Grundlagen der Informatik 1

Wintersemester 2009/2010

Prof. Dr. Max Mühlhäuser, Dr. Rößling
http://proffs.tk.informatik.tu-darmstadt.de/teaching

Übung 4 Version: 1.0 09. 11. 2009

---

1 Mini Quiz

Kreuzen Sie die wahren Aussagen an.

1. ☐ Der allgemeine Vertrag `contract: (listof X) -> boolean` ist mit diesem Vertrag vereinbar: `f: (listof number) -> number`.
2. ☐ Der allgemeine Vertrag `contract: (listof X) -> (X -> boolean)` ist mit diesem Vertrag vereinbar: `f: (listof number) -> (number -> boolean)`.
3. ☐ `f: (listof X -> X) -> boolean` ist ein sinnvoller Vertrag.
4. ☐ `lambda` ist ein Spezialfall von `local` zur Definition anonymer Funktionen.
5. ☐ Die Funktion `(lambda (x y) (+ x y))` erfüllt den Vertrag `number number -> number`.

2 Fragen

1. Was sind Prozeduren höherer Ordnung? Was sind ihre Vorteile?
2. Warum sollte man Abstraktion beim Programmieren verwenden?

Hinweis: Verwenden Sie für diese Übung bitte die Sprache „Intermediate Student with lambda“.

3 Local und Lambda (K)

Schreiben Sie die folgende Prozedur um, indem Sie einen äquivalenten `local` Ausdruck anstelle von `lambda` verwenden.

```
1 ;; multiply : number -> (number -> number)
2 ;; returns a function that takes a number as input and
3 ;; and returns the number multiplied by x
4 (define (multiply x)
5    (lambda (y) (* x y)))
```

4 Prozeduren Höherer Ordnung (K)

Sie kennen bereits Prozeduren wie $+$, die mehrere Argumente konsumieren. In dieser Aufgabe wollen wir nun zeigen, wie man solche Funktionen erzeugen kann aus Funktionen, die nur ein Argument konsumieren.

1. Wie lautet der Vertrag einer Funktion *f*, die zwei Zahlen konsumiert und eine Zahl liefert?
2. Wie lautet der Vertrag einer Funktion *g*, die eine Zahl konsumiert und eine Funktion liefert, die wiederum eine Zahl konsumiert und eine Zahl liefert?
3. Definieren Sie die Funktion `add2`, die 2 zu einer übergebenen Zahl hinzuaddiert.
4. Definieren Sie die Funktionen `add3` und `add4`, die 3 bzw. 4 zu einer übergebenen Zahl hinzuaddieren.
5. Definieren Sie die Funktion `make-adder: number -> (number -> number)`. Sie soll eine Zahl `x` konsumieren und eine Funktion erzeugen. Die erzeugte Funktion soll eine Zahl `y` konsumieren und als Ergebnis `x + y` zurück liefern. Zum Beispiel soll `(make-adder 3)` äquivalent zu `add3` sein. Verwenden Sie `lambda`, um die zurückgelieferte Funktion zu definieren. Definieren Sie `add2, add3` und `add4` mit Hilfe von `make-adder`.

   Hinweis: Eine Beispielnutzung von make-adder sieht wie folgt aus: `((make-adder 3) 4)` mit Ergebnis 7.
6. Definieren Sie die Funktion `add`, die zwei Argumente addiert, unter Verwendung von `make-adder`.
7. Definieren Sie die Funktion `uncurry`, die eine Funktion `f` mit einem Argument konsumiert und eine Funktion `g` mit zwei Argumenten zurückliefert. Dabei soll die erste Funktion `f` eine Funktion höherer Ordnung sein, die Funktionen erzeugt, die auf dem zweiten Argument von `g` operieren. Der Vertrag von `uncurry` ist also: `uncurry: (X -> (Y -> Z)) -> (X Y -> Z)`.
8. Definieren Sie nun `add` unter Verwendung von `make-adder` und `uncurry`.
9. Wozu könnte die Beschränkung auf Funktionen von nur einem Argument sinnvoll sein? Warum erlaubt man in der Praxis dennoch Funktionen auf mehreren Variablen?

