TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Lehrstuhl für Sprachen und Beschreibungsstrukturen
Einführung in die Informatik 2
Prof. Dr. Helmut Seidl, T. M. Gawlitza, S. Pott, M. Schwarz

WS 2008/09
Übungsblatt 8
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.

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 $\texttt{And(b}_1\texttt{, b}_2\texttt{)}$ 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 $\texttt{Eq(e}_1\texttt{, e}_2\texttt{)}$ 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 $\texttt{Lt(e}_1\texttt{, e}_2\texttt{)}$ 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:

$\texttt{s} ::= \texttt{Assign(<string>, e)} \mid \texttt{Read(<string>)} \mid \texttt{Write(e)} \mid \texttt{If(b, s, s)} \mid \texttt{While(b, s)} \mid \texttt{Seq(s, s)}.$

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

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`.

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.