Übungen zu Softwareentwicklung III, Funktionale Programmierung

Blatt 7, Woche 8

Leonie Dreschler-Fischer

WS 2012/2013

Ausgabe: Freitag, 7.12.2011,

Abgabe der Lösungen: bis Montag, 17.12.2012, 12:00 Uhr per email bei den Übungsgruppenleitern.

Ziel: Rekursion: Die Aufgaben auf diesem Zettel dienen dazu, sich mit dem Entwurf von endrekursiven Funktionen und einfachen Funktionen höherer Ordnung vertraut zu machen.

Bearbeitungsdauer: Die Bearbeitung sollte insgesamt nicht länger als 5 Stunden dauern.

Homepage:
http://kogs-www.informatik.uni-hamburg.de/~dreschle/teaching/Uebungen_Se_III/Uebungen_Se_III.html

Bitte denken Sie daran, auf den von Ihnen eingereichten Lösungsvorschlägen *Ihren Namen und die Matrikelnummer, den Namen der Übungsgruppenleiterin / des Übungsgruppenleiters* und *Wochentag und Uhrzeit der Übungsgruppe* anzugeben, damit wir ihre Ausarbeitungen eindeutig zuordnen können.

1 Abbilden

Bearbeitungszeit 1 Std., 9 Pnkt.

Definieren Sie eine Funktion *produkt*, die eine Liste von Zahlen n sowie einen Faktor f als Argument nimmt und eine neue Liste errechnet, die anstelle von n jeweils das Produkt $n \cdot f$ enthält. Beispiel:

> (produkt '(1 4 2) 3) ⟶ '(3 12 6)

Programmieren Sie diese Funktion in drei Varianten:

- als allgemein rekursive Funktion,
- als endrekursive Funktion,
- mittels geeigneter Funktionen höherer Ordnung.

2 Ampelsteuerung

Bearbeitungszeit 4 Std., ingesamt 22 Punkte

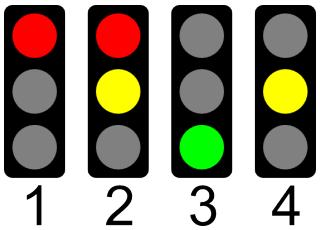
In dieser Aufgabe sollen Sie Verkehrssteuerung durch Ampeln modellieren. Die Verkehrsampeln, die für Autos den Verkehrsfluss steuern bestehen in Deutschland üblicherweise aus 3 farbigen Leuchten: Rot, Gelb und Grün. Bei dieser Aufgabe geht es ausschließlich um die Modellierung der Straßenverkehrs, Fußgängerampeln sollen nicht mit modelliert werden!

Verwenden Sie nach Möglichkeit Funktionen höherer Ordnung.

2.1 Zustände

4 Pnkt.

Überlegen Sie sich eine Kodierung der Zustands einer Ampel unter der Annahme, dass auch mehr als eine Leuchte auf einmal eingeschaltet sein darf. Erstellen Sie anschließend eine Liste, die alle möglichen Zustände in sequenzieller Abfolge enthält. Orientieren Sie sich hierbei an folgender Grafik:



2.2 Zustandsexpansion

5 Pnkt.

Bei einer realen Ampelschaltung sind die einzelnen Phasen oft unterschiedlich lang. Legen Sie deshalb eine weitere Liste an, die die Dauer der Zustände kodiert (z.B. Rot: 5s, Gelb-Rot 1s, Grün 5s, Gelb 2s).

Schreiben Sie eine Funktion *expandiere*, die zwei Listen als Argumente erhält, und jedes Element der ersten Liste um die entsprechende Anzahl in der zweiten Liste expandiert. Beispiel:

```
> (expandiere '(a b c) '(2 1 3)) ⟶ '(a a b c c c)
```

Tipp: Scheuen Sie sich nicht davor, Funktionen höherer Ordnung zu verwenden und schauen Sie sich die Funktion make-list an. Diese könnte hier sehr hilfreich sein! Beispiel: (make-list 2 a) –> '(a a)

Wenden Sie *expandiere* an, um die Zustände ihrer Ampel um die vorgegebene Dauer zu expandieren.

2.3 Simulation 1

6 Pnkt.

Schreiben Sie unter Zuhilfenahme des image-Paketes (require 2htdp/image) eine Funktion, die eine Ampel grafisch darstellt. Benutzen Sie hierfür ein Rechteck der Größe 30x70 Pixel für den Rahmen und Kreise vom Radius 10 Pixel für die einzelnen Lampen.

Schreiben Sie außerdem eine Funktion (zeige-ampel t), die eine Ampel zu einem Zeitpunkt t anzeigt. Beachten Sie, dass t auch größer als die Länge der Zustandsliste aus Aufgabe 2.2 sein kann. In diesem Fall soll erneut von vorne begonnen werden.

Mit dem Animate-Framework (require 2htdp/universe) sollten Sie nun in der Lage sein, die Ampel (mit 28 Ticks pro Sekunde) zu simulieren:

```
(animate zeige-ampel)
```

2.4 Kreuzungen

11 Pnkt.

Bei Kreuzungen gibt es üblicherweise Ampeln für alle Richtungen, aus denen Autos auf diese zufahren können. Überlegen Sie sich, wie die Ampeln des kreuzenden Verkehrs geschaltet werden müssen, damit es zu keinen Unfällen kommt. Beobachten Sie dazu ruhig eine "reale Ampel" an einer Kreuzung Ihrer Wahl. Beachten Sie insbesondere, dass es auch Phasen gibt, in denen alle Ampeln rot sind.

Geben Sie ein Zustands-Übergangsdiagramm an, und schreiben Sie eine Funktion, die den Zustand der kreuzenden Ampel in Bezug auf die Ampel des durchgehenden Verkehrs beschreibt.

Tipp: Eventuell müssen Sie die Liste aus Aufgabe 2.1 erweitern, um das gewünschte Verhalten zu erreichen.

2.5 Zusatzaufgabe: Simulation 2

5 Zusatzpnkt.

Erstellen Sie mit Hilfe des Animate-Frameworks (require 2htdp/universe) eine Simulation, die die Ampelschaltung (von 4 Ampeln) an einer Kreuzung simuliert. Als Basis-Szene können Sie z.B. folgende verwenden:

```
(define crossing-scene
  (let ((mark (make-pen "white" 5 "long-dash" "butt" "miter")))
    (scene+line
      (scene+line
        (overlay (rectangle 600 90 "solid" "gray")
                 (rectangle 90 600 "solid" "gray")
                 (rectangle 600 600 "solid" "darkgreen"))
        0 300 600 300 mark)
      300 0 300 600 mark)))
```

Am besten schreiben Sie analog zu Aufgabenteil 2.3 eine zweite Funktion, die die zweite Ampel in Abhängigkeit von t anzeigt. Da beide Funktionen Bilder zurückliefern, müssen Sie diese nur noch über die Szene legen. Wenn Sie dies in einer Funktion (ampel-kreuzung t) kapseln, können Sie diese mit folgendem Befehl animieren:

```
(animate ampel-kreuzung)
```

Erreichbare Punkte: 35

Erreichbare Zusatzunkte: 5