13. Übungsblatt zu Programmierung 1, WS 2008/09

Prof. Dr. Gert Smolka, Mark Kaminski, M.Sc.
www.ps.uni-sb.de/courses/prog-ws08/

Lesen Sie im Buch: Kapitel 14, 15

Aufgabe 14.1 c) Sei die Struktur *ISet* gegeben. Schreiben Sie eine Prozedur

$$equal : ISet.set \to ISet.set \to bool$$

die testet, ob zwei Mengen gleich sind.

Aufgabe 14.2 Warum ist die folgende Deklaration unzulässig?

```
structure S :> sig eqtype t end = struct
    type t = int -> int
end
```

Aufgabe 14.5 (Umgebungen) Unter einer Umgebung wollen wir wie in § 6.4.2 eine Funktion verstehen, die durch Strings realisierte Bezeichner auf bestimmte Werte abbildet. Deklarieren Sie eine Struktur *Env*, die Umgebungen wie folgt implementiert:

type α *env*
exception Unbound
val env : $(string * \alpha)\ list \to \alpha\ env$
val lookup : $\alpha\ env \to string \to \alpha$
val update : $\alpha\ env \to string \to \alpha \to \alpha\ env$

- *env* liefert zu einer Liste von Paaren die entsprechende Umgebung. Wenn die Liste zu einem String mehr als ein Paar enthält, soll nur eines der Paare verwendet werden.
- *lookup* liefert zu einer Umgebung f und einem String s den Wert fs. Falls f auf s nicht definiert ist, wirft *lookup* die Ausnahme *Unbound*.
- *update* liefert zu einer Umgebung f, einem String s und einem Wert x die Umgebung $f[s := x]$.

Deklarieren Sie die Struktur *Env* mit einem Signaturconstraint, der die Signatur direkt angibt (d.h. ohne Rückgriff auf eine Signaturdeklaration).

Aufgabe 14.7 (Puzzle) Für eine abstrakte Datenstruktur stellt sich oft die Frage, ob eine gewünschte Operation mit den Operationen der Struktur programmiert werden kann. Dazu wollen wir eine abstrakte Datenstruktur Puzzle betrachten, die endliche Mengen ganzer Zahlen gemäß der folgenden Signatur bereitstellt:

type set
val set : int list → set
val find : (int → bool) → set → int option

Die Operation *set* soll zu einer Liste die entsprechende Menge liefern, und die Operation *find* soll zu einer Eigenschaft und einer Menge ein Element der Menge liefern, das die Eigenschaft erfüllt.

a) Deklarieren Sie eine Struktur *Puzzle*, die die abstrakte Datenstruktur Puzzle realisiert.
b) Schreiben Sie mithilfe der Operationen von *Puzzle* eine Prozedur *set → int list*, die die Elemente einer Menge als Liste liefert.

Aufgabe 14.11 Mit Vektoren lassen sich Funktionen, deren Definitionsbereich ein endliches Intervall der ganzen Zahlen ist, mit konstanter Laufzeit berechnen. Schreiben Sie eine Prozedur *vectorize*: $(int \to \alpha) \to int \to int \to \alpha$, die zu einer Prozedur p und zwei Zahlen m, n eine Prozedur mit konstanter Laufzeit liefert, die für alle Zahlen $\{x \in \mathbb{Z} \mid m \leq x \leq n\}$ dasselbe Ergebnis wie p liefert und ansonsten die Ausnahme *Subscript* wirft. Nehmen Sie dabei an, dass p auf allen Zahlen zwischen m und n terminiert. Hinweis: Schreiben Sie die Prozedur *vectorize* so, dass sie zunächst den Vektor $[p\,m, \ldots, p\,n]$ bestimmt.

Aufgabe 14.13 Erweitern Sie die effiziente Implementierung von Schlangen um die folgenden Operationen:

a) Eine Operation *cons* : α *queue* $\to \alpha \to \alpha$ *queue*, die zu einer Schlange und einem Wert die Schlange liefert, die sich durch Anfügen des Wertes am Anfang der Schlange ergibt.
b) Eine Operation *length* : α *queue* $\to$ *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 14.14 (Priorisierte Schlangen) 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 für priorisierte Schlangen an.
b) Geben Sie eine Implementierung für priorisierte Schlangen an.
c) Geben Sie die Darstellungsinvariante Ihrer Implementierung an.

Aufgabe 15.1 Sind alle Werte des Typs *int ref list* imperativ?

Aufgabe 15.2 Welchen Wert liefert der Ausdruck $ref(3 + 2) = ref(7 - 2)$?

Aufgabe 15.4 (Generatoren) 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 $newSquare : unit \to unit \to int$, die bei jedem Aufruf einen neuen Generator für die Folge der Quadratzahlen liefert.
c) Schreiben Sie eine Prozedur $newGenerator : (int \to \alpha) \to unit \to \alpha$, die zu einer Prozedur f einen Generator für die Folge $f\,1, f\,2, f\,3, \ldots$ liefert.
d) Schreiben Sie mithilfe der Prozedur *newGenerator* einen Generator *cube* für die Folge $1, 8, 27, \ldots$ der Kubikzahlen.
e) Schreiben Sie mithilfe der Prozedur *newGenerator* eine Prozedur *newCube*, die bei jedem Aufruf einen neuen Generator für die Folge der Kubikzahlen liefert.

Aufgabe 15.5 (Generator für Fakultäten) Schreiben Sie einen Generator *fac* für die Folge $1!, 2!, 3!, \ldots$ der Fakultäten. Verwenden Sie keine Rekursion. Die Laufzeit für einen Aufruf von *fac* soll konstant sein.

Aufgabe 15.6 (Zähler mit Rücksetzen) Vervollständigen Sie die Deklaration

```
val (count, inc, reset) =
```

so, dass sie einen eingekapselten Zähler mit dem Anfangswert 0 und drei Prozeduren wie folgt liefert:

- *count* : *unit* → *int* liefert den Wert des Zählers.
- *inc* : *unit* → *unit* erhöht den Wert des Zählers um 1.
- *reset* : *unit* → *unit* setzt den Zähler auf 0 zurück.