Funktionale und objektorientierte Programmierkonzepte

Florian Kadner und Lukas Röhrig
(Gesamtleitung: Prof. Karsten Weihe)

Wintersemester 18/19
v1.4

Übungsblatt 2

Themen: Funktionen höherer Ordnung, Lambda-Ausdrücke

Relevante Folien: Funktionale Abstraktion

Abgabe der Hausübung: 09.11.2018 bis 23:55 Uhr

V Vorbereitende Übungen

V1 Theoriefragen ☆ ☆ ☆

Erklären Sie kurz in eigenen Worten, die folgenden Konzepte:

1. Was sind Funktionen höherer Ordnung? Wo liegen ihre Vorteile?
2. Was ist ein Lambda-Ausdruck?
3. Wieso sollte man Abstraktion beim Programmieren verwenden?

V2 Ausdrücke mit Funktionen höherer Ordnung ★ ☆ ☆

In der Vorlesung haben Sie die Funktionen `my-map`, `my-filter` und `my-fold` kennengelernt und diese selbst implementiert. Alle drei Funktionen gibt es mit identischer Funktionalität bereits vordefiniert in DrRacket und haben die Namen `map`, `filter` und `foldr`.

Was liefern die folgenden Ausdrücke zurück? Arbeiten Sie hier ausschließlich mit Stift und Papier und verwenden Sie DrRacket erst hinterher, nur zur Überprüfung Ihrer Ergebnisse.

1. `(map + (list 1 2 3)(list 4 5 6))`
2. `(filter positive? (list 1 -2 3 4 -5))`
3. `(foldr + 0 (list 5 -9 3 2 5 6))`
4. `(filter string? (list 1 2 "3" 4 "abc"))`
5. `(first (map list (list "x" "y" "z")))`
6. `(map list (list "a" "b" "c")(list 1 2 3)(list true false true))`
7. `(foldr cons (list -10 -1)(list 1 10 100 1000))`
8. `(foldr list (list -10 -1)(list 1 10 100 1000))`

V3 Funktionen höherer Ordnung verwenden ★ ☆ ☆

Definieren Sie die folgenden Funktionen. Außerhalb der Funktionen `map`, `filter` und `foldr` darf keine Rekursion verwendet werden.

- Eine Funktion `zip`, die aus zwei gleich langen Listen eine Liste von geordneten Paaren macht. Beispiel:
  `(zip (list "a" "b")(list 1 2))` → `(list (list "a" 1)(list "b" 2))`
- Eine Funktion `vec-mult`, die zwei gleich lange Listen von Zahlen erhält und das Skalarprodukt, also die Summe der paarweisen Produkte berechnet. Beispiel:
  `(vec-mult (list 1 2 3)(list 4 5 6))` → `(+ (* 1 4)(*2 5)(* 3 6))` → `32`

V4 Lambda-Ausdrücke ★ ☆ ☆

```
1 ;;
2 (define (z x)
3 ;;
4         (lambda (y) (* x y)))
```

Ergänzen Sie den Vertrag, sowohl für die Funktion `z`, als auch für den Lambda-Ausdruck. Was liefert `((z 3) 4)` zurück?

V5 Foo Reloaded I ★ ★ ☆

Erinnern Sie sich noch an die Funktion `foo` aus Aufgabe V7 vom letzten Übungsblatt? Zur Erinnerung: Gegeben ist ein Struct-Typ `abc` mit zwei Feldern `a` und `b`. Die Funktion `foo` bekommt einen Parameter `p` und liefert falls `p` vom Typ `abc` und zudem der Wert im Feld `b` von `p` eine Liste ist eine Liste zurück, deren erstes Element der Wert von Feld `a` in `p` ist, und der Rest der zurückgelieferten Liste ist die Liste im Feld `b` von `p` (also eine Liste in der Liste). Andernfalls liefert `foo` einfach `false` zurück.

Definieren Sie nun eine Funktion `bar1`, die einen Paramter `lst` übergeben bekommt. Für jedes Element x in `lst`, das vom Typ `abc` ist, soll die Ergebnisliste von `bar1` das Ergebnis der Anwendung von `foo` auf x enthalten. Weitere Elemente darf die Ergebnisliste von `bar1` nicht enthalten.

