9. Übungsblatt zu Programmierung

Prof. Gert Smolka, Thorsten Brunklaus
www.ps.uni-sb.de/courses/prog-ws00/

Abgabe: 19. Januar 2001 vor der Vorlesung (per Email)

Allgemeine Hinweise: Die Übungsblätter sollen in Zweiergruppen bearbeitet werden. Die Lösungen schicken Sie bitte bis Freitag vor der Vorlesung per Email an Ihren Übungsgruppenleiter. Jede Gruppe soll nur eine Lösung einreichen, versehen mit den Namen und den Matrikelnummern der Gruppenmitglieder.
Tipp: Es empfiehlt sich, die Datenstrukturen zunächst ohne Strukturdeklaration zu implementieren und zu testen. Ist dies erfolgreich, kann der Rahmen nachträglich konstruiert werden.

Um das lästige Laden von Standardstrukturen zu vermeiden, können sie den Moscow ML Interpreter mit der Option `-P full` aufrufen. Dadurch werden die wichtigsten Standardstrukturen gleich beim Starten geladen. `mosml file` führt die in `file` enthaltenen Deklarationen aus, die dann im Interpreter zur Verfügung stehen (z.B. `mosml -P full x.sml`).

Aufgabe 9.1: Endliche Funktionen (5+5+5+5+5+5) In Abschnitt 9.2 haben Sie gelernt, wie man endliche Mengen implementiert. Sie sollen nun mit denselben Techniken endliche Funktionen (siehe Abschnitt 2.3) implementieren. Dies soll für einen gegebenen Typ *key* gemäß der Signatur

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

und der folgenden Spezifikation erfolgen:

$$\begin{aligned} \mathit{empty} &= \emptyset \\ \mathit{insert}(k, a, f) &= f[a/k] \\ \mathit{lookup}(k, f) &= \text{if } k \in \mathit{Dom}\ f \text{ then } \mathit{SOME}(f(k)) \text{ else } \mathit{NONE} \end{aligned}$$

(a) Deklarieren Sie eine Struktur `IMap`, die endliche Funktionen für $key = int$ implementiert. Deklarieren Sie dazu zunächst eine passende Signatur `IMAP`.

(b) Deklarieren Sie einen Bezeichner

```
val m : int IMap.map
```

der an die folgende Funktion gebunden ist:

$$\{1 \mapsto 1,\ 5 \mapsto 3,\ 6 \mapsto 0,\ 7 \mapsto 0\}$$

(c) Geben Sie Typ- und Wertdeklarationen an, die denselben Effekt wie die Deklaration `open IMap` erzielen.

(d) Deklarieren Sie mithilfe der Struktur `IMap` eine Struktur `ISet`, die endliche Mengen gemäß der Signatur `ISET` implementiert (siehe Abschnitt 9.2).

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

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

eine Struktur liefert, die endliche Funktionen implementiert.

(f) Deklarieren Sie mithilfe des Funktors `Map` eine Struktur `IMap`, die endliche Funktionen für *key* = *int* implementiert.

Aufgabe 9.2: Priorisierte Schlangen (5+5+5+5) In Abschnitt 9.3 haben Sie gelernt, was man unter Schlangen versteht und wie man sie implementiert. Sie sollen jetzt sogenannte *priorisierte Schlangen* implementieren, deren Einträge Paare (k, v) sind, die aus einem *Schlüssel* k und einem Wert v bestehen. Für die Menge der Schlüssel muss eine Ordnung vorgegeben sein. Die Einträge einer priorisierten Schlange werden lexikografisch nach der Ordnung der Schlüssel und (für gleiche Schlüssel) nach dem Alter der Einträge geordnet. Das erste Element einer nichtleeren priorisierten Schlange ist also der Eintrag mit dem kleinsten Schlüssel, der schon am längsten in der Schlange ist. Die Einträge

```
(4, "Jim"), (2, "Maria"), (3, "Monica"), (2, "Tom")
```

sollen also die priorisierte Schlange

```
[(2, "Maria"), (2, "Tom"), (3, "Monica"), (4, "Jim")]
```

ergeben, wenn sie von links nach rechts erfolgen.
Für einen gegeben Typ *key* sollen priorisierte Schlangen gemäß der folgenden Signatur implementiert werden:

```
type 'a pqueue
val empty  : 'a pqueue
val insert : key * 'a * 'a pqueue -> 'a pqueue
val head   : 'a pqueue -> key * 'a  (* Empty *)
val tail   : 'a pqueue -> 'a pqueue     (* Empty *)
```

(a) Deklarieren Sie eine Struktur `IPQueue`, die priorisierte Schlangen für *key* = *int* implementiert. Deklarieren Sie dazu zunächst eine passende Signatur `IPQUEUE`.

(b) Deklarieren Sie einen Bezeichner

```
val q : string IPQueue.pqueue
```

der an die folgende Schlange gebunden ist:

```
[(2, "Maria"), (2, "Tom"), (3, "Monica"), (4, "Jim")]
```

(c) Deklarieren Sie einen Funktor `PQueue`, der für zwei Argumente

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

eine Struktur liefert, die priorisierte Schlangen implementiert.

(d) Deklarieren Sie mithilfe des Funktors `PQueue` eine Struktur `IPQueue`, die priorisierte Schlangen für *key* = *int* implementiert.