Universität der Bundeswehr
Fakultät für Informatik – Institut 2
Prof. Dr. Wolfgang Hommel
Michael Grabatin, Michael Steinke

WT 2020
Übungsblatt 8

Einführung in die Informatik II

06. und 09.03.2020

1 Abstrakte Datentypen (ADT)

Es ist folgende Formalisierung für Zeichenketten gegeben:

- Relevante Mengen:
  $\mathcal{A}$: Menge aller Zeichen (das Alphabet)
  $\mathcal{S}$: Menge aller Zeichenketten
- Signatur:

ϵ	$\mid \rightarrow \mathcal{S}$
$append$	$\mid \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{S}$
$concat$	$\mid \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{S}$
$reverse$	$\mid \mathcal{S} \rightarrow \mathcal{S}$

- Axiome:

$(S1)$	$concat(\epsilon, s)$	$= s$
$(S2)$	$concat(append(a, s_1), s_2)$	$= append(a, concat(s_1, s_2))$
$(S3)$	$reverse(\epsilon)$	$= \epsilon$

a) Bestimmen Sie Konstruktoren und Nicht-Konstruktoren.

> Da sich alle möglichen Zeichenketten aus ϵ und $append$ erstellen lassen, sind diese die Konstruktoren. Die Operationen $concat$ und $reverse$ sind daher die Nicht-Konstruktoren.

b) Geben Sie das Axiom $(S2)$ vollquantifiziert an.

> Mit den Quantoren lässt sich das Axiom $(S2)$ wie folgt notieren:
> $\forall a \in \mathcal{A} : \forall s_1, s_2 \in \mathcal{S} : concat(append(a, s_1), s_2) = append(a, concat(s_1, s_2))$

c) Offensichtlich ist die algebraische Definition nicht vollständig. Vervollständigen Sie diese entsprechend!

> Da es zwei Konstruktoren und zwei Nicht-Konstruktoren gibt, fehlt ein Axiom $(S4)$:
> $(S4) \quad reverse(append(a, s)) \quad = concat(reverse(s), append(a, \epsilon))$

d) Für Zeichenketten sollte im Allgemeinen auch folgende Regel gültig sein:
$(R1) \quad concat(s, \epsilon) \quad = s$
Ist diese als weiteres Axiom notwendig? Begründen Sie!

> Die Regel $concat(s, \epsilon) = s$ ist aus den bestehenden Axiomen herleitbar und daher nicht als weiteres Axiom notwendig. Zum Beweis dient eine *strukturelle Induktion* bzgl. des Aufbaus der Zeichenkette $s \in \mathcal{S}$.
>
> - Fall $s = \epsilon$:
> $concat(\epsilon, \epsilon) =_{(S1)} \epsilon$
> - Fall $s = append(a, s')$ (Induktionsvoraussetzung (IV): Es gilt $(R1)$ für $s = s'$):

$$concat(append(a, s'), \epsilon) =_{(S2)} append(a, concat(s', \epsilon)) =_{(IV)} append(a, s') = s \text{ (q.e.d)}$$

e) Die Definition soll nachträglich um eine Operation $charAt$ erweitert werden, mit der einzelne Zeichen an einem Index (mit 0 beginnend) ermittelt werden können. Führen Sie die entsprechenden Ergänzungen durch.

Zunächst ist eine Menge $\mathcal{N}$ der natürlichen Zahl als „relevante Menge“ aufzunehmen. Die Operation $charAt$ hat dann sinnvollerweise folgende Signatur:

$charAt \quad |\mathcal{S} - \{\epsilon\} \times \mathcal{N} \to \mathcal{A}$

Die Funktionen ist nur partiell definiert, da es nicht möglich ist, von einer leeren Zeichenkette oder einer Zeichenkette der Länge n ein Zeichen mit dem Index $i \geq n$ zu ermitteln. Daher wird das folgende Axiom ergänzt:

$(S5) \quad charAt(append(a, s), n) \quad = if\ n == 0\ then\ a\ else\ charAt(s, n-1)$

Zur besseren Lesbarkeit kann das Axiom auch in zwei Teile a und b zerlegt werden.

$(S5a) \quad charAt(append(a, s), 0) \quad = a$
$(S5b) \quad charAt(append(a, s), n+1) \quad = charAt(s, n)$

2 Radixsort

In der vorhergehenden Übung wurde das Sortierverfahren *Radixsort* für einen festen Typ implementiert. In dieser Aufgabe soll diese Lösung so erweitert werden, dass sie für generische Typ funktioniert.

Jeder Schlüssel ist eine Folge von m Stellen.

Zum Sortieren verwendet *Radixsort* k Fächer, je ein Fach pro möglichem Stellenwert.

Hinweis: Die Position pos wird (Scala-üblich) von 0 beginnend gezählt. Mit $pos = 2$ wird daher auf die Einerstelle einer dreistelligen Zahl (also bei $m = 3$) zugegriffen!

a) Zur Realisierung der Kellerfächer soll der folgende nicht generische Typ `Stack` in einen generischen Typ überführt werden:

```
1 //Keller
2 class StackElem(var value : Int, var next : StackElem = null)
3 class Stack(var s : StackElem = null)
4
5 //leeren Stack erstellen
6 def emptyStack : Stack = new Stack
7
8 //Pruefen ob Stack leer
9 def isEmpty(stack : Stack) : Boolean = stack.s == null
```

