Übungen zu Softwareentwicklung III, Funktionale Programmierung

Blatt 8, Woche TBA

Funktionen höherer Ordnung

Leonie Dreschler-Fischer

WS 2016/2017

Ausgabe: Freitag, TBA,

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

Ziel: Die Aufgaben auf diesem Zettel dienen dazu, sich mit dem Entwurf von Funktionen höherer Ordnung und Funktionaler Abschlüsse (closures) vertraut zu machen.

Zur Lösung der Aufgaben sollten Sie auf die in der Vorlesung behandelten Funktionen `curry, curryr, compose, filter, foldl` usw. zurückgreifen. Weitere Funktionen höherer Ordnung finden Sie im Modul "tools-module". Sie finden dieses Modul in der se3-bib und in STINE bei den Materialien zur Vorlesung.

Bearbeitungsdauer: Die Bearbeitung sollte insgesamt nicht länger als 5 1/2 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 Funktionen höherer Ordnung und Closures

Bearbeitungszeit 1 Std., 10 Pnkt.

1. Wann ist eine Racket-Funktion eine Funktion höherer Ordnung?
2. Welche der folgenden Funktionen sind Funktionen höherer Ordnung und warum?

   (a) foldr

   (b)
   ```
   (define (plus-oder-minus x)
           (if (< x 0)
               'Plus
               'Minus))
   ```

   (c)
   ```
   (define (masala f arg1)
           (lambda (arg2)
                   (f arg1 arg2)))
   ```

   (d)
   ```
   (define (flip f)
     (lambda (x y) (f y x)))
   ```

3. Das Resultat der Funktion masala (Aufgabe 1.2.c) ist eine Closure. Erklären Sie am Beispiel des Aufrufs

   ```
   ((masala / 1) 3)
   ```

   welche Umgebungen und welche Bindungen der vorkommenden Variablen bei der Auswertung des Ausdrucks eine Rolle spielen.

4. Zu welchem Wert evaluieren die folgenden Ausdrücke? *Begründen* Sie Ihre Antwort.

   ```
   (foldl  (curry * 3) 1 '(1 2 3))

   (map (flip cons) '(1 2 3) '(3 2 1))

   (filter list? '((a b ) () 1 (())))

   (map (compose (curryr / 1.8) (curry - 32))
     '(9941 212 32 -459.67))
   ```

2 Einfache funktionale Ausdrücke höherer Ordnung

(Bearbeitungszeit 30 Minuten.)

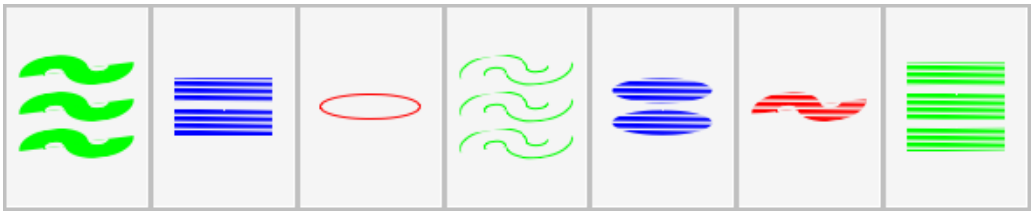
Gegeben sei eine Liste xs von ganzen Zahlen: 6 Pnkt.

1. Geben Sie einen Ausdruck an, der die *Liste der Absolutbeträge aller Zahlen* der Liste xs berechnet.
2. Geben Sie einen Ausdruck an, der die Teilliste aller glatt durch 13 teilbaren Zahlen von xs konstruiert.
3. Geben Sie einen Ausdruck an, der *die Summe der geraden Zahlen größer 3 in xs* ermittelt.
4. Zusatzaufgabe: Geben Sie einen Ausdruck an, welcher eine Liste anhand eines Prädikats (z.B. odd?) in zwei Teillisten aufspaltet und zurückgibt. 3 Zusatzpnkt.

Verwenden Sie *Funktionen höherer Ordnung.*

3 Spieltheorie: Das Kartenspiel SET!

(Bearbeitungszeit ohne Zusatzaufgabe 4 Std.), 24 Punkte + 10 Zusatzpunkte



Bei dem Kartenspiel SET! handelt es sich um ein Kartenspiel, das 1974 von Marsha Jean Falco erfunden wurde. Das Spiel besteht aus 81 Karten, die sich in vier Eigenschaften unterscheiden: Die Form der Symbole auf einer Karte, die Anzahl der Symbole, die Farbe der Symbole und die Schattierung (Füllmuster) der Symbole. Jede dieser Eigenschaften ist in drei Ausprägungen vorhanden; damit ergeben sich die $81 = 3 * 3 * 3 * 3$ verschiedenen Karten.

