Lehrstuhl für Kognitive Modellierung
Prof. Dr. Martin V. Butz
Dr. Jan Kneissler

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Informatik II

Programmierübungsblatt 4

Ausgabe: Mo, 19.05.2014 Abgabe: 21te Kalenderwoche

Für die Bearbeitung der Übungsaufgaben haben Sie 90 Minuten Zeit. Bitte geben Sie Ihre Lösungen der Aufgaben rechtzeitig im Vorlesungssystem unter dem Punkt Präsenzübungen ab. Dazu müssen Sie sich mit Ihrem Benutzernamen und Passwort am Vorlesungssystem anmelden. Sie können mehrmals innerhalb der Bearbeitungszeit Ihre Lösungen abgeben, nur die letzte Version Ihrer Abgabe wird gespeichert. Wenn Sie mit den Übungen fertig sind, bleiben Sie bitte an ihrem Platz sitzen und warten, bis ein Tutor ihre Lösungen bewertet. **Bei Fragen zur Rechnerbenutzung oder zu den Übungsaufgaben wenden Sie sich bitte jederzeit an einen der TutorInnen!**

Aufgaben (insgesamt 22 Punkte)

Notiz: Halten Sie sich beim Programmieren in Scheme an die eingeführte Konstruktionsanleitung:
a) Kurzbeschreibung als Kommentar (mittels: "; ..."");
b) Signatur (der "Vertrag" mittels: "(: ...)");
c) Testfälle (mittels "(check-within ...)" oder "(check-expect ...)" oder ...)
d) Prozedur Gerüst und Rumpf
Benutzen Sie Hilfsprozeduren, wenn Teilprobleme gelöst werden müssen!

Notiz: Die aktuelle `Info2_SchemeEssentials_XX.rkt` Datei steht Ihnen als weitere Hilfestellung zur Verfügung.

4.1 Group-By [22]

Wichtiger Hinweis: In Ihrem Arbeitsverzeichnis ist eine Datei `groupby.rkt` bereitgestellt, die eine Definition der bereits aus der Vorlesung bekannten Higher-Order-Prozedur `filter` enthält. `filter` hat die folgende Signatur:

```
(: filter ((list-of %a) (%a -> boolean) -> (list-of %a)))}
```

Speichern Sie ihren eigenen Code zu dieser Aufgabe ebenfalls in `groupby.rkt`!

Bitte wenden!

a) Schreiben Sie Prozeduren, die Listen mittels der `filter` Prozedur korrekt filtern.

i) Schreiben Sie eine Prozedur `modulo-filter`, die alle durch eine Zahl n teilbaren Zahlen aus einer Liste extrahiert. Die Prozedur soll die folgende Signatur haben und (unter anderen) die folgenden checks erfüllen [2]:

```
(: modulo-filter ((list-of integer) integer -> (list-of
   integer)))
(check-expect (modulo-filter (list 1 2 3 4 5 6) 3) (list 3 6))
(check-expect (modulo-filter (list 2 42 5 7) 6) (list 42))
```

ii) Schreiben Sie eine Prozedur `groesser-filter`, die alle Zahlen größer einer gegebenen Zahl n aus einer gegebenen Liste extrahiert. Die Prozedur soll die folgende Signatur haben und die folgenden checks erfüllen [2]:

```
(: groesser-filter ((list-of real) real -> (list-of real)))
(check-expect (groesser-filter (list 1 2 3 4 5 6) 3) (list 4 5
   6))
(check-expect (groesser-filter (list 2 42 5 7) 6) (list 42 7))
```

iii) Schreiben Sie eine Prozedur `within-filter`, die alle Zahlen größer einer gegebenen Zahl l und kleiner einer weiteren gegebenen Zahl h aus einer gegebenen Liste extrahiert. Die Prozedur soll die folgende Signatur haben und die folgenden checks erfüllen [2]:

```
(: within-filter ((list-of real) real real -> (list-of real)))
(check-expect (within-filter (list 1 2 3 4 5 6) 1 5) (list 2 3
   4))
(check-expect (within-filter (list 2 42 5 7) 5 1000) (list 42
   7))
```

b) **Wichtig**: Benutzen Sie `fold`, um diese Aufgabe zu lösen!

Implementieren Sie eine Higher-Order-Prozedur `contains?`. Die Prozedur akzeptiert eine beliebige Liste l, einen Wert x mit der gleichen Signatur wie die Listenelemente, sowie eine Prozedur, die zwei Werte mit dieser Signatur auf Gleichheit überprüfen kann. `contains?` ergibt genau dann `#t`, wenn der Wert x in der Liste l vorkommt, sonst `#f`. [4]

`contains?` soll folgende Signatur haben:

```
(: contains? ((list-of %a) %a (%a %a -> boolean) -> boolean))
```

Testfall:

```
(check-expect  (contains? (list "foo" "bar" "baz") "bar" string=?)
   #t)
```

Implementieren Sie mindestens zwei weitere Testfälle!

c) **Wichtig**: Benutzen Sie `fold`, um diese Aufgabe zu lösen!

Implementieren Sie eine Prozedur `distinct`, die die *verschiedenen* Elemente einer Liste liefert, also Duplikate entfernt. `distinct` akzeptiert eine beliebige Liste sowie eine Prozedur, die Werte mit der Signatur der Listenelemente auf Gleichheit überprüft. Benutzen Sie die bereits implementierte Prozedur `contains?`. [5]

Hinweis: Die Reihenfolge der Elemente in der Ergebnisliste kann beliebig sein.

Testfall:

```
(check-expect (distinct (list 1 2 2 3 4 1) =) (list 2 3 4 1))
```

Implementieren Sie mindestens zwei weitere Testfälle!

d) Implementieren Sie eine Prozedur `groupby`, die eine Liste nach einem beliebigen Kriterium gruppiert. `groupby` akzeptiert eine beliebige Liste (Signatur `(list-of %a)`). Außerdem akzeptiert `groupby` eine Prozedur mit der Signatur `(%a -> %b)`, die einen Wert von der Signatur der Listenelemente (`%a`) auf einen Gruppierungswert mit der Signatur `%b` abbildet. Als Drittes akzeptiert `groupby` eine Vergleichsprozedur mit der Signatur `(%b %b -> boolean)`, die zwei Gruppierungswerte auf Gleichheit überprüft.

`groupby` fasst Elemente der Eingangsliste, die auf den gleichen Gruppierungswert abgebildet werden, in einer Liste zusammen. Das Ergebnis von `groupby` ist die Liste der enstehenden Gruppen, also eine Liste von Listen. [7]

Beispiel: Eine Gruppierung von natürlichen Zahlen bezüglich ihres Rests bei ganzzahliger Division durch 3 kann wie folgt vorgenommen werden:

```
(check-expect (groupby (list 1 2 3 4 5 6)
                       (lambda (n) (modulo n 3))
                       =)
              (list (list 1 4) (list 2 5) (list 3 6) ))
```

Hinweise:

- **i)** Implementieren Sie *mindestens* zwei weitere Testfälle an!
- **ii)** Berechnen Sie zuerst die Liste der *unterschiedlichen* Gruppierungswerte. Im Beispiel ist dies die Liste `(list 1 2 0)`. Berechnen Sie anschließend für jeden dieser Gruppierungswerte die Liste derjenigen Elemente der Ursprungsliste, die auf diesen Gruppierungswert abgebildet werden.
- **iii)** Benutzen Sie die bereits implementierten bzw. bereitgestellten Prozeduren `map`, `distinct` und `filter`.

Abgabe: Programm `groupby.rkt`