Verbindliche Anforderung: Sie dürfen in dieser Aufgabe noch keine Funktionen höherer Ordnung wie `map` oder `filter` verwenden. Diese Funktionalitäten müssen von Ihnen selbst implementiert werden.

V6 Foo Reloaded II ★ ★ ☆

Definieren Sie nun eine Funktion `bar2`. Diese besitzt die gleiche Funktionalität wie `bar1` aus Aufgabe V5. In dieser Aufgabe wird die Funktionalität allerdings nicht mehr selbst geschrieben, sondern an die vordefinierten Funktionen `map` und `filter` delegiert. Nutzen Sie Lambda-Ausdrücke, welche Sie innerhalb der Aufrufe von `map` und `filter` definieren.

V7 Kartesisches Produkt ★★☆

Definieren Sie eine Funktion `cartesian-prod`, die zwei Zahlenlisten erhält und das kartesische Produkt der beiden bildet. Beispiel:
`(cartesian-prod (list 1 2)(list 3 4))`
→ `(list (list 1 3)(list 1 4)(list 2 3)(list 2 4))`

Verwenden Sie eine Kombination aus Funktionen höherer Ordnung und Lambda-Ausdrücke für Ihre Lösung.

V8 Bibliothek Leihgebühren ★★★

Eine Bibliothek verwaltet ihr Leihsystem nun in Racket. Dazu wird ein neuer Struct-Typ `br` definiert.

```
(define-struct br (id pop type))
```

Das Feld `id` ist dabei ein String und stellt die ID-Nummer des ausgeliehenen Buches dar. Das Feld `pop` ist eine Zahl zwischen 1 bis 6 und gibt die Beliebtheit des Buches an (je größer die Zahl, desto beliebter das Buch). Das letzte Feld `type` ist wieder ein String, der entweder `"Single"` oder `"Subscription"` sein kann, je nachdem ob es sich um eine einmalige Ausleihe oder einen Abonnenten handelt.

Folgende Regeln gelten in der Bibliothek: Abonnenten zahlen für jedes ausgeliehene Buch pauschal 1,50€. Bei normalen Kunden berechnet sich der Preis über die Beliebtheit des Buches. Pro Beliebtheitsstufe kostet das Buch 1,75€. Somit kostet ein Buch mit Beliebtheitsstufe 3 beispielsweise 5,25€.

Ihre Aufgabe ist es nun eine Funktion `fee-total` zu definieren. Diese enthält eine Liste von `br`-Structs (die Ausleihliste) und gibt die Gesamteinnahmen aus eben dieser Ausleihliste zurück.

V9 Wer bekommt die Zulassung? ★★★

Schreiben Sie eine Prozedur zur Prüfung der Zuteilung einer Studienleistung im Modul X. Dort sind 50 Hausaufgaben-, 35 Zwischenklausur- und 50 Projektpunkte sowie insgesamt mindestens 180 Punkte aus den drei Bereichen zusammen erforderlich. Definieren Sie dazu eine Funktion `passed` mit folgender Signatur

```
(list of number)(list of (list of number number number))-> (list of number)
```

Diese Funktion erhält aus Datenschutzgründen separat die Liste der Matrikelnummern sowie eine Liste von Listen mit Punkten für Hausaufgaben, Zwischenklausur und Projekt (in dieser Reihenfolge). Die Precondition ist dabei, dass die Listen gleich lang sind und dass die Matrikelnummer an Position i der ersten Liste zu den Punkten an Position i der zweiten Liste gehört (vergessen Sie die Precondition nicht im Vertrag der Funktion). Die Ergebnisliste enthält die Matrikelnummern aller Studierenden, die die Bedingungen für die Studienleistung erfüllt haben. Die Reihenfolge der Studierenden soll dabei erhalten bleiben.

V10 Bildverarbeitung in Racket ★ ★ ★

Funktionen aus dieser Aufgabe können auch für Aufgabe H verwendet werden.

Um diese Aufgaben in DrRacket ausführen zu können, setzen Sie bitte `(require 2htdp/image)` in die oberste Zeile Ihrer Datei ein (vergleichen Sie auch die Vorlage der Hausübung).

