Übungen zu Softwareentwicklung III, Funktionale Programmierung

Blatt 10, Woche 12

Funktionen höherer Ordnung und kombinatorische Probleme

Leonie Dreschler-Fischer

WS 2014/2015

Ausgabe: Freitag, 9.2015,

Abgabe der Lösungen: bis Montag, 18.1.2015, 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 Rekursion zur Lösung kombinatorischer Probleme vertraut zu machen.

Bearbeitungsdauer: Die Bearbeitung sollte insgesamt nicht länger als 6 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 Listen als Mengen

(Bearbeitungszeit 2 1/2 Std.)

1.1 Allquantor und Existenzquantor als Funktionen höherer Ordnung:

6 Pnkt.

1. Definieren Sie ein Prädikat `every p? xs`, das überprüft, ob für *jedes* Element der Liste xs das Prädikat p? erfüllt ist. `every` entspricht dem *Allquantor„für alle“.*

   ```
   (every (curry =) '(3 3 3)) ⟶ #t
   ```

2. Definieren Sie weiterhin ein Semiprädikat `some p? xs`, das überprüft, ob mindestens ein Element der Liste xs das Prädikat p? erfüllt und das gefundene Element als Resultat zurückgibt.

   `some` entspricht dem Existenzquantor *„es gibt“.*

   ```
   (some (curry =) '(1 3 9) ) ⟶ 3
   ```

1.2 Prädikate über Relationen

14 Pnkt.

Gegeben sie eine Menge m, repräsentiert als Liste, sowie eine Relation r als Teilmenge von $m \times m$, die als Liste von *dotted pairs* repräsentiert sei. Verwenden Sie die Prädikate **some** und **every**, um die folgenden Prädikate für Relationen zu definieren:

symmetrisch?: Ist r symmetrisch?

asymmetrisch?: Ist r asymmetrisch?

reflexiv?: Ist r reflexiv?

transitiv? Ist r transitiv?

aequi? r: Ist r eine Äquivalenzrelation?

ord? r: Ist r eine strikte Ordnungsrelation?

2 Das Kreuzprodukt von Mengen: Baumrekursion

(Bearbeitungszeit 1. 1/2 Std.),

Gegeben seien Mengen $M_1, M_2, \ldots$ als Listen ms1, ms2, ... repräsentiert. 7 Pnkt.

1. Definieren Sie eine Funktion Kreuzprodukt, die die Menge $M_1 \times M_2$ errechnet.

   Repräsentieren Sie die Tupel der Relation als Listen mit zwei Elementen. (m1 m2). Beispiel:

$$\begin{aligned} M_1 &= \{1, 2\} \\ M_2 &= \{a, b, c\} \\ M_1 \times M_2 &= \{(1,a)(1,b)(1,c)(2,a)(2,b)(2,c)\} \end{aligned}$$

```
(Kreuzprodukt '(1 2) '(a b c))
  ⟶ '( (1 a) (1 b) (1 c) (2 a) (2 b) (2 c))
```

7 Pnkt.

2. Verallgemeinern Sie die Funktion, so daß Sie das Kreuzprodukt beliebig vieler Mengen berechnen können.

   Die Menge der Basismengen sei als Liste von Listen repräsentiert.

```
(Produkt '((1 2) (a b) (X)))
   ⟶ '( (1 a X) (1 b X) (2 a X) (2 b X) (2 c X))
```

7 Pnkt.

3. Definieren Sie eine Funktion Kombination, die die Liste aller Kombinationen (Auswahl von k verschiedenen Elementen aus einer Menge M der Mächtigkeit n ohne Beachtung der Reihenfolge) berechnet.

```
(Kombination 2 '(a b c) )
   ⟶ '( (a b) (a c) (b c))
```

3 Wiederholung, Klausurvorbereitung

(Bearbeitungszeit 2 Std.)

1. Zu welchen Werten evaluieren die folgenden Racket-Ausdrücke? 5 Zusatzpnkt.

   (a) (**max** (**min** 2 (− 2 3)))

   (b) ‘(+ ,(− 2 4) 2)

   (c) (car '(Alle meine Entchen))

   (d) (cdr '(schwimmen auf (dem See)))

   (e) (**cons** 'Listen '(sind einfach))

   (f) (**cons** 'Paare 'auch)

   (g)
   ```
   (equal?
     (list 'Racket 'Prolog 'Java)
     '(Racket Prolog Java))
   ```

   (h)
   ```
   (eq?
    (list 'Racket 'Prolog 'Java)
    (cons 'Racket '(Prolog Java)))
   ```

   (i)
   ```
   (map
     (lambda (x) (* x x x))
     '(1 2 3))
   ```

   (j) (filter odd? '(1 2 3 4 5))

   (k) ((curry **min** 6) 2)

   (l) ((curry = 2) 2)

2. Zur Syntax von Racket: Gegeben seien die folgenden Definitionen: 5 Zusatzpnkt.

   ```
   (define *a* 10)
   (define *b* '*a*)
   (define (merke x)(lambda () x))
   (define (test x)
    (let ((x (+ x *a*))))
    (+ x 2))
   ```

   Haben die folgenden Ausdrücke einen wohldefinierten Wert und zu welchem Wert evaluieren sie?

