Funktionale Programmierung SS 2013 - Lösung 4

Prof. Dr. U. Kastens
Institut für Informatik, Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn
27.05.2013

Lösung zu Aufgabe 1

```
fun outside("("::"*"::t) = inside(t)
|   outside(h::t) = h::outside(t)
|   outside nil = nil and
    inside ("*"::")"::t) = outside(t)
|   inside (h::t) = inside(t)
|   inside nil = nil;

outside ["c","l","a","s","s"," ","(","*","k","e","y","*",")"," ",
         "n","a","m","e"," ","(","*","i","d","*",")"," ","b","o","d","y",";"];
```

a) Das Funktionspaar filtert SML-Kommentare der Form `(* ... *)` aus einem Text, der als eine Liste von Buchstaben gegeben ist.

b) Eliminieren der wechselseitigen Rekursion:

```
fun inside("*"::")"::t) = t
|   inside (h::t) = inside(t)
|   inside nil = nil;

fun outside("("::"*"::t) = outside(inside(t))
|   outside(h::t) = h::outside(t)
|   outside nil = nil;
```

Lösung zu Aufgabe 2

a) Typ für allgemeine Bäume (beliebig viele Kinder):

```
datatype 'a tree = Lf | Br of 'a * 'a tree list;
```

b) Verallgemeinerte Version der Funktion `size`:

```
fun size(Lf) = 0
|   size(Br(_, [])) = 1
|   size(Br(x, h::t)) = size(h) + size(Br(x, t));
```

Aufruf mit dem geforderten Baum:

```
val t = Br(1, [Br(2,[]),
               Br(3,
                  [Br(4,[]),
                   Br(5, []),
                   Br(6,
                      [Br(7,[])
                      ]
                     )
                   ]
                  )
              ]
             );

size(t);  (* = 7 *)
```

Lösung zu Aufgabe 3

a) Die abstrakte Datentyp-Definition "Keller":

```
abstype
   'a stack = SEmpty | SCons of 'a * 'a stack
   with
   exception ExEmpty;
   val empty = SEmpty;

   fun push (v,s) = SCons(v, s);
   fun pop(SCons(v,s)) = s
   |   pop(SEmpty) = raise ExEmpty;
   fun top(SCons(v,s)) = v
   |   top(SEmpty) = raise ExEmpty;
end;
```

b) Ein Auswerter für Postfix-Ausdrücke:

```
datatype operation = plus | times;
datatype elem = oper of operation | value of int;

(* Beispiel: Der Postfix-Ausdruck "2 16 * 10 +" *)
val e42 = [value(2), value(16),  oper(times), value(10), oper(plus)];

fun stackeval(value(v)::t, s) = stackeval(t, push(v,s))
|  stackeval(oper(times)::t, s) = stackeval(t, let val (v1, v2) = (top(s),top(pop(s))) in
                                                push(v1*v2, pop(pop(s))) end)
|  stackeval(oper(plus)::t, s) = stackeval(t, let val (v1, v2) = (top(s),top(pop(s))) in
                                               push(v1+v2, pop(pop(s))) end)
|  stackeval(nil, s) = top(s);

fun evaluate(expr) = stackeval(expr, empty);
evaluate(e42);
```

Lösung zu Aufgabe 4

a) Die Funktion `eval`, die den Dezimalwert von Binärzahlen berechnet:

```
fun evala (nil,a)  = a
|   evala (0::t,a) = evala(t, 2*a)
|   evala (1::t,a) = evala(t, 2*a + 1);

fun eval l = evala (l, 0);
```

b) Mit Fehlerbehandlung, falls andere Ziffern als 0 und 1 vorkommen:

```
fun evala (nil,a)  = a
|   evala (0::t,a) = evala(t, 2*a)
|   evala (1::t,a) = evala(t, 2*a + 1)
|   evala (_) = raise illegaldigit;

fun eval l = evala (l, 0);

eval [1,0,1,0,1,0] handle illegaldigit => ~1;
eval [1,1] handle illegaldigit => ~1;
eval [1,2,3] handle illegaldigit => ~1;
```