TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK
Lehrstuhl für Sprachen und Beschreibungsstrukturen
Einführung in die Informatik 2
Prof. Dr. Helmut Seidl, Ralf Vogler, Stefan Schulze Frielinghaus

WS 2015/16
Übungsblatt 9

OCaml-Hausaufgaben die nicht kompilieren werden nicht gewertet! Solltet Ihr einmal in die Situtation kommen, dass sich ein Fehler in einen Teil Eurer Abgabe eingeschlichen hat, der zur Folge hat, dass sich die gesamte Datei nicht mehr kompilieren lässt, dann kommentiert diesen Teil aus und ersetzt ihn durch den Standardwert aus der Angabe (z.B. let f x = todo ()). Andernfalls wird die gesamte OCaml-Hausaufgabe mit Null Punkten gewertet.

Aufgabe 9.1 OCaml-Hausaufgabe

Die Vorlagen und Lösungen zu den Programmieraufgaben sind ab sofort über `https://github.com/vogler/info2-ha` verfügbar, damit Änderungen (sollte es während der Bearbeitungszeit welche geben) einfacher nachvollziehbar sind.

```
git clone https://github.com/vogler/info2-ha.git && cd info2-ha
git pull # update
git log # changes
```

Zur Abgabe melden Sie sich mit Ihrer TUM-Kennung auf `https://vmnipkow3.in.tum.de` an und laden ihre `ha5.ml`-Datei (nicht umbenennen!) hoch. Die Ergebnisse der Tests sind sichtbar sobald die Abgabe abgearbeitet wurde. Die endgültige Bewertung erfolgt aber erst nach der Frist.

Die Datei `batteries.ml` enthält die bisher implementierten Funktionen und ist auch in der Testumgebung verfügbar (muss nicht hochgeladen werden).

Lösungsvorschlag 9.1

Siehe github.

Aufgabe 9.2 OCaml-Tutoraufgaben

1. **Lambdas:** Machen Sie sich mit den folgenden Lambda-Ausdrücken vertraut und inferrieren den jeweiligen Typ. Welche Semantik haben die jeweiligen Ausdrücke?

   1.1. `fun x -> fun y -> fun z -> x (y z)`

   1.2. `fun x -> fun y -> fun z -> z (x + (fst y))`

   1.3.
   ```
   fun x -> fun y -> let rec z = fun l ->
           match l with
           | [] -> []
           | u::us -> (x u) :: (z us)
           in z y
   ```

2. **Exceptions:**

   2.1. Diskutieren Sie über die Unterschiede bei Exceptions in OCaml und Java. Was sind Vor-/Nachteile von Exceptions?

2.2. Schreiben Sie eine Funktion mit dem Typen `('a -> bool) -> 'a list -> 'a`, die das erste Element einer Liste zurück gibt, bei welchem ein Prädikat wahr ist. Das Prädikat wird als erster Parameter erwartet. Falls die Liste kein Element enthält auf welchem das Prädikat wahr ist, soll die Exception `Not_found` geworfen werden.

2.3. Schreiben Sie eine Funktion `combine : 'a list -> 'b list -> ('a * 'b) list` die zwei Listen als Parameter nimmt und diese elementweise kombiniert. D.h. `combine [a1; ...; an] [b1; ...; bn]` liefert als Wert `[(a1,b1); ...; (an,bn)]`. Für Listen unterschiedlicher Länge soll eine Ausnahme `Invalid_argument` geworfen werden.

Lösungsvorschlag 9.2

1. **Lambdas:**

   1.1. Funktionskomposition: `('a -> 'b) -> ('c -> 'a) -> 'c -> 'b`
   In den Hausaufgaben verwendeten wir bereits `let (%) g f x = g (f x)` um die Komposition als `g%f` schreiben zu können.

   1.2. `int -> int * 'a -> (int -> 'b) -> 'b`

   1.3. `List.map : ('a -> 'b) -> 'a list -> 'b list`

2. **Exceptions:**

   2.1. In OCaml sind Exceptions ein erweiterbarer Summentyp:

```
# exception A;;
exception A
# exception B;;
exception B
# [A; B];;
- : exn list = [A; B]
# raise;;
- : exn -> 'a = <fun>
# List.map (function A -> 1 | B -> 2 | _ -> 3) [A; B; Not_found];;
- : int list = [1; 2; 3]
```

   Die Funktion `raise` ist jedoch speziell und kann nicht selbst implementiert werden. Sie bricht die Ausführung ab und geht die Aufrufhierarchie nach oben bis die Exception gefangen wird oder man die Wurzel erreicht hat und das ganze Programm abbricht. In Java ist dies ganz ähnlich, allerdings gibt es in OCaml nur `unchecked` Exceptions (d.h. ob eine Funktion eine Exception wirft oder nicht ist nicht im Typen widergespiegelt), während es in Java auch `checked` Exceptions gibt, welche gefangen werden müssen. Allerdings kann man in OCaml mit eigenen Typen oder bereits definierten wie z.B. `option` einfach `checked` Exceptions simulieren.

   Gegen `checked` Exceptions gibt es nichts einzuwenden — `unchecked` Exceptions haben jedoch den Nachteil dass der Compiler sie nicht überprüft und sollten vermieden werden. Case in point: `NullPointerException` in Java.

   2.2. `List.exists` ≡

```
fun p -> fun l -> let rec fold = fun l ->
      match l with
```

```
    | [] -> raise Not_found
    | x::xs -> if p x then x else fold xs
    in fold l
```

2.3.
```
let rec combine x y =
    match x, y with
    | [], [] -> []
    | x::xs, y::ys -> (x, y) :: (combine xs ys)
    | _ -> raise (Invalid_argument "combine lists of different length")
```

Die obige Implementierung ist nicht endrekursiv. Eine endrekursive Implementierung sieht wie folgt aus:

```
let combine x y =
  let rec helper a = function
    | [], [] -> a
    | x::xs, y::ys -> helper ((x, y) :: a) (xs, ys)
    | _ -> raise (Invalid_argument "combine lists of different length")
  in helper [] (x, y) |> List.rev
```

Hat aber den Nachteil, dass die Liste umgedreht werden muss, sofern die kanonische Ordnung erhalten bleiben soll.