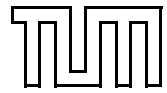
TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK



WS 11/12
Einführung in die Informatik II **Übungsblatt 8**
Univ.-Prof. Dr. Andrey Rybalchenko, M.Sc. Ruslán Ledesma Garza 20.12.2011

Dieses Blatt behandelt Kapitel 14.1 – 14.4 und 14.7 aus dem Buch zur Vorlesung. Lesen Sie diese Kapitel!

Aufgabe 14.1

Sei die Struktur ISet gegeben.

```
signature ISET =
  sig
    type set
    val set : int list → set
    val union : set → set → set
    val subset : set → set → bool
  end

structure ISet :> ISET =
struct
  type set = int list
  fun set xs = xs
  fun union xs ys = xs@ys
  fun elem ys x = List.exists (fn y ⇒ y=x) ys
  fun subset xs ys = List.all (elem ys) xs
end
```

a) Schreiben Sie eine Prozedur `member : int → ISet.set → bool`, die testet, ob eine Zahl in einer Menge enthalten ist.

b) Schreiben Sie eine Prozedur `empty : ISet.set → bool`, die testet, ob eine Menge leer ist.

c) Schreiben Sie eine Prozedur `equal : ISet.set → ISet.set → bool`, die testet, ob zwei Mengen gleich sind.

Aufgabe 14.2

Warum ist die folgende Deklaration unzulässig?

```
structure S :> sig eqtype t end = struct
  type t = int → int
end
```

Aufgabe 14.3

Die Struktur `ISet` aus Abbildung 14.1 auf S. 281 implementiert die abstrakte Datenstruktur ISet korrekt aber ineffizient. Zum einen kann die eine Menge Liste Elemente mehrfach enthalten (z.B. bei `ISet.set [1,1]`). Zum anderen hat die Operation `subset` die quadratische Laufzeit (in der Summe der Länge der darstellenden Listen).

Schreiben Sie eine Struktur `ISSet`, die Mengen mit strikt sortierten Listen implementiert (§5.4 und 5.5). Achten Sie darauf, dass Sie die Operation `set` mit linear-logarithmischer und die Operationen `union` und `subset` mit linearer Laufzeit realisieren.

Aufgabe 14.5 (Umgebungen)

Unter einer Umgebung wollen wir wie in §6.4.2 eine Funktion verstehen, die durch Strings realisierte Bezeichner auf bestimmte Werte abbildet. Deklarieren Sie eine Struktur `Env`, die Umgebungen wie folgt implementiert:

```
type 'a env
exception Unbound
val env : (string * 'a) list → 'a env
val lookup : 'a env → string → 'a
val update : 'a env → string → 'a → 'a env
```

- `env` liefert zu einer Liste von Paaren die entsprechende Umgebung. Wenn die Liste zu einem String mehr als ein Paar enthält, soll nur eines der Paare verwendet werden.
- `lookup` liefert zu einer Umgebun `f` und einem String `s` den Wert `f s`. Falls `f` auf `s` nicht definiert ist, wirft `lookup` die Ausnahme `Unbound`.
- `update` liefert zu einer Umgebung `f`, einem String `s` und einem Wert `x` die Umgebung `f [s:=x]`.

Aufgabe 14.6 (Listen)

Deklarieren Sie eine Struktur `MyList`, die Listen mit der folgenden Signatur bereitstellt:

```
eqtype 'a list
val empty : 'a list
val cons : 'a → 'a list → 'a list
val null : 'a list → bool
val hd : 'a list → 'a
val tl : 'a list → 'a list
```

Achten Sie darauf, dass die Struktur alle Operationen mit konstanter Laufzeit implementiert. Machen Sie sich klar, dass alle anderen Listenoperationen mit den Operationen der Struktur programmiert werden können. Zeigen Sie das am Beispiel von `foldl`.

Aufgabe 14.7 * (Puzzle)

Für eine abstrakte Datenstruktur stellt sich oft die Frage, ob eine gewünschte Operation mit den Operationen der Struktur programmiert werden kann. Dazu wollen wir eine abstrakte Datenstruktur Puzzle betrachten, die endliche Mengen ganzer Zahlen gemäß der folgenden Signatur bereitstellt:

```
type set
val set : int list → set
val find : (int → bool) → set → int option
```

Die Operation `set` soll zu einer Liste die entsprechende Menge liefern, und die Operation `find` soll zu einer Eigenschaft und einer Menge ein Element der Menge liefern, das die Eigenschaft erfüllt. Die Operation find ruft den Eigenschafts-Test nur auf Elementen der Menge auf.

a) Deklarieren Sie eine Struktur `Puzzle`, die die abstrakte Datenstruktur Puzzle realisiert.

b) Schreiben Sie mithilfe der Operationen von `Puzzle` eine Prozedur `set → int list`, die die Elemente einer Menge als Liste liefert.

Aufgabe 14.8

Schreiben Sie eine Prozedur `vector2list : 'a vector → 'a list`, die zu einem Vektor die entsprechende Liste liefert.

Aufgabe 14.9

Schreiben Sie eine Prozedur `cons : 'a → 'a vector → 'a vector`, die zu `x` und $[x_1, \ldots, x_n]$ den Vektor $[x, x_1, \ldots, x_n]$ liefert, Verwenden Sie dabei `Vector.concat`.

Aufgabe 14.15 (Optional)

Schreiben Sie polymorphe Prozeduren `ssort`, `smerge` und `ssublist`, wie sie für die Deklaration des Funktors `Set` in Abbildung 14.6 erforderlich sind.