Bilder bestehen aus vielen aufeinanderfolgenden Pixeln. Jedes Pixel nimmt dabei genau die Farbe an, die durch sein sogenanntes RGB-Tripel beschrieben werden. Dies ist durch die Darstellung im sogenannten RGB-Farbraum, ein sogenannter technischer Farbraum, der die Farbwahrnehmung durch das additive Mischen der drei Grundfarben nachbildet, begründet. Jede Farbe lässt sich dabei durch ein Tripel (R, G, B) darstellen, wobei die drei Zahlen jeweils den Anteil der jeweiligen Grundfarbe angeben. So ist das klassische rot durch $(255, 0, 0)$, gelb als Mischung zweier Grundfarben durch $(255, 255, 0)$ und braun als Mischung aller Grundfarben als $(153, 102, 51)$ dargestellt.

Der Einfachheit wegen benutzen wir nur Bilder im PNG-Format (dh. Dateiendung .png), die keine transparenten Farben enthalten, also keinen Alphakanal besitzen. Ein Bild ist in Racket immer ein Struct vom Typ `image`. Jedes Bild besteht aus seinen aufeinanderfolgenden Pixeln. In Racket ist ein Pixel als `color`-Struct definiert:

```
(define-struct color (red green blue alpha))
```

Die ersten drei Felder sind das Tripel des RGB-Farbraums und liegen zwischen 0 und 255. Den Alpha-Wert ignorieren wir in dieser Übung, er soll immer auf 255 gesetzt werden. Folgende Funktionen gibt es bereits für die Bildverarbeitung in Racket:

- Um aus einem `image` die entsprechenden `color`-Structs zu bekommen, gibt es die Funktion `(image->color-list img)`. Diese gibt eine Liste von `color`-Structs für das übergebene Bild zurück.
- Um aus einer Liste von `color`-Structs ein Bild zu generieren gibt es die Funktion `(color-list->bitmap clr-lst width height)`. Diese benötigt neben der Liste von `color`-Structs auch die Breite und Höhe des zu generierenden Bildes (über die Funktionen `(image-width img)` und `(image-height img)` abfrufbar).
- Mit `(bitmap/file "image.png")` laden Sie das Bild im PNG-Format namens "image", welches im gleichen Verzeichnis wie die .rkt Datei liegt. Mittels `(save-image img "out.png")` speichern Sie ein `image`-Struct unter dem Namen "out" dort.

Nutzen Sie für die folgenden beiden Aufgaben Funktionen höherer Ordnung!

1. Definieren Sie eine Funktion `(count-black-white img)`. Diese bekommt ein Bild übergeben, welches nur aus schwarzen (0,0,0) und weißen Pixeln (255,255,255) besteht. Zurückgegeben werden soll eine zweielementige Liste, welche an erster Position die Anzahl an schwarzen und an zweiter Position die Anzahl an weißen Pixeln enthält.
2. Definieren Sie eine Funktion `(negative-transformation img)`. Diese bekommt ein Bild als `image`-Struct übergeben und gibt die Negativtransformation dieses Bildes zurück. Dazu berechnen Sie die RGB-Werte für jeden Pixel neu über den folgenden Zusammenhang: $(R_{\text{neg}}, G_{\text{neg}}, B_{\text{neg}}) = (255 - R, 255 - G, 255 - B)$

H Zweite Hausübung — Gesamt 10 Punkte
Protanopie

In dieser zweiten Hausübung beschäftigen Sie sich mit der sogenannten Protanopie, der Rot-Grün-Blindheit. Darunter versteht man eine Anomalie der Netzhaut, die zur Farbfehlsichtigkeit führt und zur Folge hat, dass Rottöne nicht mehr wahrgenommen werden können.

Die Netzhaut des menschlichen Auges besteht aus zwei Arten von Sinneszellen - die Stäbchen und die Zapfen. In dieser Hausübung beschäftigen wir uns nur mit den Zapfen der Netzhaut, da diese für das Farbsehen bei Tageslicht verantwortlich sind. Wir unterscheiden hierbei drei verschiedene Typen von Zapfen:

- S-Zapfen (Short wavelength/kurze Wellenlänge): Decken den blauen Bereich des sichtbaren Farbspektrums ab.
- M-Zapfen (Medium wavelength/mittlere Wellenlänge): Decken einen Bereich zwischen blauem und orangem Licht ab.
- L-Zapfen (Long wavelength/lange Wellenlänge): Decken die Wahrnehmung von rotem Licht ab.

