TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Lehrstuhl für Sprachen und Beschreibungsstrukturen
Prof. Dr. Helmut Seidl, Ralf Vogler, Stefan Schulze Frielinghaus

WS 2015/16
Einführung in die Informatik 2 **Übungsblatt 8**

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 8.1 OCaml-Hausaufgabe

Zur Abgabe melden Sie sich mit Ihrer TUM-Kennung auf `https://vmnipkow3.in.tum.de` an und laden ihre `ha4.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).

In der Datei gibt es zwei Module mit Funktionen die zu implementieren sind:

a) `Map`: Mehr Funktionene zu Maps als Binärbäume.

b) `Json`: Funktionen zum Arbeiten mit JSON-Dokumenten.

Lösungsvorschlag 8.1

Siehe Moodle.

Aufgabe 8.2 OCaml-Tutoraufgaben

a) Im folgenden definieren wir einige Hilfsfunktion zum Umgang mit `option`. Machen Sie sich klar wie Sie mit `>>=` Ihren Code verständlicher machen können.

```
module Option = struct
  let some x = Some x
  let is_some x = x<>None
  let is_none x = x=None

  let map f = function
    | Some x -> Some (f x)
    | None -> None

  let bind o f = match o with
    | Some x -> f x
    | None -> None
  let (>>=) = bind
```

```
  let filter p = function
    | Some x when p x -> Some x
    | _ -> None

  let default x = function
    | Some y -> y
    | None -> x
  (* Some 1 |? 0 = 1, None |? 0 = 0, None |? Some 1 |? 0 = 1*)
  let (|?) x y = default y x

  let try_some f x = try Some (f x) with _ -> None
end
```

b) Wir nutzen im Folgenden Binärbäume um eine Abbildung von Schlüsseln auf Werte darzustellen. Verstehen Sie den Typ und implementieren Sie folgende Grundfunktionen:

```
type ('k, 'v) t = Empty | Node of 'k * 'v * ('k, 'v) t * ('k, 'v) t
val empty : ('k, 'v) t
(* {  }, { k1 -> v1 }, { k1 -> v1, k2 -> v2 } *)
val show : ('k -> string) -> ('v -> string) -> ('k, 'v) t -> string
val add : 'k -> 'v -> ('k, 'v) t -> ('k, 'v) t
val find : 'k -> ('k, 'v) t -> 'v option
```

Lösungsvorschlag 8.2

a) Funktionen die `option` zurückgeben, können nun sinnvoll komponiert werden:

```
  val f1 : a -> b option
  val f2 : b -> c option
  val f3 : c -> d option

  let f : a -> d option = fun x -> f1 x >>= f2 >>= f3
```

b)
```
let empty = Empty

let show sk sv m =
  let sp k v = sk k ^ " -> " ^ sv v in
  let (|^) a b = if a <> "" then if b <> "" then a ^ ", " ^ b else a else b in
  let rec f = function
    | Empty -> ""
    | Node (k, v, l, r) -> sp k v |^ f l |^ f r
  in
  "{ " ^ f m ^ " }"

(* alternativ: *)
let rec to_list = function
  | Empty -> []
  | Node (k, v, l, r) -> (k,v) :: to_list l @ to_list r
let show sk sv m = to_list m |> MyList.map (fun (k,v) -> sk k ^ " -> " ^ sv v)
  |> String.concat ", " |> fun s -> "{ " ^ s ^ " }"
```

```
let rec add k v m = match m with
  | Empty -> Node (k, v, Empty, Empty)
  | Node (k', v', l, r) ->
     if k = k' then m
     else if k < k' then Node (k', v', add k v l, r)
     else Node (k', v', l, add k v r)

let rec find k = function
  | Empty -> None
  | Node (k', v', l, r) ->
      if k = k' then Some v'
      else find k (if k < k' then l else r)
```