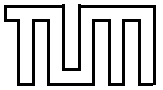
TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK



Lehrstuhl für Sprachen und Beschreibungsstrukturen WS 2008/09
Einführung in die Informatik 2 **Übungsblatt 8**
Prof. Dr. Helmut Seidl, T. M. Gawlitza, S. Pott, M. Schwarz 02.12.2008

Abgabe: 09.12.2008 (vor der Vorlesung)

Aufgabe 8.1 (H) Höhere Funktionen

Schreiben Sie unter Verwendung der Funktion `List.fold_right` (und ohne Verwendung der Funktionen `List.map` bzw `List.filter`)

a) eine Funktion `map`, sodass `map = List.map` gilt,

b) eine Funktion `filter`, sodass `filter = List.filter` gilt.

Lösungsvorschlag 8.1

a)
```
open List

let map f l =
  fold_right (fun x l' -> (f x)::l') l []
```

b)
```
open List

let filter p l =
  fold_right
    (fun x l -> if p x then x::l else l)
    l
    []
```

Aufgabe 8.2 (H) MiniJava-Interpreter

Diese Aufgabe ist eine Fortführung der Aufgabe 7.3. Ziel ist es, einen Interpreter für eine einfache imperative Sprache in OCaml zu implementieren. Die boolschen Ausdrücke `b` dieser Sprache sind durch folgende Grammatik spezifiziert:

$$\texttt{b} ::= \texttt{Not(b)} \mid \texttt{And(b, b)} \mid \texttt{Eq(e, e)} \mid \texttt{Lt(e, e)},$$

wobei `e` die arithmetischen Ausdrücke von Aufgabe 7.3 bezeichnen.

a) Der Ausdruck `Not(b)` wertet sich genau dann zu `true` aus, wenn sich der Ausdruck `b` zu `false` auswertet.

b) Der Ausdruck $\texttt{And}(\texttt{b}_1, \texttt{b}_2)$ wertet sich genau dann zu `true` aus, wenn sich die beiden Ausdrücke $\texttt{b}_1$ und $\texttt{b}_2$ zu `true` auswerten.

c) Der Ausdruck $\texttt{Eq}(\texttt{e}_1, \texttt{e}_2)$ wertet sich genau dann zu `true` aus, wenn sich die Ausdrücke $\texttt{e}_1$ und $\texttt{e}_2$ zu dem selben Integer-Wert auswerten.

d) Der Ausdruck $\texttt{Lt}(\texttt{e}_1, \texttt{e}_2)$ wertet sich genau dann zu `true` aus, wenn sich $\texttt{e}_1$ zu einen kleineren Wert als $\texttt{e}_2$ auswertet.

Die Anweisungen `s` dieser Sprache sind durch folgende Grammatik spezifiziert:

`s` ::= `Assign(<string>, e)` | `Read(<string>)` | `Write(e)` | `If(b, s, s)` | `While(b, s)` | `Seq(s, s)`.

Die Semantik der Anweisungen ist wie folgt spezifiziert.

a) Die Anweisung `Assign(x, e)` entspricht der MiniJava-Anweisung `x = e`.

b) Die Anweisung `Read(x)` entspricht der MiniJava-Anweisung `x = read()`.

c) Die Anweisung `Write(e)` entspricht der MiniJava-Anweisung `Write(e)`.

d) Die Anweisung $\texttt{If}(\texttt{b}, \texttt{s}_1, \texttt{s}_2)$ entspricht der MiniJava-Anweisung `if(b)` $\texttt{s}_1$ `else` $\texttt{s}_2$.

e) Die Anweisung `While(b, s)` entspricht der MiniJava-Anweisung `while(b) s`.

f) Die Anweisung $\texttt{Seq}(\texttt{s}_1, \texttt{s}_2)$ ist eine Anweisung, die zuerst die Anweisung $\texttt{s}_1$ und dann die Anweisung $\texttt{s}_2$ ausführt.

Ein Programm dieser Sprache ist lediglich eine Anweisung. Beispielsweise ist

```
Seq(Read(n),
Seq(Assign(f,Const(0)),
Seq(Assign(vf,Const(1)),
Seq(While(Lt(Const(0),Var(n)),
  Seq(Assign(tmp,Add(Var(vf),Var(f))),
  Seq(Assign(vf,Var(f)),
  Seq(Assign(f,Var(tmp)),
  Assign(n,Sub(Var(n),Const(1)))))))),
Write(Var(f))))))
```

ein Programm, das die `n`-te Fibonacci-Zahl bestimmt. Dieses Programm entspricht dem folgenden MiniJava-Programm:

```
int n, f, vf, tmp;
n = read();
f = 0;
vf = 1;
while (0 < n) {
  tmp = vf + f;
  vf = f;
  f = tmp;
  n = n - 1;
}
write(f);
```