Die Eigenschaften sind:

Form	Oval	Rechteck	Welle
Farbe	rot	blau	grün
Anzahl	ein	zwei	drei
Füllung	Linie	Schraffur	Fläche

Man kann verschiedene Varianten dieses Spiels spielen, die für diesen Aufgabenzettel aber keine Rolle spielen. Sie alle drehen sich um das so genannte SET.
Ein Set besteht aus drei Karten, die für jede Eigenschaft die Bedingung erfüllen müssen, dass alle Karten in dieser Eigenschaft übereinstimmen oder dass keine zwei der Karten in dieser Eigenschaft übereinstimmen. Das heißt:

- Alle Karten haben dieselbe Anzahl an Symbolen, oder jede hat eine andere Anzahl.
- Alle Karten zeigen dasselbe Symbol, oder jede zeigt ein anderes Symbol.
- Die Symbole einer Karte haben dieselbe Farbe wie die der anderen Karten, oder jede Karte hat eine andere Farbe.
- Die Symbole einer Karte haben dieselbe Schattierung wie die der anderen Karten, oder jede Karte hat eine andere Schattierung.

Dabei wird jede Eigenschaft unabhängig von den anderen betrachtet.

Zieht man zwei beliebige Karten aus dem Spiel, dann gibt es genau eine weitere Karte, die beide zu einem SET ergänzt. Drei Karten, die ein Set ergeben, könnten zum Beispiel sein:

- 3 rote Ovale, gefüllte Fläche
- 2 rote Rechtecke, Fläche nicht gefüllt
- 1 rote Welle, schraffiert

Diese Karten bilden ein SET, da sie eine unterschiedliche Anzahl, gleiche Farbe, unterschiedliche Symbole und unterschiedliche Schattierung enthalten.

Das Spiel läuft wie folgt ab: Es werden 12 Karten vom Stapel offen ausgelegt. Der oder die Spieler müssen nun drei Karten heraussuchen, die ein SET bilden. Wenn ein Spieler ein SET gefunden hat, zeigt er auf die Karten. Die Mitspieler prüfen die Eigenschaften. Bilden die drei Karten kein SET, muss der Spieler einmal aussetzen. Wenn das SET richtig ist, kann er die Karten wegnehmen, und es werden drei neue vom Kartenstapel aufgedeckt. Findet man unter den 12 Karten kein SET, werden drei weitere Karten hinzugefügt. Gewinner ist derjenige, der die meisten Karten gesammelt hat.

Weitere (auch bebilderte) Beispiele und Informationen zu dem Spiel können unter: https://de.wikipedia.org/wiki/Set_(Spiel) abgerufen werden.

Aufgabe:
Schreiben Sie ein Racket Programm, welches erkennen kann, ob sich einem einem Spiel (bestehend aus zwölf zufällig vom Stapel gezogenen Karten) ein oder mehrere SETs befinden. Lösen Sie dazu die folgenden Teilaufgaben und verwenden Sie *Funktionen höherer Ordnung* oder nichtdeterministische Funktionen, wann immer es Ihnen sinnvoll erscheint:

1. Entwerfen und implementieren Sie die Repräsentation der verfügbaren Ausprägungen einer Spielkarte, und bedenken Sie dabei bereits die folgenden Schritte (z.B. dass sich Funktionen höherer Ordnung leichter auf Listen als auf andere Strukturen anwenden lassen). Fahren Sie anschließend mit der Repräsentation einer Spielkarte fort. — 4 Pnkt.

2. Erzeugen Sie die Menge (Liste) aller 81 Spielkarten, aus denen das Kartenspiel besteht. Achten Sie dabei auf die Wiederverwendung von definierten Symbolen, insbesondere der Ausprägungen, die Sie in der vorigen Aufgabe implementiert haben. Schreiben sie eine Funktion, die die Karten grafisch als Bild darstellt. Hierzu können Sie die Funktion show−set−card aus dem Modul „setkarten−module“ verwenden. Nach Installation der se3-bib können Sie es mit (**require** se3−bib/setkarten−module) einbinden. — 10 Pnkt.

   Die Signatur von show−set−card ist:

```
(define (show-set-card n the-pattern the-mode the-color)
  ; n: 1, 2, or 3
  ; the-pattern: 'waves, 'oval, 'rectangle
  ; the-mode: 'outline, 'solid, 'hatched
  ; the-color: 'red, 'green, 'blue
```

3. Schreiben Sie eine Funktion, die für drei Spielkarten bestimmt, ob es sich bei diesen um ein SET handelt oder nicht. Testen sie diese Funktion mit manuell ausgewählten Karten des Spiels, die Sie im vorigen Schritt erzeugt haben. Der Funktionsaufruf könnte zum Beispiel so aussehen: 10 Pnkt.

```
(is-a-set? '((2 red oval hatched)
             (2 red rectangle hatched)
             (2 red wave hatched))) ⟶ #t
(is-a-set? '((2 red rectangle outline)
             (2 green rectangle outline)
             (1 green rectangle solid))) ⟶ #f
```

4. Zusatzaufgabe: 7 Zusatzpnkt.
   Ziehen Sie aus den 81 Spielkarten zufällig zwölf Karten, wie dies auch im realen Spiel passiert. Zeichnen sie ein Bild der zwölf Karten. Finden sie alle möglichen SETs, die in den aktuellen zwölf Karten vorkommen und geben Sie diese aus.

Erreichbare Punkte: 40

Erreichbare Zusatzunkte: 10