14. Übungsblatt zu Informatik 1 WS 98/99

Andreas Rossberg, Prof. Gert Smolka

Hinweis: Dieses Übungsblatt dient zur Vorbereitung auf die Klausur und muss nicht mehr abgegeben werden. Die Musterlösung finden Sie am 19. Februar im Netz.

Aufgabe 14.1: Listen

Schreiben Sie folgende Prozeduren:

(a) `average: real list -> real`, mit `average[`x_1`,`$\cdots$`,`x_n`] = (`x_1`*`$\cdots$`*`x_n`)/`n.

(b) `split: 'a list * int -> 'a list * 'a list`, so dass `split(l,n)` eine Liste in zwei Teillisten aufspaltet, von denen die erste die ersten `n` Elemente enthält und die zweite die restlichen. Beispielsweise ist `split([3,1,4,3],2) = ([3,1],[4,3])`. Ist `n<0` oder `n>length l`, so soll die Ausnahme `Subscript` geliefert werden.

(c) `all: ('a -> bool) -> 'a list -> bool`, liefert eine Prozedur, die `true` ergibt genau dann, wenn das übergebene Prädikat für alle Elemente der Liste `true` ergibt.

Aufgabe 14.2: Schleifen

Schreiben Sie eine nicht-rekursive Prozedur `rtab: int -> int list`, die für ein $n \geq 0$ mithilfe einer Schleife die Liste `[`n`,`$\cdots$`,1]` erzeugt.

Aufgabe 14.3: Syntax und Grammatiken

Gegeben sei folgende Grammatik:

$$\begin{array}{llllllllllll} c & ::= & 0 & \mid & 1 & \mid & 2 & \mid & 3 & \mid & 4 & \text{(constant)} \\ e & ::= & c & \mid & (e) & \mid & +\ e_1\ e_2 & \mid & *\ e_1\ e_2 & & & \text{(expression)} \end{array}$$

Betrachten Sie dazu den Ausdruck $a = + * 2\ (+\ 0\ 4)\ * 3\ 1$.

(a) Definieren Sie einen geeigneten Typ `token`, der die Terminale dieser Grammatik repräsentiert. Geben Sie einen Wert des Typs `token list` an, der den Ausdruck a darstellt.

(b) Definieren Sie eine abstrakte Syntax für die Grammatik und geben Sie die Darstellung des Ausdrucks a an.

(c) Schreiben Sie einen Evaluator für Ausdrücke gemäß e.

Aufgabe 14.4: Bäume

Folgender Typ repräsentiert binäre Bäume:

```
datatype 'a btree = Leaf of 'a | Branch of 'a * 'a btree * 'a btree
```

Schreiben Sie folgende Prozeduren:

(a) `depth: 'a btree -> int`, berechnet die Tiefe des Baumes.

(b) `frontier: 'a btree -> 'a list`, liefert die Grenze.

(c) `sum: int btree -> int`, berechnet die Summe aller Markierungen für Bäume über ganzen Zahlen.

Aufgabe 14.5: Gleichheit von Bäumen

Gegeben sei ein Typ für markierungslose binäre Bäume

```
datatype btree = Leaf | Branch of btree * btree
```

Schreiben Sie eine Prozedur

```
equals: btree * btree -> bool
```

die zwei Bäume auf Gleichheit überprüft, ohne dazu den vordefinierten Gleichheitsoperator `=` zu verwenden.

Aufgabe 14.6: Graphen

Knoten von Graphen seien durch folgende Typdeklaration definiert:

```
datatype 'a info = N of 'a * 'a node list
withtype 'a node = 'a info ref
```

Schreiben Sie folgende Prozeduren, und achten Sie dabei auf korrekte Behandlung zyklischer Graphen:

(a) `size: 'a node -> int`, berechnet die Anzahl der vom gegebenen Knoten aus erreichbaren Knoten.

(b) `isReachable: 'a node * 'a node -> bool`, so dass `isReachable(a,b)` ermittelt, ob der Knoten `b` von `a` aus erreichbar ist.

Aufgabe 14.7: Binäre Suche

Schreiben Sie eine Prozedur

```
contains: int array * int -> bool
```

die für eine sortierte Reihung in logarithmischer Zeit ermittelt, ob die gegebene Zahl in der Reihung enthalten ist. Zunächst wird dazu anhand der mittleren Zelle der Reihung entschieden, in welcher Hälfte sich das Element gegebenenfalls befinden muss.

Aufgabe 14.8: Schlangen mit Prioritäten

Ein mögliche Signatur `PQueue` für Schlangen mit Prioritäten ist

```
type 'a pqueue

exception Empty

val new:      ('a * 'a -> bool) -> 'a pqueue
val put:      'a pqueue * 'a -> unit
val get:      'a pqueue -> 'a                  (* Empty *)
val isEmpty:  'a pqueue -> bool
```

Die Prozedur `get` liefert das älteste Element mit höchster Priorität. Die Priorität ist hierbei durch eine frei wählbare Ordnung auf den Elementen selbst gegeben. Implementieren Sie dieses Modul.