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

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Informatik II

Tafel-Übungsblatt 2

Ausgabe: Do, 01.05.2014 - **Abgabe: Do, 08.05.2014 - 23:59 Uhr**

Informationen: **Die Abgabe muss in elektronischer Form erfolgen!**
d.h. Ihre Bearbeitung muss in den angegebenen Formaten bis vor Mitternacht ins CIS System hochgeladen werden.

Aufgaben [31]

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!

2.1 Mein Oder [3]

Ned Nerd sieht partout nicht ein, warum `or` eine Spezialform sein sollte (abgesehen von der Tatsache, dass `or` eine unbestimmte Anzahl von Argumenten nimmt). Er fragt sich, warum man `or` nicht als eine normale Prozedur definieren kann. Um das rauszufinden wickelt Ned ein `or` mit zwei Argumenten in eine selbstgeschriebene Prozedur ein. Das Ergebnis sieht so aus:

```
(define mein-oder
   (lambda (test-1 test-2)
      (if test-1
          #t
          test-2)))
```

Zu Neds großer Begeisterung funktioniert `mein-oder` anscheinend genauso wie das eingebaute `or` für zwei Argumente:

```
> (mein-oder (= 10 10) (> 2 5))
#t
> (mein-oder (> 23 42) (< 5 2))
#f
```

Neds Freundin Eva-Lu Ator ist skeptisch und meint, `mein-oder` verhielte sich anders als das eingebaute `or` für zwei Argumente. Wer von beiden hat Recht? Begründen Sie Ihre Entscheidung mit dem Substitutionsmodell und der Umformungsregel für `or`:

```
(or)                = #f
(or ⟨a1⟩ ... ⟨an⟩) = (if ⟨a1⟩
                          #t
                          (or ⟨a2⟩ ... ⟨an⟩))
```

Falls Eva-Lu Recht hat: Finden Sie ein Programm, an dem sich der Unterschied beobachten lässt.

Abgabe: als Racket-datei mit Kommentarblock: `mein-oder.rkt`

2.2 Kompakt [5]

Kürzen Sie das folgende Programm, in dem Sie überflüssigen Code entfernen oder logische Ausdrücke bzw. nicht notwendige Fallunterscheidungen zusammenfassen. Beachten Sie, dass die Signatur der Funktionen nicht verändert werden darf.

```
(: less-zero? (number -> boolean))
(define less-zero?
  (lambda (x)
    (if (not (< x 0))
        #f
        #t)))

(: f (number -> boolean))
(define f
  ((lambda (x) x)
   (lambda (y)
     (cond
       ((> y 11) #t)
       ((< y 11) #f)
       ((= y 11) #t)))))

(: g (boolean boolean -> boolean))
(define g
  (lambda (a b)
    (or (not b)
        (and a (not a)))))

(: greater-equal-zero? (number -> boolean))
(define greater-equal-zero?
  (lambda (x)
    (cond
      ((>= x 0) #t)
      (else #f))))
```

Abgabe: Programm `kompakt.rkt`

2.3 Fussball [3]

Beim Fußball lässt die Rückennummer eines Spieler häufig Rückschlüsse auf die Position des Spielers zu. Wir machen dabei folgende Annahmen:

- Ein *Torwart* hat die Rückennummer 1.
- Ein *Abwehrspieler* hat die Rückennummer 2, 3, 4 oder 5.
- Ein *Mittelfeldspieler* hat die Rückennummer 6, 7, 8 oder 10.
- Ein *Stürmer* hat die Rückennummer 9 oder 11.
- Ein *Ersatzspieler* hat eine Rückennummer zwischen 12 und 99.
- Alle anderen Rückennummern sind ungültig.

Schreiben Sie nun eine Prozedur mit folgender Signatur:
`(: natural->position (natural -> (one-of "Torwart" "Abwehr" "Mittelfeld" "Sturm" "Ersatz" "Ungueltig"))`
Die Prozedur soll dabei zu einer gegebenen Rückennummer die zugehörige Position berechnen.
Verwenden Sie beim Schreiben der Prozedur die Konstruktionsanleitungen für Prozeduren und für Fallunterscheidungen. Testen Sie die Prozedur `natural->position`, mit mindestens sechs Testfällen, so dass für jede Position (auch ungültige Positionen) ein Testfall existiert und damit alle Fälle abgedeckt sind.

