



## 12. Übungsblatt zu Programmierung 1, WS 2010/11

Prof. Dr. Gert Smolka, Christian Doczkal, M.Sc.

[www.ps.uni-sb.de/courses/prog-ws10/](http://www.ps.uni-sb.de/courses/prog-ws10/)

---

Lesen Sie im Buch: Kapitel 14

---

**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  $\alpha$  env
exception Unbound
val env : (string *  $\alpha$ ) list →  $\alpha$  env
val lookup :  $\alpha$  env → string →  $\alpha$ 
val update :  $\alpha$  env → string →  $\alpha$  →  $\alpha$  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 *f.s*. 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* →  $\alpha$ ) → *int* → *int* → *int* →  $\alpha$ , 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  $[p\ m, \dots, p\ n]$  bestimmt.

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

- Eine Operation *cons*:  $\alpha\ queue \rightarrow \alpha \rightarrow \alpha\ queue$ , die zu einer Schlange und einem Wert die Schlange liefert, die sich durch Anfügen des Wertes am Anfang der Schlange ergibt.
- Eine Operation *length*:  $\alpha\ queue \rightarrow int$ , 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.

- Geben Sie eine Signatur für priorisierte Schlangen an.
- Geben Sie eine Implementierung für priorisierte Schlangen an.
- 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  $x_1, x_2, \dots$  von Werten eines Typs  $t$  ist eine Prozedur *unit*  $\rightarrow t$ , die beim  $n$ -ten Aufruf das Folgenglied  $x_n$  liefert.

- Schreiben Sie einen Generator *square* für die Folge 1, 4, 9, ... der Quadratzahlen.
- Schreiben Sie eine Prozedur *newSquare*:  $unit \rightarrow unit \rightarrow int$ , die bei jedem Aufruf einen neuen Generator für die Folge der Quadratzahlen liefert.
- Schreiben Sie eine Prozedur *newGenerator*:  $(int \rightarrow \alpha) \rightarrow unit \rightarrow \alpha$ , die zu einer Prozedur  $f$  einen Generator für die Folge  $f\ 1, f\ 2, f\ 3, \dots$  liefert.
- Schreiben Sie mithilfe der Prozedur *newGenerator* einen Generator *cube* für die Folge 1, 8, 27, ... der Kubikzahlen.
- 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!, ... der Fakultäten. Verwenden Sie keine Rekursion. Die Laufzeit für einen Aufruf von *fac* soll konstant sein.