Der generische Typ sieht wie folgt aus:

```
1 //Keller
2 class StackElem[T](var value : T, var next : StackElem[T] = null)
3 class Stack[T](var s : StackElem[T] = null)
4
5 //leeren Stack erstellen
6 def emptyStack[T] : Stack[T] = new Stack[T]
7
8 //Pruefen ob Stack leer
```

```
9 def isEmpty[T](stack : Stack[T]) : Boolean = stack.s == null
```

b) Modifizieren Sie ebenfalls die Implementierung der folgenden Prozeduren, damit diese auf dem generischen Stack funktionieren.

```
1  //Element hinzufuegen
2  def push(stack : Stack, value : Int) : Unit = {
3    stack.s = new StackElem(value, stack.s)
4  }
5
6  //Oberstes Element entfernen
7  def pop(stack : Stack) : Unit = {
8    if (stack.s == null) throw new
9      IllegalArgumentException("Stack is empty!")
10   else stack.s = stack.s.next
11 }
12
13 //Oberstes Element vom Stack liefern
14 def top(stack : Stack) : Int = {
15   if (isEmpty(stack)) throw new
16     IllegalArgumentException("Stack is empty!")
17   else return stack.s.value
18 }
```

Die Implementierungen sehen wie folgt aus:

```
1  //Element hinzufuegen
2  def push[T](stack : Stack[T], value : T) : Unit = {
3    stack.s = new StackElem(value, stack.s)
4  }
5
6  //Oberstes Element entfernen
7  def pop[T](stack : Stack[T]) : Unit = {
8    if (stack.s == null) throw new
9      IllegalArgumentException("Stack is empty!")
10   else stack.s = stack.s.next
11 }
12
13 //Oberstes Element vom Stack liefern
14 def top[T](stack : Stack[T]) : T = {
15   if (isEmpty(stack)) throw new
16     IllegalArgumentException("Stack is empty!")
17   else return stack.s.value
18 }
```

c) In der nicht generischen Version wurde eine Prozedur `getNumPos`, welche bei gegebenen m maximalen Stellen pro Schlüssel, an der Stelle pos, den Wert von num zurückgibt, verwendet. Zum Beispiel wurde für $m = 3, pos = 2, num = 216$ das Ergebnis 6 zurückgegeben. Für einen beliebigen Typ `T` funktioniert diese Funktion nicht mehr. Sie benötigen jetzt eine Funktion, die Ihnen zu einem belibigen Datentyp eine für die Sortierung in die Keller geeignete Darstellung zurück gibt. Definieren Sie nun die Prozedur `radixsort`, welche einen Array `keys : Array[T]` nach dem *Radixsort*-Verfahren sortiert und dabei eine als (curri-sierten) Parameter übergebene Funktion `getKeyPos: (Int, T, Int)=>Int` zur Bestimmung der Kellerfächer verwendet. Die bekannte Funktion `getNumPos` soll zum Beispiel für die Sortierung von Integern als `getKeyPos` übergeben werden können.

Die Funktion kann wie folgt implementiert werden:

```
1  def radixsort[T](getKeyPos: (Int, T, Int) => Int)(k : Int, m :
       Int, keys : Array[T]) : Unit = {
2      // k Kellerfaecher anlegen
3      val stacks = Array.fill(k)(emptyStack[T])
4
5      // alle Positionen der Schluessel von hinten durchlaufen
6      for (pos <- m - 1 to 0 by -1) {
7          // 1. Phase: Verteilen des Inhalts von keys auf die
8          // Kellerfaecher
9          for (i <- 0 to keys.length - 1) {
10             val key = keys(i)
11             push(stacks(getKeyPos(m, key, pos)), key)
12         }
13
14         var idx = keys.length - 1
15
16         // 2.Phase: Leeren der Kellerfaecher und schreiben
17         // nach keys
18         for (j <- k - 1 to 0 by -1) {
19             val stack = stacks(j)
20             while (!isEmpty(stack)) {
21                 keys(idx) = top(stack)
22                 pop(stack)
23                 idx -= 1
24             }
25         }
26     }
27 }
```

Beispielanwendung:

```
1  def getNumPos(m : Int, num : Int, pos : Int) : Int = {
2    var n = num
3    for (i <- 1 to (m - pos) - 1) n /= 10
4
5    return n % 10
6  }
7
8  var a1 = Array(3, 432, 453, 3, 32, 431, 24, 3)
9
10 radixsort(getNumPos)(10, 3, a1)
11 a1 //> res0: Array[Int] = Array(3, 3, 3, 24, 32, 431, 432, 453)
```

d) Welche Werte müssen für k und m gewählt werden, wenn das Eingabearray für den Radixsort aus Zeichenketten mit Buchstaben besteht und wie muss eine Funktion `getKeyPos` für Strings aussehen? Implementieren Sie diese Funktion und testen Sie Ihre Funktion an folgendem Beispiel: `Array("alice", "bob", "eve", "trent")`

m muss so gewählt werden, dass alle Stellen des längsten Strings betrachtet werden können. In diesem Fall $m = 5$. k muss so gewählt werden, dass alle Rückgabewerte von `.getNumericValue` einsortiert werden können. Angenommen die Eingabe kann nur Buchstaben enthalten, dann ist die größte Rückgabe von `'z'.getNumericValue //> 35`.

```
1  def getStrPos(m: Int, s: String, pos: Int) : Int = {
2    if(s.length <= 0 || pos >= s.length){
3        return 0
4    } else {
5        // Zugriff in diesem Fall vom Anfang des Strings da "a" > "
         bb"
6        return s(pos).getNumericValue
7    }
8  }
9
10 var a2 = Array("trent", "bob", "alice", "eve")
11
12 radixsort(getStrPos)(36, 5, a2)
13 a2 //> res1: Array[String] = Array(alice, bob, eve, trent)
```