12. Aufgabenblatt zur Vorlesung

Informatik A

WiSe 2015/16

Wolfgang Mulzer

Abgabe am 26. Januar 2016 bis 12 Uhr in die Tutorenfächer
Dies ist das vorletzte Aufgabenblatt.

Aufgabe 1 Bottom-Up Merge-Sort *10 Punkte*

Bottom-Up Merge Sort ist eine Variante von Merge-Sort, welche die Mischoperation "von unten" durchführt. Es funktioniert folgendermaßen:

- Wandle die Eingabeliste `xs::[t]` um in eine Liste von Listen `xss::[[t]]`, welche für jedes Element `x` aus `xs` genau eine Liste der Form `[x]` enthält.
- Solange `xss` mehr als eine Liste enthält, gehe `xss` durch und mische benachbarte Listen mit Hilfe von `merge`.
- Gib die verbleibende Liste in `xss` zurück.

Bearbeiten Sie die folgenden Aufgaben:

(a) Implementieren Sie Bottom-Up Merge Sort in Haskell.

(b) Schätzen Sie ab, wie viele Vergleiche Bottom-Up Merge Sort benötigt.

Aufgabe 2 Induktion *10 Punkte*

(a) Beweisen Sie mittels struktureller Induktion, dass für alle endlichen Listen (von Listen) `xss` und alle Funktionen `f` gilt:

```
concat (map (map f) xss) = map f (concat xss).
```

Zur Erinnerung:

```
(++)::[a] -> [a] -> [a]
[] ++ ys    = ys                    -- ++.1
(x:xs) ++ ys = x:(xs ++ ys)         -- ++.2

concat::[[a]] -> [a]
concat []       = []                --concat.1
concat (xs:xss) = xs ++ concat xss  --concat.2

map::(a -> b) -> [a] -> [b]
map g []     = []                   --map.1
map g (y:ys) = g y : map g ys       --map.2
```

Hinweis: Beweisen Sie zunächst folgendes Lemma: Für alle endlichen Listen `xs,ys::[a]` gilt:

```
map f (xs ++ ys) = map f xs ++ map f ys.
```

(b) Zeigen Sie mit vollständiger Induktion, dass für alle natürlichen Zahlen `n` gilt:

```
iter n id = id.
```

Dabei ist die Iterationsfunktion defininiert durch

```
iter::Int -> (a -> a) -> (a -> a)
iter n f
| n > 0     = f . iter (n - 1) f --iter.1
| otherwise = id                 --iter.2
```

und `id::a -> a` ist die polymorphe Identitätsfunktion, durch

```
id x = x  --id.1
```

gegeben. Der `.`-Operator beschreibt die Funktionskomposition: für Funktionen `g::a -> b` und `f::b -> c` hat die Funktionskomposition `f . g` die Signatur `f . g::a -> c` und

```
(f . g) x = f (g x)  -- .1
```

Aufgabe 3 Codierung *10 Punkte*

Der *Kreuzsicherungscode* wandelt ein 0-1-Wort der Länge n^2 in ein 0-1-Wort der Länge $n^2 + 2n$ um, indem er die Eingabe als $n \times n$-Matrix auffasst und für jede der n Zeilen und n Spalten jeweils ein Paritätsbit anhängt.

(a) Zeigen Sie: Der Kreuzsicherungscode ist 1-fehlerkorrigierend.

(b) Schreiben Sie eine Kodierungsfunktion `encode::Int -> [Int] -> [Int]` für den Kreuzsicherungscode. Dabei soll der erste Parameter die Dimension n der Code-Matrix angeben.

Definieren Sie dazu zunächst geeignete Teilaufgaben. Achten Sie im Sinne der Wiederverwendbarkeit darauf, Ihre Funktionen so allgemein wie möglich zu halten.

(c) (freiwillig, 5 Zusatzpunkte) Geben Sie eine Implementierung einer funktionierenden 1-fehlerkorrigierenden Decodierfunktion für den Kreuzsicherungscode.

Geben Sie immer alle Signaturen an und machen Sie geeignete Testläufe. Versehen Sie Ihr Skript mit geeigneten Kommentaren.