such that *down 0 t* yields the term that is obtained by decrementing the free variable occurrences in *t* by 1. If the variable 0 occurs free in *t*, *down* should raise the exception *Domain*. The first argument of *down* provides the number of λ's on the path to the root.

c) Complete the following procedure *red* : *ter* → *ter* such that it yields the $\beta\eta$-normal form of a term:

```
fun red (A(s,t)) = beta (red s) (red t)
  | red (L(ty,t)) = eta ty (red t)
  | red t = t
and beta s t = case s of
    L(_,s') =>
  | _ =>
and eta ty t = case t of
    A(t', V 0) =>
  | _ =>
```

d) Test your procedure *red* with the terms from Exercise 2.1.

e) There are ill-typed terms whose simplification with the β- and the η-rule does not terminate. Find such a term. Hint: Start with the term $\lambda x.xx$.

Exercise 2.3 (Elimination) If the constants S, K, I are available for all types, then there exists for every term a logically equivalent term that does not contain λ's (a so-called combinatory term). Find equivalent combinatory terms for the following terms. Ignore types.

a) $\lambda xy.y$

b) $\lambda xyz.y$

c) $\lambda xyz.x$

d) $\lambda x.fx(fxx)$

Now assume the typing $x, y, z : T$ and $f : T \to T \to T$. Determine the type subscripts of the occurrences of the polymorphic constants S, K, I in your combinatory terms.

Exercise 2.4 (Elimination) It is not difficult to write a procedure $elim : ter \to ter$ that computes equivalent combinatory terms if types are ignored:

```
free0 tests whether the variable 0 occurs free in t

fun free0 t = (down 0 t; false) handle Domain => true


elim' yields for a combinatory term t a combinatory term t'
such that λT.t and t' are logically equivalent

fun elim' (V 0) = ...
  | elim' (A(s, V 0)) = if free0 s then ... else ...
  | elim' (A(s,t)) = ...
  | elim' t = ...

fun elim (A(s,t)) = A(elim s, elim t)
  | elim (L(_,t)) = down 0 (elim' (elim t))
  | elim t = t
```

Complete *elim′* such that *elim* yields $S(Kg)(Sf(Ka))$ for $\lambda x.g(fxa)$. The procedure *down* is from Exercise 2.2.