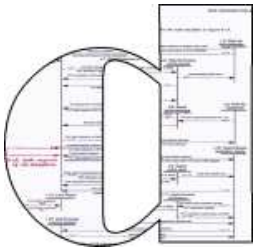
DEPENDABLE SYSTEMS AND SOFTWARE

Fachrichtung 6.2 — Informatik
Prof. Dr.-Ing. Holger Hermanns
Dipl.-Inform. Lijun Zhang



Übungsblatt 15 (Programmierung I)

Lesen Sie im Skript: Kapitel 14

Aufgabe 15.1 Schreiben Sie eine Prozedur $vector2list : \alpha\ vector \to \alpha\ list$, die zu einem Vektor die entsprechende Liste liefert.

Aufgabe 15.2 Schreiben Sie eine Prozedur $cons : \alpha\ vector \to \alpha\ vector$, die zu x und $[x_1, \ldots, x_n]$ den Vektor $[x, x_1, \ldots, x_n]$ liefert. Verwenden Sie dabei `Vector.concat`.

Aufgabe 15.3 Sie sollen endliche Funktionen in Standard ML mit einem abstrakten Datentyp realisieren. Als Argumente für die Funktionen sollen nur die als *Schlüssel* bezeichneten Werte eines vorgegebenen Typs `key` möglich sein. Für `key` soll eine Vergleichsprozedur `compare: key * key` $\to$ `order` gegeben sein. Der abstrakte Datentyp soll die folgende Signatur haben:

```
type 'a map
val empty  : 'a map
val insert : key -> 'a -> 'a map -> 'a map
val lookup : key -> 'a map -> 'a option
```

Die Semantik soll die folgenden Bedingungen erfüllen:

- Für jeden Typ t stellen die Werte des abstrakten Typs `t map` endliche Funktionen `key` $\to t$ dar.
- `empty` stellt die leere Funktion dar.
- `insert` liefert zu einer endlichen Funktion f, einem Schlüssel k und einem Wert a die Funktion $f[k := a]$.
- `lookup` liefert zu einem Schlüssel k und einer endlichen Funktion f den Wert $SOME(f\ k)$, falls f auf k definiert ist, und $NONE$ sonst.

a) Deklarieren Sie für `key=int` eine entsprechende Signatur *IMAP*.
b) Deklarieren Sie für `key=int` eine entsprechende Struktur *IMap*, die den abstrakten Typkonstruktor durch die folgende Deklaration realisiert:

```
type 'a map = (int * 'a) list
```

c) Geben Sie eine Deklaration an, die den Bezeichner `m` an die Funktion $\{(1,1),\ (5,3)\}$ bindet.

d) Deklarieren Sie einen Funktor `Map`, der zu den Argumenten

```
type key
val  compare : key * key -> order
```

eine Struktur liefert, die endliche Funktionen für `key` implementiert.

Aufgabe 15.4 Schreiben Sie polymorphe Prozeduren *ssort*, *smerge* und *ssublist*, so wie sie für die Deklaration des Funktors *Set* in Abbildung 14.6 erforderlich sind.

Aufgabe 15.5 Deklarieren sie mit Hilfe des Funktors *Set* eine Struktur *AdjSet*, welche Ihnen erlaubt, Adjazenzmengen für die Sprache L darzustellen. Dazu benötigen Sie eine Prozedur

`compare: (mark * lterm) * (mark * lterm)` $\rightarrow$ `order`

welche Paare aus Markierungen und Prozessen aus L ordnet. Die dafür nötige Ordnung `compM` auf Markierungen kann wie folgt gewählt sein.

```
fun compM (Tau,Tau) = EQUAL
  | compM (Tau,_) = LESS
  | compM (_,Tau) = GREATER
  | compM (In(u) , In(v)) = String.compare(u,v)
  | compM (Out(u),Out(v)) = String.compare(u,v)
  | compM (In _ ,_) = LESS
  | compM (Out _ ,_) = GREATER
```

Aufgabe 15.6 Variieren Sie die Ihnen aus der Modellimplementierung bekannte Prozedur *step*, welche die Semantik von L implementiert, so, daß Adjazenmengen nicht durch Listen dargestellt werden, sondern stattdessen durch die Struktur *AdjSet* realisiert werden.

Hinweis: Sofern Sie das Resultat in Ihrer Implementierung ausprobieren wollen, müssen Sie bei der Kompilierung beachten, daß der Funktor aus Aufgabe 15.4 für *mosml* in einer separaten Datei ausgelagert sein muss. Sie können die beiden Dateien (Windows) getrennt mit `mosmlc -toplevel <functor.sml>` und `mosmlc -toplevel <functor>.ui <file.sml> -o <binary.exe>` kompilieren, oder auf einen Schlag mit:
`mosmlc -toplevel <functor.sml> <file.sml> -o <binary.exe>`.