5 Fold, Map und Co (K)

1. Wie ist der Vertrag von `map`?
2. Definieren Sie die Funktion `mymap`, die für eine Funktion und eine Liste als Parameter das Selbe tut wie `map`, aber ohne `map` zu verwenden.
3. Zu was wird `(foldl + 4 '(1 2 3))` ausgewertet?
4. Zu was wird `(map + '(1 2 3) '(4 5 6))` ausgewertet?
5. Definieren Sie die Prozedur `zip`, die aus zwei gleich langen Listen eine Liste von geordneten Paaren macht. Beispiel: `(zip '(a b c) '(1 2 3))` ergibt `(list '(a 1) '(b 2) '(c 3))`.
6. Definieren Sie eine Funktion `vec-mult`, die zwei gleich lange Vektoren (Listen von Zahlen) erhält und das Skalarprodukt berechnet. Verwenden Sie `map` und/oder `foldl`.
7. Definieren Sie eine Funktion `cartesian-product`, die zwei Listen $l_1 : (l_{11}...l_{1n})$ und $l_2 : (l_{21}...l_{2n})$ erhält und das kartesische Produkt $((l_{11}, l_{21}), ...(l_{11}, l_{2n}), ...(l_{1n}, l_{21}), ...(l_{1n}, l_{2n}))$ dieser beiden Listen zurück gibt. Verwenden Sie `map` und `foldr`.

Hausübung

Die Vorlagen für die Bearbeitung werden im GdI1-Portal bereitgestellt. Kommentieren Sie Ihren selbst erstellten Code. Die Hausübung muss bis zum Abgabedatum im GdI1-Portal abgegeben werden. Der Fachbereich Informatik misst der Einhaltung der Grundregeln der wissenschaftlichen Ethik großen Wert bei. Zu diesen gehört auch die strikte Verfolgung von Plagiarismus. Mit der Abgabe Ihrer Hausübung bestätigen Sie, dass Sie bzw. Ihre Gruppe alleiniger Autor des gesamten Materials sind. Falls Ihnen die Verwendung von Fremdmaterial gestattet war, so müssen Sie dessen Quellen deutlich zitieren. Falls Sie die Hausübung in einer Lerngruppe bearbeitet haben, geben Sie dies bitte deutlich bei der Abgabe an. Alle anderen Mitglieder der Lerngruppe müssen als Abgabe einen Verweis auf die gemeinsame Bearbeitung einreichen, damit die Abgabe im Portal auch für sie bewertet werden kann.

Abgabedatum: Freitag, 20. 11. 2009, 16:00 Uhr

Denken Sie bitte daran, Ihren Code hinreichend gemäß den Vorgaben zu kommentieren (Scheme: Vertrag, Beschreibung und Beispiel sowie zwei Testfälle pro Funktion; Java: JavaDoc). Zerlegen Sie Ihren Code sinnvoll und versuchen Sie, wo es möglich ist, bestehende Funktionen wiederzuverwenden. Wählen Sie sinnvolle Namen für Hilfsfunktionen und Parameter.

Hinweis:

Im Portal stehen neben dem eigentlichen Kurs noch die folgenden Hilfsmittel als separate Kurse für Sie bereit:

- Kurs „Hilfen zur Nutzung des Portals“: hier finden Sie Information zum Portal und wie man es möglichst effizient nutzen kann. Das Portal kann sehr viel, und nicht alles davon sieht man auf den ersten Blick. Daher empfehlen wir Ihnen sehr, sich auch in diesen Kurs einzutragen und die dortigen Materialien durchzuarbeiten!
- Kurs „Tipps zum effektiven Studieren“: hier erhalten Sie Tipps rund um das Studium, etwa zum Bilden von Lerngruppen oder der Vorbereitung auf Klausuren. Eine Einschreibung in diesen Kurs ist daher ebenfalls ratsam. Zusätzlich arbeiten wir an Tipps zur Programmierung, etwa Hinweisen zum korrekten Verfassen von Verträgen in Scheme.

Nutzen Sie diese Hilfsangebote! Kommentare und Anregungen zu weiteren Themen sind willkommen.

6 Bild-Operationen (7 Punkte)

Bei der Verarbeitung von Bildern stellt sich oftmals das Problem, dass man Kanten im Bild erkennen möchte. Kanten sind dabei einfach große Farb- oder Helligkeitsunterschiede in der lokalen Umgebung eines Bildpunkts. Eine Möglichkeit zur Erkennung von Kanten ist der sogenannte *Sobel-Operator*. Dabei wird für jeden Bildpunkt in dessen Umgebung geschaut, wie sehr sich der Farbwert in horizontaler oder vertikaler Richtung verändert. Der Betrag der Änderung bestimmt den neuen Farbwert in einem transformierten Bild, welches die Kanten verstärkt darstellt.

Ein Anwendungsgebiet dieser Kantenerkennung ist etwa die Analyse von Aufnahmen einer Fahrbahn. Verwandte Techniken nutzen aktuelle Oberklasseautos sowie LKWs, um zu erkennen, wenn der Fahrer (unabsichtlich) seine Fahrspur verlässt - und damit potenziell Unfälle zu vermeiden.

