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

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

# Informatik II

## Tafel-Übungsblatt 1

Ausgabe: Do, 24.04.2014 - **Abgabe: Do, 01.05.2014 - 23:59 Uhr**

Informationen: Abgabe dieses Übungsblatts ist (siehe auch oben): **Donnerstag, 1.5.14**.
**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 [35]

**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!

### 1.1 Mitternachtsformel [4]

Quadratische Gleichungen der Form $ax^2 + bx + c = 0$ können mit der sogenannten *Mitternachtsformel* gelöst werden. Die Mitternachtsformel liefert zwei Ergebnisse:

$$x_{1,2} = \frac{-b \pm \sqrt{D}}{2a} \tag{1}$$

mit Diskriminante $D = b^2 - 4ac$.
Zur Vereinfachung können Sie davon ausgehen, dass D $> 0$ gilt.

**a)** Programmieren Sie eine Prozedur `discriminant` mit drei Parametern für $a$, $b$ und $c$, welche die Diskriminante berechnet. Erkennen und abstrahieren Sie weitere Teilprobleme! [1]

**b)** Programmieren Sie eine Prozedur `mitternacht-1` mit drei Parametern für $a$, $b$ und $c$, die die erste Lösung der Mitternachtsformel, also x1, berechnet. Benutzen Sie dabei die Prozedur discriminant aus der vorherigen Teilaufgabe. [1]

**c)** Programmieren Sie eine Prozedur `mitternacht-2` mit drei Parametern für $a$, $b$ und $c$, die die andere Lösung der Mitternachtsformel, also x2, berechnet. Benutzen Sie dabei die Prozedur discriminant aus der vorherigen Teilaufgabe. [1]

**d)** Programmieren Sie eine Prozedur `mitternacht-out` mit drei Parametern für $a$, $b$ und $c$, welche die beiden Lösungen mithilfe der vorigen Prozeduren ausgibt. Zum Beispiel für $a = 1$, $b = 0$ und $c = -1$: "Lösung ist (1, -1)". [1]

**Hinweise:**

- Zur Lösung können Sie die eingebaute Prozedur `sqrt : (number -> number)` verwenden: `(sqrt x)` liefert die Quadratwurzel von $x$.

- Um einen String aus einer Zahl zu erstellen benutzen Sie die Prozedur `number->string : (number -> string)`.
- Um einen String aus mehreren, kürzeren Strings zu erzeugen, benutzen Sie die Prozedur `string-append (string string ... -> string)`.

Abgabe: Programm `mitternacht.rkt`

## 1.2 Benzinverbrauch Umrechnen [9]

In den USA und in Europa gibt es unterschiedliche Maße für die Energieeffizienz von Kraftfahrzeugen:

- In Europa ist das gängige Maß der *Verbrauch* in Liter pro 100km (l/100km);
- in den USA ist das gängige Maß die *Reichweite* in Meilen pro Gallone (mi/gal).

Schreiben Sie Prozeduren, die zwischen beiden Maßeinheiten umrechnen. Gehen Sie dazu wie folgt vor: Halten Sie sich bei jeder Prozedur, die Sie schreiben, an die Konstruktionsanleitung.

**a)** Schreiben Sie eine Prozedur `liters-per-100-km`, die eine Menge Benzin in Liter und die Reichweite dieses Benzins in Kilometer akzeptiert und daraus den Verbrauch in Liter pro 100km berechnet. [1]

**b)** Schreiben Sie eine Prozedur `miles-per-gal`, die eine Entfernung in Meilen und den Benzinverbrauch auf diese Entfernung in Gallonen akzeptiert und daraus die Reichweite in Meilen pro Gallone berechnet. [1]

**c)** Definieren Sie eine Konstante `km-per-mile` (eine US-Meile entspricht etwa $1,61$ Kilometer) und schreiben Sie zwei Prozeduren `kms->miles` und `miles->kms`, die jeweils eine Entfernung in der einen Maßeinheit akzeptieren und die Entfernung in die jeweils andere Maßeinheit umrechnen. [1.5]

**d)** Definieren Sie eine Konstante `liters-per-gal` (eine Gallone entspricht etwa $3,79$ Liter) und schreiben Sie zwei Prozeduren `liters->gal` und `gal->liters`, die jeweils eine Menge in einer Maßeinheit akzeptieren und die Menge in die jeweils andere Maßeinheit umrechnen. [1.5]