Bei Menschen mit Protanopie sind entweder keine L-Zapfen auf der Netzhaut vorhanden oder die vorhandenen L-Zapfen liefern eine falsche Farbantwort. Dies hat zur Folge, dass Menschen mit Protanopie im kurzwelligen Bereich (wie Farbgesunde) ein sattes Blau, im mittelwelligen Bereich Grau und im langwelligen Bereich ein sattes Gelb sehen. Ihr Ziel ist es, in dieser Aufgabe die Protanopie zu simulieren. Dafür finden Sie Beispiele in Abbildung 1.

Abbildung 1: Beispielbilder vor der Simulation

Abbildung 2: Beispielbilder nach angewandter Simulation der Protanopie

Bildquellen: Äpfel, Bahn und Regenbogen.

Um diese Simulation zu bewerkstelligen, werden Sie mit zwei verschiedenen Farbräumen arbeiten. Der erste ist der sehr populäre RGB-Farbraum, den Sie bereits in den vorbereitenden Übungen kennengelernt haben. Der zweite Farbraum ist der LMS-Farbraum. Dieser erhält auch ein Tripel (L, M, S), wobei die drei Einträge an die menschlichen Zapfen angelehnt sind (verlgeiche oben). Weitere Details sind für diese Übung nicht notwendig. Wir werden Sie in den nachfolgenden Aufgaben Schritt für Schritt anleiten.

Achtung: Verwenden Sie in allen Teilaufgaben Funktionen höherer Ordnung und Lambda-Ausdrücke, wo immer das möglich und sinnvoll ist. Falls Sie Funktionen von Grund auf selbst implementieren, die Sie mit Funktionen höherer Ordnung und Lambda-Ausdrücken kompakter hätten ausdrücken können, müssen Sie mit Punktabzügen rechnen. Ist die Verwendung explizit gefordert, verlieren Sie auf jeden Fall Punkte, sollten Sie diese Anforderung ignorieren.

Matrixarithmetik

Für die hier notwendigen Berechnungen werden wir Vektoren und Matrizen verwenden, wie Sie sie schon aus der Schulmathematik kennen sollten. Um diese in Racket darzustellen, verwenden wir verschachtelte Listen.

Für die Matrix $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ schreiben wir `(list (list 1 2 3)(list 4 5 6))`.

Für den Vektor $\begin{pmatrix} 7 \\ 8 \\ 9 \end{pmatrix}$ schreiben wir `(list (list 7)(list 8)(list 9))`.

Sollten Sie mit der Matrizenmultiplikation nicht vertraut sein, so holen Sie dies an dieser Stelle nach. Ein gutes Beispiel findet unter folgendem Link:

https://mathepedia.de/Multiplikation_Definitionen_und_Operationen.html

In Worten bedeutet eine Matrizenmultiplikation folgendes: Um den Eintrag der Ergebnismatrix in Reihe i und Spalte j zu bekommen, so nehmen Sie die Reihe i der linken Matrix, die Spalte j der rechten Matrix und betrachten diese beide als Vektoren. Das Skalarprodukt der beiden ist dann der gesuchte Eintrag.

H1 Matrizenmultiplikation 3 Punkte

Definieren Sie zu Beginn eine Funktion `(matrix-multiplication m1 m2)`.

Diese erhält zwei Matrizen im zuvor beschriebenen Format und liefert die Ergebnismatrix im gleichen Format zurück.

Verbindliche Anforderung: Verwenden Sie für die Berechnung der Ergebnismatrix das in Aufgabe V3 vorgestellte Skalarprodukt an geigneter Stelle. Deligieren Sie die Berechnung des Skalarproduktes an die Funktionen `foldr` und `map`.

Tipp: In dieser Aufgabe könnte es hilfreich sein zusätzliche Funktionen zu definieren, die Sie für die Berechnung verwenden wollen.