Abgabe: Programm `fussball.rkt`

2.4 Clamp-It [4]

Schreiben Sie eine Prozedur `minimum`, die als Argumente zwei Zahlen nimmt und die kleinere der beiden Zahlen zurückgibt. Schreiben Sie analog dazu eine Prozedur `maximum`, die die größere der beiden Zahlen zurückgibt. Die bereits eingebauten Prozeduren `min` und `max` sollen hier bitte nicht verwendet werden.
Schreiben Sie nun mit diesen beiden Prozeduren eine dritte Prozedur `clamp`, die als Argumente drei Zahlen `x, untergrenze, obergrenze` bekommt. Die Prozedur soll `x` beschränkt auf das Intervall `[untergrenze, obergrenze]` zurückgeben. D.h. wenn `x` in diesem Intervall liegt, wird es unverändert zurückgegeben, sonst wird die naheliegendere Intervallgrenze zurückgegeben. Formal bedeutet das:

$$\texttt{clamp}\,(\texttt{x}, \texttt{untergrenze}, \texttt{obergrenze}) = \begin{cases} \texttt{untergrenze} & \text{falls } \texttt{x} < \texttt{untergrenze} \\ \texttt{obergrenze} & \text{falls } \texttt{x} > \texttt{obergrenze} \\ \texttt{x} & \text{sonst} \end{cases}$$

Abgabe: Programm `clamp.rkt`

2.5 Kalender [5]

Schreiben Sie eine Record-Definition für *Kalenderdaten* (bestehend aus Tag, Monat und Jahr).

a) Schreiben Sie eine Prozedur `calendar-date-ok?`, die feststellt ob ein Kalenderdatum-Record einem tatsächlichen Kalenderdatum entspricht, also korrekte Daten wie 1.1.1970 von unsinnigen wie 34.17.2006 unterscheidet. Lassen Sie dazu Schaltjahre außer acht. Abstrahieren Sie in geeigneter Weise, um die Korrektheit des Tages und des Monats in geeigneter Weise zu überprüfen.

b) (Bonus-Aufgabe, 2 Punkte) Berücksichtigen Sie Schaltjahre!

Hinweis: Zur Lösung der Aufgabe kann die eingebaute Prozedur

```
(: modulo (integer integer -> integer))
```

hilfreich sein. Sie bestimmt den Rest einer ganzzahligen Division.

Abgabe: Programm `calendar.rkt`

2.6 Kreaturen [11]

Auf einer Reise um die Welt triffst Du einen interessanten Zeitgenossen: Dr. Krosskretur, ein Experte auf dem Gebiet der Kreuzung von mystischen Kreaturen. Auf seiner Forschungsinsel gibt es drei klassische Grundkreaturen:

- Der Garnolaf, der Stärke besitzt.
- Das Ronugor, das Wissen besitzt.
- Die Tschipotol, die Risikobereitschaft besitzt.

Die Merkmale der Kreaturen sind unterschiedlich ausgeprägt. Die Krosskretur-Kreaturenmerkmal-Skala geht von 0 bis 100. Die Grundkreaturen haben jeweils nur ein Merkmal, keine besitzt ein Merkmal einer anderen Grundkreatur.
Nun kreuzt Dr. Krosskretur die Grundkreaturen miteinander. Es entstehen also

- Das Ronulaf, mit Wissen und Stärke.
- Der Tschigor, mit Wissen und Risikobereitschaft.
- Die Gapotol, mit Stärke und Risikobereitschaft.
- Das Tschigaronu, mit Wissen, Stärke und Risikobereitschaft.

Leider erben die Kreuzungen nicht die vollen Merkmale beider Grundkreaturen. So wird bei der Kreuzung von Tschipotol mit Ronugor das übernommene Wissen um 10% verringert aber die Risikobereitschaft bleibt gleich; bei Garnolaf mit Tschipotol nimmt die Risikobereitschaft um 5% ab, aber die Stärke legt um 8% zu; bei der Ronugor-Garnolaf Kreuzung lässt die Stärke um 5% nach, das Wissen bleibt aber erhalten. Kreuzt man alle drei Grundkreaturen, nimmt jede Eigenschaft um 3% ab.
Werden zwei Grundkreaturen der gleichen Sorte gekreuzt, so entsteht eine neue Grundkreatur der gleichen Sorte mit $\frac{2}{3}$ der addierten Eigenschaften der beiden Grundkreaturen. Man kann nur Grundkreaturen kreuzen.
Dr. Krosskretur braucht ein Programm, das die Eigenschaften der neuen Kreatur berechnet, bevor er die Kreuzungen durchführt. Helfen Sie ihm dabei!

a) Machen Sie eine Datenanalyse und erstellen Sie passende Rekorddefinitionen. Geben Sie auch die Signaturen der Rekord-Prozeduren an (die Signaturen der Selektoren sind nicht erforderlich)! [3]

b) Schreiben Sie für jede der oben aufgelisteten Kreuzungen von zwei Grundkreaturen eine Prozedur, die die Kreuzung vornimmt und eine neue Kreatur zurückgibt. [3]

c) Schreiben Sie auch für die dreier Kreuzung eine Prozedur, die drei Grundkreaturen in der Reihenfolge Tschipotol, Garnolaf, Ronugor akzeptiert, die dreifach Kreuzung vornimmt und die neue Kreatur zurückgibt. [2]

d) Schreiben Sie nun eine Prozedur, die zwei Grundkreaturen in beliebiger Reihenfolge akzeptiert, die Kreuzung vornimmt und eine neue Kreatur zurückgibt. Benutzen Sie dazu die Prozeduren aus den vorherigen Teilaufgabe. [3]

Abgabe: Programm `kreaturen.rkt`