Semantics, WS 2003 – Assignment 14

Prof. Dr. Gert Smolka, Dipl.-Inform. Guido Tack
http://www.ps.uni-sb.de/courses/sem-ws03/

Recommended reading: TAPL, chapters 29, 30, Paper on **Henk** [Peyton Jones, Meijer, 1997]

Exercise 14.1: Evaluator and type checker for F_ω based on LC Implement F_ω based on the Lambda Cube. Represent expressions as follows:

```
datatype exp = V of int | A of exp*exp | L of exp*exp | Q of exp*exp | S | B
```

For instance, $\lambda X:*.\lambda x:X.x$ is represented as `L(S, L(V 0, V 0))`.

(a) Write procedures `shift: int -> int -> exp -> exp` and `subst: int -> exp -> int -> exp -> exp` that shift and substitute expressions. The first argument of `shift` and `subst` is called *cutoff* and counts the binders passed. Given a cutoff c a variable occurrence x is local iff $x < c$. An application `shift c d e` adds d to all free variable occurences in e. An application `subst c e y e'` replaces all free occurrences of y in e by e'.

(b) Write a procedure `eval: exp -> exp` that computes the β-normal form of an expression by reducing all β-redeces. Do not use the procedures `shift` and `subst`. Instead, use the procedure

```
fun beta e e' = shift 0 ~1 (subst 0 e 0 (shift 0 1 e'))
```

to apply the body e of a procedure to an expression e'.

(c) Represent environments as lists of expressions and use the procedure

```
exception Error of string
fun get g x = shift 0 (x+1) (List.nth(g,x) handle Subscript
                              => raise Error"unbound variable")
```

to obtain the sort of a variable x from an evironment g. Write a procedure `check: exp list -> exp -> exp` that checks whether an expression is well-sorted in an environment and returns its sort if this is the case. You can assume that the initial environment for check is always `nil`. Write `check` such that the expressions it returns and puts in the environment upon recursion are β-normal. Make sure that only well-sorted expressions are evaluated to avoid divergence.

Use the procedures `get`, `beta`, `eval` and

```
fun squiggle S S = ()   (* S *)
  | squiggle B S = ()   (* F *)
  | squiggle B B = ()   (* Fw *)
  | squiggle _ _ = raise Error"illegal quantification"
```

Do not use `shift` and `subst`.

(d) Write a procedure `checkE: exp list -> exp list` that checks whether an environment is well-sorted and returns the normalized environment if this is the case.

Hint: We provide examples to test your procedures at the Web site of the course.

Exercise 14.2 Binary sums $X + Y$ can be provided by a generic ADT that takes X and Y as arguments (Exercise 13.4) or by an ADT that provides a type operator $* \to * \to *$ and polymorphic introduction and elimination procedures.

(a) Realize both possibilities in SML.

(b) Write the ADT with the type operator in F_ω. Give its type. Test your ADT with the LC implementation from the previous exercise.