H2 Farbe zu Vektor vice versa 2 Punkte

Definieren Sie in dieser Aufgabe zwei Funktionen.

1. Die Funktion `(color->vector clr)` bekommt ein `color`-Struct übergeben und gibt die RGB-Werte als Vektor zurück (also als Liste von Listen von Zahlen).
2. Die Funktion `(vector->color vec)` bekommt einen Vektor übergeben und gibt die Farbe als `color`-Struct zurück. Runden Sie die RGB-Werte immer auf ganze Zahlen herunter. Sollten Werte kleiner als 0 oder größer als 255 auftreten, so setzen Sie diese Werte einfach auf 0 beziehungsweise 255.

Hinweis: Beachten Sie für diese und die nachfolgenden Übungen nochmal besonders die Hinweise in Aufgabe V10.

H3 Simulation der Protanopie-Werte für einen Vektor 1 Punkt

In dieser Aufgabe wollen wir die Simulation der Protanopie an einem gegebenen RGB-Vektor durchführen[1]. Definieren Sie dazu die Funktion

`(rgb-vector-protanopia vec)`

die einen RGB-Vektor bekommt und den Vektor mit den transformierten Werten zurückgibt.

Dazu transfomieren Sie den gegebenen RGB-Vektor zunächst in den LMS-Farbraum:

$$\begin{pmatrix} \mathrm{L} \\ \mathrm{M} \\ \mathrm{S} \end{pmatrix} = \begin{pmatrix} 17.8824 & 43.5161 & 4.1193 \\ 3.4557 & 27.1554 & 3.8671 \\ 0.02996 & 0.18431 & 1.4670 \end{pmatrix} \cdot \begin{pmatrix} \mathrm{R} \\ \mathrm{G} \\ \mathrm{B} \end{pmatrix}$$

Dann simulieren Sie die Protanopie im LMS-Farbraum:

$$\begin{pmatrix} \mathrm{L}_{\mathrm{Protanopie}} \\ \mathrm{M}_{\mathrm{Protanopie}} \\ \mathrm{S}_{\mathrm{Protanopie}} \end{pmatrix} = \begin{pmatrix} 0 & 2.02344 & -2.52581 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \mathrm{L} \\ \mathrm{M} \\ \mathrm{S} \end{pmatrix}$$

Zum Schluss wandeln Sie die simulierten Werte wieder in den RGB-Farbraum um:

$$\begin{pmatrix} \mathrm{R}_{\mathrm{Protanopie}} \\ \mathrm{G}_{\mathrm{Protanopie}} \\ \mathrm{B}_{\mathrm{Protanopie}} \end{pmatrix} = \begin{pmatrix} 0.0809 & -0.1305 & 0.1167 \\ -0.0102 & 0.0540 & -0.1136 \\ -0.0003 & -0.0041 & 0.6935 \end{pmatrix} \cdot \begin{pmatrix} \mathrm{L}_{\mathrm{Protanopie}} \\ \mathrm{M}_{\mathrm{Protanopie}} \\ \mathrm{S}_{\mathrm{Protanopie}} \end{pmatrix}$$

Alle benötigten Matrizen sind bereits in der Vorlage definiert.

[1] Als Grundlage für die Berechnungen dieser Hausübung wurde die hier verlinkte Quelle genutzt.

Für die Aufgaben H4-H6 brauchen Sie keine eigenen Tests zu schreiben.

H4 Protanopie-Simulation für ein gegebenes Bild I 1 Punkt

Definieren Sie die Funktion `(protanopia-recursive img)`, die ein Bild übergeben bekommt und die Protanopie-Simulation rekursiv auf jedes Pixel anwendet und das daraus resultierende Bild (`image`-Struct) zurückgibt.

Verbindliche Anforderung: Sie dürfen in dieser Aufgabe keine Funktionen höherer Ordnung verwenden, sondern nur "ganz normale"Rekursion. Definieren Sie sich geeignete, zusätzliche Funktionen zur Hilfe.

H5 Protanopie-Simulation für ein gegebenes Bild II 1 Punkt

Definieren Sie die Funktion `(protanopia-map img)`, die ein Bild übergeben bekommt und die Protanopie-Simulation auf jedes Pixel, mittels Funktionen höherer Ordnung, anwendet. Kommt Ihnen bekannt vor?