In dieser Aufgabe sollen Sie eine vereinfachte Variante des Sobel-Operators zur Kantenerkennung implementieren. Dieser vereinfachte Operator arbeitet nicht auf einer 3x3 Umgebung der Bildpunkte, sondern lediglich auf einer 3x1 Umgebung. Entsprechend werden nur Helligkeitsänderungen in horizontaler Richtung betrachtet. Der horizontale Sobel-Operator kann dabei vereinfacht für die 3x1 Umgebung eines Bildpunktes wie folgt als Vektor aufgefasst werden:

```
1 (define sobel-we (list 2 0 -2))
```

1. Fügen Sie in DrScheme über “Sprache → Teachpack hinzufügen . . . ” das Teachpack `image.ss` hinzu. Laden Sie nun das Bild 1 “highway.png” aus dem Portal herunter und binden Sie es über “Einfügen → Bild einfügen . . . ” in ihr aktuelles Scheme Dokument ein. Geben Sie ihm per **define** den Namen “highway”.

Abbildung 1: Eine zweispurige Autobahn, © Wikimedia Commons

Hinweis: Eine Übersicht über die Funktionalität des `image.ss` Teachpacks finden Sie unter `http://download.plt-scheme.org/doc/372/html/teachpack/image.html`. Zum Arbeiten mit Bildern werden diese per `image->color-list` in eine Liste von color-Strukturen überführt, welche man per `color-list->image` wieder in ein Bild überführen kann.

Wichtiger Hinweis: Entfernen Sie bitte **unbedingt das Bild aus Ihrem Quelltext** *bevor* Sie ihn abgeben!

2. Schreiben Sie eine Scheme-Prozedur `brightness: color -> number`, welche für eine gegebene color-Struktur deren Helligkeit als arithmetisches Mittel der Werte der einzelnen Farbkomponenten zurückliefert (1 Punkt).

3. Implementieren Sie nun eine Prozedur

   apply−for−env3x1: (listof color) ((listof color −> color)) −> (listof color)

   Diese Prozedur konsumiert eine Liste von color-Strukturen und eine Prozedur und liefert als Ergebnis die Anwendung der übergebenen Prozedur auf allen zusammenhängenden Tripeln der übergebenen Liste. Für die letzten beiden Aufrufe können Sie die Tripel einfach mit (make-color 0 0 0) auffüllen. Ignorieren Sie Zeilenumbrüche im Bild (2 Punkte).

4. Schreiben Sie die Prozedur sobel: (**list** color color color) −> color.

   Diese konsumiert eine Liste von drei color-Strukturen und produziert eine color-Struktur. Verrechnen Sie die Helligkeitswerte des übergebenen Tripels mit der am Anfang der Aufgabe angegebenen *sobel-we* Liste, indem Sie das Skalarprodukt (siehe Präsenzaufgaben) der beiden per `vec-mult` bilden und eine neue color-Struktur mit dem Skalarprodukt als Grauwert erzeugen (2 Punkte).

   Hinweis: Das Skalarrodukt ist potentiell größer als erlaubte Werte für eine Farbkomponente in `(make-color red green blue)` und/oder keine natürliche Zahl. Verwenden Sie der Einfachheit halber `floor: number -> number` zum Abrunden, `min: number number -> number` zum Begrenzen des Zahlenbereichs und `abs: number -> number` zur Bildung des Betrags.

5. Implementieren Sie die Scheme-Prozedur `image-transform-3x1`, welche eine image-Struktur und eine Prozedur entgegennimmt und die oben implementierte `apply-for-env3x1` Prozedur mit der übergebenen Prozedur verwendet, um ein neues Bild zu erstellen (2 Punkte).

   Hinweis: `color-list->image` wird in der oben genannten Übersicht beschrieben. Als "Pinhole" können Sie einfach 0 0 angeben. Für diese Prozedur brauchen Sie *keine Tests zu schreiben* (siehe unten).

Hinweis: Das Ergebnisbild ist schwer zu testen. Erstellen Sie stattdessen ein neues Bild mit den erkannten Kanten, indem Sie das Bild des Highways mit der `sobel` Prozedur transformieren. Wenn Sie alles richtig gemacht haben, sollte ein Bild wie in Abbildung 2 erscheinen. Vergleichen Sie Ihr Bild optisch mit dem Beispiel aus Abbildung 2.

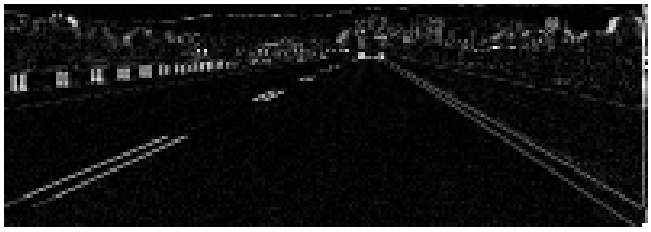

Abbildung 2: Bild mit verstärkten Kanten