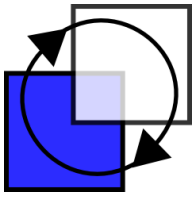
REACTIVE SYSTEMS GROUP

Universität des Saarlandes

Prof. Bernd Finkbeiner, Ph.D.
Markus Rabe, M.Sc.



# Programmierung 1 (SS 2010) - 14. Übungsblatt

http://react.cs.uni-saarland.de/prog1/

Lesen Sie Kapitel 14.6 bis 15.8

### Aufgabe 14.1

Deklarieren Sie mit Hilfe des Funktors *Set* eine Struktur *IntSet*, die Mengen über ganzen Zahlen bereitstellt.

### Aufgabe 14.2

Deklarieren Sie einen Funktor *VSet*, der Mengen über einem geordneten Elementtyp $t$ durch strikt sortierte Vektoren gemäß der folgenden Signatur implementiert:

$$\begin{aligned}&\mathit{type}\ \mathit{set}\\&\mathit{val}\ \mathit{set}\colon t\ \mathit{list} \to \mathit{set}\\&\mathit{val}\ \mathit{subset}\colon \mathit{set} \to \mathit{set} \to \mathit{bool}\\&\mathit{val}\ \mathit{elem}\colon t \to \mathit{set} \to \mathit{bool}\end{aligned}$$

Dabei soll *set* linear-logarithmische, *subset* lineare und *elem* logarithmische Komplexität haben.

### Aufgabe 14.3

Erweitern Sie die effiziente Implementierung von Schlangen um eine Operation

$$\mathit{length}\colon \alpha\ \mathit{queue} \to \mathit{int},$$

die die Länge einer Schlange mit konstanter Laufzeit liefert. Stellen Sie Schlangen dabei durch Tripel $(q, r, n)$ dar und formulieren Sie die Invariante für die neue Darstellung.

### Aufgabe 14.4

Bei den Einträgen einer priorisierten Schlange handelt es sich um Paare $(p, x)$, die aus einer Priorität $p \in \mathbb{Z}$ und einem Wert $x$ bestehen. Bei der Erweiterung einer priorisierten Schlange um einen Eintrag $(p, x)$ wird dieser so in die Schlange eingetragen, dass alle bisherigen Einträge mit einer Priorität $\geq p$ vor ihm und alle bisherigen Einträge mit einer Priorität $< p$ nach ihm stehen.

a) Geben Sie eine Signatur und eine Implementierung für priorisierte Schlangen an.

b) Geben Sie die Darstellungsinvariante Ihrer Implementierung an.

**Aufgabe 14.5**

a) Sind alle Werte des Typs *int ref list* imperativ?

b) Welchen Wert liefert der Ausdruck *ref* $(3 + 2) =$ *ref* $(7 - 2)$?

c) Erklären Sie, weshalb für eine Referenz $r$ die Aussage $!(ref\ r) = r$ gilt, $ref(!r) = r$ jedoch nicht.

**Aufgabe 14.6**

Ein Generator für eine Folge $x_1, x_2, \ldots$ von Werten eines Typs $t$ ist eine Prozedur $unit \to t$, die beim $n$-ten Aufruf das Folgenglied $x_n$ liefert.

a) Schreiben Sie einen Generator *square* für die Folge $1, 4, 9, \ldots$ der Quadratzahlen.

b) Schreiben Sie eine Prozedur $newSquare\colon unit \to unit \to int$, die bei jedem Aufruf einen neuen Generator für die Folge der Quadratzahlen liefert.

c) Schreiben Sie eine Prozedur $newGenerator\colon (int \to \alpha) \to unit \to \alpha$, die zu einer Prozedur $f$ einen Generator für die Folge $f\ 1, f\ 2, f\ 3, \ldots$ liefert.

d) Schreiben Sie mit Hilfe der Prozedur *newGenerator* einen Generator *cube* für die Folge $1, 8, 27, \ldots$ der Kubikzahlen.

e) Schreiben Sie mit Hilfe der Prozedur *newGenerator* eine Prozedur *newCube*, die bei jedem Aufruf einen neuen Generator für die Folge der Kubikzahlen liefert.

**Aufgabe 14.7**

Schreiben Sie eine Prozedur $copy\colon \alpha\ array \to \alpha\ array$, die zu einer Reihung eine Kopie liefert (d.h. eine neue Reihung, die dieselbe Komponentenfolge wie die gegebene Reihung hat).

**Aufgabe 14.8**

Schreiben Sie eine Prozedur $toList\colon \alpha\ array \to \alpha\ list$, die zu einer Reihung die Liste ihrer Komponenten liefert. Verwenden Sie aus der Struktur *Array* nur *Array.foldl*.

**Aufgabe 14.9**

Der im Buch beschriebene Intervalltest kann weiter verbessert werden, indem man in einer Zelle mitzählt, wie viele Positionen des Logbuchs noch nicht auf *true* gesetzt wurden. Sobald alle Positionen des Logbuchs auf *true* gesetzt sind, kann der Test beendet werden. Schreiben Sie eine Prozedur *interval′*, die einen entsprechend optimierten Intervalltest implementiert.

**Aufgabe 14.10**

Schreiben Sie eine Prozedur $rotate\colon \alpha\ array \to unit$, die die Komponenten einer Reihung um eine Position nach rechts verschiebt und die letzte an die Stelle der ersten rückt. Beispielsweise soll die Komponentenfolge [1,2,3,4] zu [4,1,2,3] werden. Verwenden Sie eine Hilfsprozedur $rotate' : int \to \alpha \to unit$, die für $l$ und $x$ alle Positionen ab $l$ um eins nach rechts verschiebt und die Position $l$ auf $x$ setzt. Die letzte Komponente soll dabei verloren gehen.

**Aufgabe 14.11**

Implementieren Sie *Quicksort* zum Sortieren von Reihungen. Orientieren Sie sich an dem in Aufgabe 15.14 des Buches beschriebenen Vorgehen.

**Aufgabe 14.12**

a) Schreiben Sie eine Prozedur $fac\colon int \to int$, die mit einer Schleife zu $n \in \mathbb{N}$ die $n$-te Fakultät $n!$ bestimmt.

b) Schreiben Sie eine Prozedur $gcd\colon int \to int \to int$, die mit einer Schleife zu zwei positiven Zahlen den größten gemeinsamen Teiler bestimmt.

c) Schreiben Sie eine Prozedur $reverse\colon int\ array \to unit$, die eine Reihung mit Hilfe einer Schleife durch Umordnung ihrer Komponenten reversiert.

**Aufgabe 14.13**

Zeichnen Sie die Kastendarstellungen der Zustände, die die Schlange $q$ gemäß den folgenden Deklarationen durchläuft.

```
open Queue
val q = queue ()
val _ = (insert q 7; insert q 13; remove q)
```

**Aufgabe 14.14**

Warum können Sie in Abbildung 15.4 im Buch in der Prozedur *insert* die Zeile

```
val dummy = ref D
```

nicht durch

```
val dummy = !f
```

ersetzen?