1. `*a*`
2. `(+ *a* *b*)`
3. `(+ (eval *a*) (eval *b*))`
4. `(and (> *a* 10) (> *b* 3))`
5. `(or (> *a* 10) (/ *a* 0))`
6. `(+ 2 (merke 3))`
7. `(+ 2 ((merke 3)))`
8. `(test 4)`

3. Geben Sie geeignete Racket-Ausdrücke an, um die folgenden Werte zu berechnen: 2 Zusatzpnkt.

   (a) $3 \times 4 + 5 \times 6$

   (b) $\sqrt{1 - \sin^2(x)}$

4. Definieren Sie die folgenden beiden Funktionen c und mytan als Racket-Funktionen: 2 Zusatzpnkt.

$$\begin{aligned} \mathrm{c}(a,b) &= \sqrt{a^2+b^2}, \quad a,b \in R \\ \mathrm{mytan}(\alpha) &= \frac{\sin\alpha}{\sqrt{1-\sin^2\alpha}}, \quad -\frac{\pi}{2} < \alpha < \frac{\pi}{2} \end{aligned}$$

5. Wandeln Sie die folgenden Ausdrücke in die Präfixnotation von Racket um: 2 Zusatzpnkt.

   (a) $1 + 4/2 - 1$

   (b) $\frac{2-\frac{1+3}{3+2*3}}{\sqrt{3}}$

6. Wandeln Sie den folgenden Ausdruck in Infixnotation um: `(* (+ 1 2 3) (- 2 3 (- 2 1)))` 1 Zusatzpnkt.

7. Definieren Sie eine *rekursive* Funktion `(laengen xss)` in Racket, die folgendes leistet:

   Gegeben sei eine Liste von Listen. Bilden Sie diese ab auf die Liste der Längen der Unterlisten. 4 Pnkt.

   Beispiel: `(laengen (1 3 4) ( Auto Bus) () (3 4 5 6))` $\longrightarrow$ `(3 2 0 4)`.

   Geben Sie die Funktion in zwei Varianten an: einmal linear rekursiv ohne Akkumulator und einmal endrekursiv. Woran erkennen Sie eine endrekursive Funktion?

8. Ein abstrakter Datentyp für Längenangaben: 10 Pnkt.

   Für viele Anwendungen ist es notwendig, nicht nur den numerischen Wert einer physikalischen Größe zu kennen sondern auch die Einheit, in der dieser Wert angegeben ist.

   Implementieren Sie die Zugriffsfunktionen für einen Datentyp „laenge“:

   Eine Längenangabe sei repräsentiert als Liste mit zwei Elementen, nämlich dem numerischen Wert und der Einheit, beispielsweise `(0.5 m)`, `(3 cm)`, `(6.3 km)`, `(6 inch)` usw.

   (a) Definieren Sie eine Konstruktorfunktion `(make-length value unit)`, die aus einem Wert und einer Längeneinheit ein Element vom Typ Längenangabe erzeugt:
   `(make-length 3 'cm)` $\longrightarrow$ `(3 cm)`

   (b) Definieren Sie Selektorfunktionen für den Zugriff auf den Wert einer Längenangabe `(value-of-length len)`
   und die Längeneinheit `(unit-of-length len)`: `(value-of-length (make-length 3 'cm))` $\longrightarrow$ `3`
   `(unit-of-length (make-length 3 'cm))` $\longrightarrow$ `cm`

   (c) Definieren Sie eine Skalierungsfunktion `(scale-length len fac)`, mit der Längenangaben um einen bestimmtem Faktor vergrößert (verkleinert) werden können:
   `(value-of-length (scale-length (make-length 3 'cm) 2))` $\longrightarrow$ `6`

   (d) Gegeben sei eine Tabelle `*conversiontable*` mit Umrechnungsfaktoren für Längenangaben. Diese Tabelle sei als Assoziationsliste organisiert, die zu jeder Längeneinheit den Umrechnungsfaktor gegenüber der Angabe in Metern enthält:

```
(define *conversiontable* ;
   '( ;  (unit . factor)
      (m . 1)
      (cm . 0.01)
      (mm . 0.001)
      (km . 1000)
      (inch . 0.0254)
      (feet . 0.3048)
      (yard . 0.9144)))
```

Definieren Sie die folgenden Operationen auf Längenangaben unter Verwendung der Funktionen Konstruktoren und Selektoren `make-length, unit-of-length, value-of-length, scale-length`:

- Auslesen des Umrechnungsfaktors aus der Tabelle `(factor unit)`:
  `(factor 'mm)` $\longrightarrow$ `0.001`

- Normalisierung (Wandlung in Meterangaben) `(length->meter len)`: `(length->meter '(3 cm)))` $\longrightarrow$ `(0.03 m)`

- Vergleich zweier Längenangaben `(length< len1 len2)` und `(length= len1 len2)`:
  `(length< '(3 cm) '(6 km))` $\longrightarrow$ `#t`
  `(length= '(300 cm) '(3 m))` $\longrightarrow$ `#t`

- Addition und Subtraktion zweier Längenangaben `(length+ len1 len2)`:
  `(length+ '(3 cm) '(1 km))` $\longrightarrow$ `(1000.03 m)`
  `(length- '(1.001 km) '(1 m))` $\longrightarrow$ `(1000.0 m)`

(e) Gegeben sei eine Liste von Längenangaben:
`( (6 km) (2 feet) (1 cm) (3 inch))`.
Verwenden Sie Funktionen höherer Ordnung (`reduce, map, filter, iterate, curry`...), um funktionale Ausdrücke für folgende Werte zu schreiben:

- Die Abbildung der Liste auf eine Liste, die die Längenangaben in Metern enthaelt,
- eine Liste aller Längen, die kürzer als 1m sind,
- die Summe der Längenangaben in Metern,
- die Längenangabe mit der kürzesten Länge in der Liste.

Erreichbare Punkte: 55

Erreichbare Zusatzunkte: 17