Verbindliche Anforderung: Außerhalb der Funktionen `map`, `filter` und `foldr` darf keine Rekursion verwendet werden.

H6 Ähnlichkeit zweier Bilder 2 Punkte

Zum Abschluss wollen wir nun feststellen, wie viele Pixel eines Ausgangsbildes durch die Protanopie-Simulation gar nicht oder nur minimal modifiziert wurden. Dazu bestimmen wir die Anzahl an denjenigen Pixeln des Ausgangsbildes, die eine kleine Differenz zwischen dem jeweilig zugehörigen Pixel des Bildes mit angewandter Protanopie-Simulation aufweisen. Die Differenz zwischen zwei RGB-Werten, berechnen Sie über den folgenden Zusammenhang:

$$\text{Diff}_{(R_1,G_1,B_1),(R_2,G_2,B_2)} = |R_1 - R_2| + |G_1 - G_2| + |B_1 - B_2|$$

Anschließend teilen wir diese Anzahl durch die Gesamtanzahl an Pixeln im Ausgangsbild und multiplizieren das daraus resultierende Ergebnis mit 100, um den prozentualen Anteil an Pixeln zu erhalten, die gar nicht oder nur minimal modifiziert wurden. Definieren Sie dazu die folgende Funktion `(image-similarity img1 img2 max-difference)`, die zwei gleichgroße Bilder und eine Zahl übergeben bekommt und zuerst die Differenzen zwischen allen RGB-Werten der beiden Bilder berechnet und eine Liste aus ihnen konstruiert. Anschließend entfernen Sie alle Differenzen aus eben dieser Liste, die größer als die übergebene Zahl `max-difference` ist. Nun teilen Sie die Anzahl an Elementen in dieser Liste durch die Gesamtanzahl an Pixeln im Ausgangsbild und multiplizieren das daraus resultierende Ergebnis mit 100, um den prozentualen Anteil an Pixeln zurückzugeben, die gar nicht oder nur minimal modifiziert wurden.

Verbindliche Anforderung: Außerhalb der Funktionen `map`, `filter` und `foldr` darf keine Rekursion verwendet werden. Realisieren Sie die Differenzfunktion als Lambda-

Ausdruck und definieren Sie das Filterprädikat innerhalb der Aufrufe von `map` bzw. `filter` unter Verwendung von `lambda`-Ausdrücken.

H7 Protanopie-Simulation auf Bildern 0 Punkte

Mithilfe der Vorlage können Sie nun die Protanopie-Simulation auf Bilder anwenden. Nutzen Sie die im Bildverarbeitungsabschnitt genannten "nützlichen" Funktionen um ein Bild einzulesen, die Simulation durchzuführen und anschließend abspeichern zu können.

In der Vorlage zur Hausübung finden Sie bereits einige Bilder aus Abbildung 1, welche Sie dafür nutzen können.

Hinweise zur Bearbeitung:

Die Aufgaben bauen hier teilweise aufeinander auf. Für die korrekte Ausführung von Aufgabe H3 wird beispielsweise die Implementierung von Aufgabe H1 vorausgesetzt.

Bearbeiten Sie die Aufgabe dann trotzdem und tun Sie so, als hätten Sie die hervorgehenden Aufgaben korrekt lösen können. Wir werden in diesen Fällen Ihre fehlerhafte oder fehlende Lösung durch die Referenzlösung ersetzen. Sollte dann Aufgabe H3 korrekt implementiert sein und die Ausführung nur an Aufgabe H1 scheitern erhalten Sie trotzdem für diese Aufgabe dann volle Punkte.

Erinnerung an verbindliche Anforderungen: Für alle Racket-Hausübungen gilt, dass Zuweisung mittels `set!`, `let`, `begin` oder ähnliches nicht erlaubt ist! Sollten Sie durch eigene Recherche auf dieses Konstrukt stoßen dürfen Sie es nicht verwenden, Sie müssen die Hausübungen mittels normaler Rekursion, wie in der Vorlesung vorgestellt, lösen.