Schreiben Sie ein OCaml-Programm, das ein solches Programm aus einer Datei einliest und anschließend ausführt. Zur Implementierung sollten Sie wie folgt vorgehen:

a) Definieren Sie einen Typ `bool_expr` zur Repräsentation boolscher Ausdrücke.

b) Definieren Sie eine Funktion `get_bexpr`, die einen Term in einen boolschen Ausdruck umwandelt.

c) Definieren Sie eine Funktion `eval_bool` zur Auswertung boolscher Ausdrücke unter Variablenbelegungen.

d) Definieren Sie einen Typ `stmt` für Anweisungen.

e) Definieren Sie eine Funktion `get_stmt`, die einen Term in eine Anweisung umwandelt.

f) Definieren Sie eine Funktion `run`, die eine Anweisung ausführt. Diese erhält als Parameter ein Statement `s` sowie eine *Variablenbelegung* `sigma` und liefert eine *Variablenbelegung* zurück. Der Rückgabewert entspricht der *Variablenbelegung* nach Ausführung des Statements unter der Annahme, dass vor Ausführung des Statements die Variablenbelegung `sigma` aktuell war.

g) Vervollständigen Sie Ihre Implementierung zu einem Interpreter. **Hinweis:** Auf den ersten Kommandozeilen-Parameter kann über `Sys.argv.(1)` zugegriffen werden.

h) Testen Sie Ihren Interpreter anhand des oben genannten Beispiels.

Hinweis: Verwenden Sie die OCaml-Funktionen `string_of_int`, `print_string` und `read_int`.

Lösungsvorschlag 8.2

```
(* Einschliesslich der Loesungen zu Aufgabe 7.3 *)

open Mo
exception ParseError of string

type var = string
type expr =
    Const of int
  | Var   of var
  | Add   of expr * expr
  | Sub   of expr * expr
  | Mul   of expr * expr
  | Div   of expr * expr
type bexpr =
    Not of bexpr
  | And of bexpr * bexpr
  | Eq  of expr * expr
  | Lt  of expr * expr
type stmt =
    Assign of var * expr
  | Read   of var
  | Write  of expr
  | If     of bexpr * stmt * stmt
  | While  of bexpr * stmt
  | Seq    of stmt * stmt

let rec get_expr t =
  match t with
    Node("Const", [Node(x,[])]) -> Const(int_of_string x)
  | Node("Var", [Node(x,[])])   -> Var(x)
  | Node("Add", [a1;a2])        -> Add(get_expr a1, get_expr a2)
  | Node("Sub", [a1;a2])        -> Sub(get_expr a1, get_expr a2)
  | Node("Mul", [a1;a2])        -> Mul(get_expr a1, get_expr a2)
  | Node("Div", [a1;a2])        -> Div(get_expr a1, get_expr a2)
  | _                           -> raise (ParseError (string_from_term t))
let rec get_bexpr t =
  match t with
    Node("Not", [a])     -> Not(get_bexpr a)
  | Node("And", [a1;a2]) -> And(get_bexpr a1, get_bexpr a2)
  | Node("Lt", [a1;a2])  -> Lt(get_expr a1, get_expr a2)
  | Node("Eq", [a1;a2])  -> Eq(get_expr a1, get_expr a2)
  | _                    -> raise (ParseError (string_from_term t))
let rec get_stmt t =
  match t with
    Node("Assign",[Node(v,[]);e]) -> Assign(v,get_expr e)
  | Node("Read",[Node(v,[])])     -> Read(v)
  | Node("Write",[e])             -> Write(get_expr e)
  | Node("If",[b;s1;s2])          -> If(get_bexpr b, get_stmt s1, get_stmt s2)
  | Node("While",[b;s])           -> While(get_bexpr b, get_stmt s)
  | Node("Seq",[s1;s2])           -> Seq(get_stmt s1, get_stmt s2)
  | _                             -> raise (ParseError (string_from_term t))

let rec eval rho = function
     Add (e1,e2) -> (eval rho e1) + (eval rho e2)
   | Sub (e1,e2) -> (eval rho e1) - (eval rho e2)
```

```
   | Mul (e1,e2) -> (eval rho e1) * (eval rho e2)
   | Div (e1,e2) -> (eval rho e1) / (eval rho e2)
   | Const i     -> i
   | Var v       -> rho v
let rec beval rho = function
    Not b    -> not (beval rho b)
  | And(a,b) -> (beval rho a) && (beval rho b)
  | Eq(a,b)  -> (eval rho a) = (eval rho b)
  | Lt(a,b)  -> (eval rho a) < (eval rho b)
let update rho x v y = if x = y then v else rho y
let rec run s rho =
  match s with
      Assign(x,expr)  -> update rho x (eval rho expr)
    | Read(x)         -> print_string "Zahl = "; update rho x (read_int ())
    | Write(expr)     -> print_string (string_of_int (eval rho expr));
                         print_string "\n"; rho
    | If(b,s1,s2)     -> if beval rho b then run s1 rho else run s2 rho
    | While(b,s')     -> if beval rho b then run s (run s' rho) else rho
    | Seq(s1,s2)      -> run s2 (run s1 rho)
let run p = run p (fun _ -> 0)

let _ = run (get_stmt (term_from_file Sys.argv.(1)))
```