**e)** Schreiben Sie die Prozedur `l/100km->mi/gal`, die einen Verbrauch in Liter pro 100km akzeptiert und in die Reichweite in Meilen pro Gallone umrechnet. Benutzen Sie dafür die Prozeduren, die Sie in den anderen Teilaufgaben erstellt haben. Sollten Sie auf weitere Teilprobleme stoßen, abstrahieren Sie diese Teilprobleme in geeignete Prozeduren. [2]

**f)** Schreiben Sie die Prozedur `mi/gal->l/100km`, die eine Reichweite in Meilen pro Gallone akzeptiert und in den Verbrauch in Liter pro 100km umrechnet. Benutzen Sie dafür die Prozeduren, die Sie in den anderen Teilaufgaben erstellt haben. Sollten Sie auf weitere Teilprobleme stoßen, abstrahieren Sie diese Teilprobleme in eigene Prozeduren. [2]

Abgabe: Programm `benzinverbrauch.rkt`

## 1.3 Reduktion Verstehen [14]

Beobachten Sie die Auswertung der folgenden Ausdrücke im Stepper von DrRacket und geben Sie die Reduktionsregeln an, die der Stepper ausführt!
Um Schreibarbeit zu sparen, ist es sinnvoll, die Regeln des Substitutionsmodells wie folgt ab zu kürzen:

| | |
|---|---|
| $\text{eval}_{\text{id}}$ | Auswertung einer Variablen |
| $\text{apply}_{\text{prim}(f)}$ | Applikation einer eingebauten Funktion $f$ |
| $\text{apply}_{\lambda}$ | Applikation einer `lambda`-Abstraktion |

Für den Ausdruck `(+ (* 1 2) 3)` zeigt der Stepper zum Beispiel `(+ 2 3)` als erstes Zwischenergebnis an. In diesem Schritt kommt die Regel $\text{apply}_{\text{prim}(*)}$ (für die Applikation der eingebauten Prozedur `*`) zur Anwendung.
Benennen Sie für die folgenden Ausdrücke für jeden Schritt des Steppers die Reduktionsregel, die der Stepper angewendet hat. Halten Sie dabei die einzelnen Zwischenergebnisse fest.

**Bemerkung:** Bei der Auswertung eingebauter Funktionen kann direkt der $\text{apply}_{\text{prim}(f)}$-Schritt spezifiziert werden (der im Stepper nicht gezeigte Zwischenschritt $\text{eval}_{\text{id}} f$ kann ignoriert werden).

```
((lambda (a) (* a 6))
 (+ ((lambda (a) (+ a 2)) 3) 2))

(define pi 3.141)
(define 2pi (* 2 pi))

((lambda (x) (* 2 x)) pi)

(define quadrat (lambda (x) (* x x)))
(define area
  (lambda (x)
    (* (quadrat x) pi)))

(area 2)

((lambda (y) y) (lambda z) z))
```

Abgabe: als Textdatei `stepper.txt`

## 1.4 Namensbindung [8]

Betrachten Sie folgendes *tippfehlerfreies* Programm:

```
(define frage 23)
(define antwort 42)

(define allwissender
  (lambda (antwort frage allwissender)
    ((lambda (antwort)
       (lambda (frrage)
         (lambda (frage)
           (lambda (frage)
             (lambda (frage)
               (allwissender (- antwort frage)))))))
     (* antwort frage))))

(((((allwissender frage antwort (lambda (allwissender) allwissender))
    (- antwort frage))
   frage)
  antwort)
 frage)
```

**a)** Benutzen Sie das Prinzip der *lexikalischen Bindung* um zu erklären, warum das Ergebnis des obigen Programms `943` ist. Erklären Sie also insbesondere Schritt für Schritt, was während der Auswertung wie reduziert wird. [5]

**b)** Ändern Sie in dem Programm die Variablennamen unter Erhaltung der Bedeutung des Programms *und begründen Sie die Korrektheit Ihrer Lösung.* Und zwar...

- einmal so, dass eine minimale Anzahl von Namen verwendet wird; [1]
- und einmal so, dass eine maximale Anzahl von Namen verwendet wird. [2]

Abgabe: als Textdatei `allwissender.txt`