9. Übungsblatt zu Programmierung WS 2002 / 03

Prof. Dr. Gert Smolka und Dipl.-Inform. Thorsten Brunklaus
www.ps.uni-sb.de/courses/prog-ws02/

Abgabe: Montag, 6. Januar 2003

Schicken Sie Ihre Lösungen per E-Mail an Ihren Übungsgruppenleiter. Geben Sie in der E-Mail zuerst ihren Name und die Nummer Ihrer Übungsgruppe an. Bitte reichen Sie alle Lösungen mit nur einer E-Mail ein.

Zusammengefügt sollen Ihre Lösungen ein semantisch zulässiges Programm ergeben. Prüfen Sie das, indem Sie ihre Lösungen in eine Datei `x.sml` schreiben und diese mittels der Eingabe `use "x.sml"` in einen neu gestarteten Interpreter einlesen. Wenn Sie eine Standardstruktur X benötigen, laden Sie diese am Anfang der Datei `x.sml` mit einer Zeile `;load "X";`. Trennen Sie ihre Lösungen durch Kommentare der Form `(* Aufgabe 9.1 *)`.

Tipp: Es empfiehlt sich, die Datenstrukturen zunächst ohne Strukturdeklaration zu implementieren und zu testen. Ist dies erfolgreich, kann der Rahmen nachträglich konstruiert werden.

Aufgabe 9.1: Endliche Mengen mit Gleicheit ($5 + 10$**)** Im Skript finden Sie eine Implementierung eines abstrakten Typs für endliche Mengen von ganzen Zahlen. Dabei wird eine Signatur `ISET` und eine Struktur `ISet` verwendet. Sie sollen diese Typabstraktion um eine Operation `eq` erweitern, die die Gleichheit zweier Mengen testet.

(a) Geben Sie eine Signatur `ISET'` an, die die Signatur `ISET` um eine passend getypte Operation `eq` erweitert.

(b) Geben Sie eine Struktur `ISet'` an, die die erweiterte Typabstraktion implementiert. Dabei sollen Mengen durch sortierte Listen ohne Doppelauftreten dargestellt werden. Beachten Sie, dass Sie mit dem polymorphen Operator `=` testen können, ob zwei Listen gleich sind.

Aufgabe 9.2: Endliche Funktionen ($5 + 10 + 5 + 5 + 10 + 10 + 5$**)** Sie sollen eine Typabstraktion für endliche Funktionen 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 Spezifikation

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

erfolgen. Die Definition der Notation $f[k := a]$ finden Sie in Kapitel 2 (Adjunktion von Funktionen).

(a) Sei die Deklaration `type key = int` gegeben. Deklarieren Sie eine Signatur `IMAP` für die gerade beschriebene Typabstraktion.

(b) Deklarieren Sie eine Struktur `IMap`, die die beschriebene Typabstraktion gemäß der Signatur `IMAP` implementiert. Dabei sollen endliche Funktionen durch Prozeduren dargestellt werden.

(c) Deklarieren Sie einen Bezeichner

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

dessen Wert die folgende Funktion darstellt:

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

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

(e) Deklarieren Sie mithilfe der Struktur `IMap` eine Struktur `ISet`, die endliche Mengen gemäß der Signatur `ISET` (siehe Skript) implementiert. Stellen Sie dabei endliche Mengen durch endliche Funktionen $\mathbb{Z} \rightharpoonup \{()\}$ dar.

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

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

eine Struktur liefert, die endliche Funktionen implementiert.

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

Aufgabe 9.3: Priorisierte Schlangen ($5 + 10 + 5 + 10 + 5$**)** Sie haben gelernt, was man unter Schlangen versteht und wie man sie implementiert. Sie sollen jetzt *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 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. Wenn beispielsweise die Paare

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

in der angebenen Ordnung (von links nach rechts) in die leere Schlange eingetragen werden, soll dies die folgende priorisierte Schlange liefern:

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

(a) Sei die Deklaration `type key = int` gegeben. Deklarieren Sie eine Signatur `IPQUEUE`, die die folgenden Felder spezifiert:

```
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 *)
```

(b) Deklarieren Sie eine Struktur `IPQueue`, die die beschriebene Typabstraktion gemäß der Signatur `IPQUEUE` implementiert.

(c) Deklarieren Sie einen Bezeichner

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

dessen Wert die folgende Schlange darstellt:

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

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

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

eine Struktur liefert, die priorisierte Schlangen implementiert.

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