Aufgabe 8.3 (P) Verzeichnisstruktur mithilfe polymorpher Typen

In dieser Aufgabe wollen wir den Umgang mit polymorphen Typen üben. Sie dürfen und sollten in dieser Aufgabe die Listenfunktionale `fold_left`, `map` und `filter` sinnvoll einsetzen. An dieser Stelle sei auch auf die OCaml-Referenz verwiesen:

http://caml.inria.fr/distrib/ocaml-3.10/ocaml-3.10-refman.pdf

a) Definieren Sie einen OCaml-Typ `'a dir` zur Repräsentation von Verzeichnisstrukturen. In einer Verzeichnisstruktur vom Typ `'a dir` sollen Daten vom Typ `'a` hierarchisch organisiert werden können. Der Typ `'a` könnte zum Beispiel ein Typ zur Repräsentation von E-Mails sein. Ein Verzeichnis besteht aus einem Namen, einer Liste von Werten vom Typ `'a` und einer Liste von Unterverzeichnissen.

b) Definieren Sie eine Funktion `search : 'a dir -> ('a -> bool) -> 'a list`, die als Argumente eine Verzeichnisstruktur `d` und ein Prädikat `p` erhält. Der Aufruf `search d p` liefert schließlich alle in der Verzeichnisstruktur organisierten Inhalte, die das Prädikat `p` erfüllen.

c) Definieren Sie eine Funktion `mkdir : string -> 'a dir -> 'a dir`. Der Aufruf `mkdir n d` soll ein Verzeichnis mit Namen `n` im Verzeichnis `d` anlegen. Falls das Verzeichnis bereits existiert, so soll nichts geschehen.

d) Definieren Sie eine Funktion `find_and_apply : ('a dir -> 'a dir) -> 'a dir -> string list -> 'a dir`. Der Aufruf `find_and_apply f d p` soll auf das durch den Pfad `p` beschriebenen Unterverzeichnis die Funktion `f` anwenden, die dieses Verzeichnis unter Umständen verändert.

e) Definieren Sie eine Funktion `mkdir : 'a dir -> string list -> string -> 'a dir`. Ein Aufruf `mkdir d p n` soll in der Verzeichnisstruktur `d` ein Verzeichnis mit dem Namen `n` in dem durch den Pfad `p` bezeichneten Unterverzeichnis anlegen. Ist bereits ein Verzeichnis mit diesem Namen vorhanden, so soll nichts geschehen.

f) Definieren Sie eine Funktion `add : 'a dir -> string list -> 'a -> 'a dir`. Ein Aufruf `add d p c` soll in der Verzeichnisstruktur `d` in dem durch den Pfad `p` bezeichneten Verzeichnis den Inhalt `c` hinzufügen.

Lösungsvorschlag 8.3

```
open List
open String

type 'a dir = Node of string * 'a list * 'a dir list

type mail = {von:string;an:string;text:string}

let meinverzeichnis =
  Node("Posteingang",
    [{von="Thomas";an="Sylvia";text="Hallo␣Sylvia!"};
     {von="Martin";an="Sylvia";text="Hallo␣Sylvia!"}],
    []
  )

(* Teil a *)
```

```
let rec search acc d p =
  let Node(name,content,children) = d in
  let acc = filter p content @ acc in
  fold_left (fun acc c -> search acc c p) acc children

(* Version 2 *)
let rec search acc d p =
  let Node(name,content,children) = d in
  let acc =
    fold_left (fun acc c -> if p c then c::acc else acc) acc content
  in
  fold_left (fun acc c -> search acc c p) acc children
(*-- Version 2 *)

let search d p = search [] d p

let von_thomas =
  let p = function {von="Thomas"} -> true | _ -> false in
  search meinverzeichnis p

(* Teil b*)
let mkdir n d =
  let Node(name,content,children) = d in
  match filter (fun (Node(name',_,_)) -> n = name') children with
    [] -> Node(name,content,Node(n,[],[])::children)
  | _ -> d

(* Teil c *)
let rec find_and_apply f d = function
  [] -> f d
| l::p ->
    let Node(name,content,children) = d in
    let app d' =
      let Node(name',content',children') = d' in
      if name' = l then
        find_and_apply f d' p
      else
        d'
    in
    Node(name,content,map app children)

(* Teil d*)
let mkdir d p n =
  find_and_apply (mkdir n) d p

(* Teil e*)
let add d p c =
  find_and_apply (fun (Node(n,cs,dirs)) -> Node(n,c::cs,dirs)) d p
```