Assignment 2
Introduction to Computational Logic, SS 2006

Prof. Dr. Gert Smolka, Dipl.-Inform. Mathias Möhl
http://www.ps.uni-sb.de/courses/cl-ss06/

Exercise 2.1 (Schönfinkel's U) In his paper from 1924, Moses Schönfinkel showed that the Boolean operations and the quantifiers can be expressed with the function

$$U \in (X \to \mathbb{B}) \to (X \to \mathbb{B}) \to \mathbb{B}$$
$$Ufg = (\forall x \in X\colon \neg(fx \wedge gx))$$

If f and g are interpreted as subsets of X, then U tests whether f and g are disjoint. Describe the following functions with U:

a) Negation $\neg \in \mathbb{B} \to \mathbb{B}$
b) Disjunction $\vee \in \mathbb{B} \to \mathbb{B} \to \mathbb{B}$
c) Existential quantification $\exists \in (X \to \mathbb{B}) \to \mathbb{B}$

Exercise 2.2 (Henkin's Reduction) In a paper from 1963, Leon Henkin showed that the Boolean operations and the quantifiers can be expressed with the identities

$$\doteq_X \in X \to X \to \mathbb{B}$$
$$x \doteq_X y = (x = y)$$

a) Express 1 with $\doteq_{\mathbb{B}\to\mathbb{B}}$.
b) Express 0 with 1 and $\doteq_{\mathbb{B}\to\mathbb{B}}$.
c) Express $\neg$ with 0 and $\doteq_{\mathbb{B}}$.
d) Express $\wedge$ with 1 and $\doteq_{(\mathbb{B}\to\mathbb{B}\to\mathbb{B})\to\mathbb{B}}$.
e) Express $\forall_X$ with 1 and $\doteq_{X\to\mathbb{B}}$.

Exercise 2.3 (Choice Function) Let $\mathbf{C} \in (\mathbb{N} \to \mathbb{B}) \to \mathbb{N}$ be a function that satifies

$$\forall f \in \mathbb{N} \to \mathbb{B}\colon \quad f(\mathbf{C}f) = \exists f$$

Furthermore, let the Boolean operations $\neg$, $\wedge$, $\vee$, $\Rightarrow$ and the functions

$$+, \cdot \in \mathbb{N} \to \mathbb{N} \to \mathbb{N}$$
$$\leq, \doteq \in \mathbb{N} \to \mathbb{N} \to \mathbb{B}$$

be given. Describe the following values with the given constants.

a) $0 \in \mathbb{N}$
b) $1 \in \mathbb{N}$
c) Subtraction $- \in \mathbb{N} \to \mathbb{N} \to \mathbb{N}$
d) Integer division $div \in \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ (Example: $div\, 7\, 3 = 2$)
e) $max \in \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ such that $max\, x\, y$ yields the maximum of x, y
f) $if \in \mathbb{B} \to \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ such that $if\, b\, x\, y$ yields x if $b = 1$ and y otherwise
g) Existential quantifier $\exists \in (\mathbb{N} \to \mathbb{B}) \to \mathbb{B}$

Exercise 2.4 (Normal Forms) Derive the $\beta\eta$-normal forms of the following terms. Ignore types.

a) $\lambda xy.fx$
b) $\lambda xy.fy$
c) $\lambda xy.fxy$
d) $(\lambda xy.fyx)zxy$
e) $(\lambda xy.fxy)ab$
f) $(\lambda z.a)(\lambda xy.fxy)$
g) $(\lambda xu.x)(\lambda vy.y)$
h) $(\lambda f.fa)(\lambda xy.fxy)b$
i) $(\lambda z.(\lambda xy.fxy)((\lambda z.z)z))ab$
j) $(\lambda fgx.fx(gx))(\lambda xy.x)(\lambda xz.x)$

Exercise 2.5 (Lambda Elimination) Suppose the constants S, K, I are available for all types and satisfy the following semantic equivalences:

$$\lambda x.x \equiv I \qquad (I)$$
$$\lambda y.x \equiv Kx \qquad (K)$$
$$\lambda x.st \equiv S(\lambda x.s)(\lambda x.t) \qquad (S)$$

Then there exists for every term a semantically equivalent combinatory term. (A combinatory term is a term that doesn't contain abstractions.) Derive equivalent

combinatory terms for the following terms using the laws β, η, I, K, and S. Ignore types.

a) $\lambda xy.y$

b) $\lambda xyz.y$

c) $\lambda xyz.x$

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

e) $(\lambda fx.fy)(\lambda x.x)$

Exercise 2.6 (Implementation of Terms) We implement terms in Standard ML. To keep things simple, we omit types and constants. The implementation is based on the declarations

```
type index = int  (* non-negative *)
datatype ter = I of index | A of ter * ter | L of ter
```

Variables and de Buijn indices are both represented with the constructor I and non-negative integers:

$$\begin{aligned} \lambda x.x &\rightsquigarrow L(I\ 0) \\ \lambda fx.fx &\rightsquigarrow L(L(A(I\ 1,\ I\ 0))) \\ \lambda x.fx &\rightsquigarrow L(A(I(f+1),\ I\ 0)) \end{aligned}$$

As an example, we consider a procedure $free : index \to ter \to bool$ that tests whether a variable occurs in a term. We realize *free* with a procedure $free'$ that takes the number of superordinate abstractions as additional argument:

```
fun free' d x (I y) = y>=d andalso x=y-d
  | free' d x (A(s,t)) = free' d x s orelse free' d x t
  | free' d x (L t) = free' (d+1) x t

fun free x t = free' 0 x t
```

a) Write a procedure $subst : (index \to ter) \to ter \to ter$ such that *subst f t* yields the term that is obtained from t by replacing every free occurrence of every variable x with fx. For instance:

$$subst\ (fn\ i\ \Rightarrow\ if\ i = x\ then\ I\ y\ else\ I\ i)\ (L(I(x+1))) \quad = \quad (L(I(y+1)))$$

b) Write a procedure $lam : index \to ter \to ter$ that implements the abstraction operator $\lambda x.t$. For instance: $lam\ x\ (A(I\ y,\ I\ x)) = L(A(I(y+1),\ I\ 0))$.