TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Lehrstuhl für Sprachen und Beschreibungsstrukturen
Einführung in die Informatik 2
Prof. Dr. Helmut Seidl, T.M. Gawlitza, S. Pott

WS 2008/09
Übungsblatt 5
11.11.2008

Abgabe: 18.11.2008 (vor der Vorlesung)

Aufgabe 5.1 (H) Tripel

a) Gegeben sei folgende MiniJava-Prozedur:

```
void f() {
  z = x;
  x = y;
  y = z;
}
```

Welche der folgenden Tripel sind gültig. Begründen Sie Ihre Antworten.

i) $\{\mathtt{x} = l_\mathtt{x} \wedge \mathtt{y} = l_\mathtt{y} \wedge l_\mathtt{x} \leq l_\mathtt{y}\} \quad \mathtt{f}(); \quad \{\mathtt{x} = l_\mathtt{y} \wedge \mathtt{y} = l_\mathtt{x} \wedge l_\mathtt{y} \leq l_\mathtt{x}\}$

ii) $\{\mathtt{x} = l_\mathtt{x} \wedge \mathtt{y} = l_\mathtt{y} \wedge l_\mathtt{x} \leq l_\mathtt{y}\} \quad \mathtt{f}(); \quad \{\mathtt{x} = l_\mathtt{y} \wedge \mathtt{y} = l_\mathtt{x} \wedge l_\mathtt{x} \leq l_\mathtt{y}\}$

iii) $\{\mathtt{x} = l_\mathtt{x} \wedge \mathtt{y} = l_\mathtt{y} \wedge l_\mathtt{x} \leq l_\mathtt{y}\} \quad \mathtt{f}(); \quad \{\mathtt{x} = l_\mathtt{x} \wedge \mathtt{y} = l_\mathtt{y} \wedge l_\mathtt{y} \leq l_\mathtt{x}\}$

b) Geben Sie eine Prozedur $\mathtt{f}()$ an, für die das Tripel

$$\{\text{false}\} \quad \mathtt{f}(); \quad \{\text{true}\}$$

gültig ist. Begründen Sie Ihre Antwort.

c) Im folgenden seien $\mathtt{x}, \mathtt{y}, \mathtt{z}, \mathtt{n}$ globale Programmvariablen. Nehmen Sie an, dass folgendes Tripel für die nicht näher spezifizierte Prozedur $\mathtt{f}()$ gültig sei:

$$\{\mathtt{x} = l_\mathtt{x} \wedge \mathtt{y} = l_\mathtt{y} \wedge |l_\mathtt{x}| \leq 2^{l_\mathtt{y}} \wedge l_\mathtt{y} \geq 0\} \quad \mathtt{f}(); \quad \{\mathtt{x} = 0 \wedge \mathtt{z} = l_\mathtt{x} \cdot l_\mathtt{y} + 42\} \tag{1}$$

Welche der folgenden Tripel sind unter dieser Annahme gültig. Begründen Sie Ihre Antworten.

Für ein Tripel dessen Gültigkeit nicht durch die Gültigkeit des Tripels in (1) impliziert wird, ist ein Gegenbeispiel anzugeben. D.h. es ist eine Prozedur $\mathtt{f}()$ anzugeben für die das Tripel in (1) erfüllt ist, aber das Tripel in der entsprechenden Teilaufgabe nicht erfüllt ist. Um zu zeigen, dass ein Tripel nicht gilt sind lediglich Werte für $\mathtt{x}$ und $\mathtt{y}$ zu benennen, für die das Tripel nicht gilt.

i) $\{\mathtt{x} = l_\mathtt{x} \wedge \mathtt{y} = l_\mathtt{y} \wedge |l_\mathtt{x}| \leq 2^{l_\mathtt{y}} \wedge l_\mathtt{y} \geq 0\} \quad \mathtt{f}(); \quad \{\mathtt{y} = l_\mathtt{y}\}$

ii) $\{\mathtt{x} = l_\mathtt{x} + 2 \cdot l_\mathtt{y} \wedge \mathtt{y} = l_\mathtt{y} \wedge |l_\mathtt{x} + 2 \cdot l_\mathtt{y}| \leq 2^{l_\mathtt{y}} \wedge l_\mathtt{y} \geq 0\} \quad \mathtt{f}(); \quad \{\mathtt{x} = 0 \wedge \mathtt{z} = l_\mathtt{x} \cdot l_\mathtt{y} + 2 \cdot l_\mathtt{y} \cdot l_\mathtt{y} + 42\}$

iii) $\{\mathtt{x} = l_\mathtt{x} \wedge \mathtt{y} = l_\mathtt{y} \wedge |l_\mathtt{x}| \leq 2^{l_\mathtt{y}} \wedge l_\mathtt{y} \geq 0 \wedge \mathtt{n} \leq 5\} \quad \mathtt{f}(); \quad \{\mathtt{x} = 0 \wedge \mathtt{z} = l_\mathtt{x} \cdot l_\mathtt{y} + 42 \wedge \mathtt{n} \leq 5\}$

iv) $\{\mathtt{x} = l_\mathtt{x} \wedge \mathtt{y} = 2 \cdot l_\mathtt{y} \wedge |l_\mathtt{x}| = 4^{l_\mathtt{y}} \wedge 2 \cdot l_\mathtt{y} \geq 0\} \quad \mathtt{f}(); \quad \{\mathtt{x} \leq 0 \wedge \mathtt{z} = 2 \cdot l_\mathtt{x} \cdot l_\mathtt{y} + 42\}$

Lösungsvorschlag 5.1

a) i) Dieses Tripel ist *nicht* gültig. Die logischen Variablen $l_{\mathtt{x}}$ und $l_{\mathtt{y}}$ verändern ihren Wert nicht. Wenn vor der Prozedurausführung $l_{\mathtt{x}} \leq l_{\mathtt{y}}$ galt, muss es danach immer noch gelten.

ii) Die Prozedur tauscht die Werte der Variablen `x` und `y`. Dieses Tripel ist also gültig.

iii) Dieses Tripel ist *nicht* gültig. Die logischen Variablen $l_{\mathtt{x}}$ und $l_{\mathtt{y}}$ verändern ihren Wert nicht. Wenn vor der Prozedurausführung $l_{\mathtt{x}} \leq l_{\mathtt{y}}$ galt, muss es danach immer noch gelten.

b) Jedes Programm erfüllt dieses Tripel.

c) i) Dieses Tripel ist *nicht* gültig. Das vorgegebene Tripel in (1) macht keine Aussage über y nach der Ausführung von `f()`. Gegenbeispiel:

```
void f() {
  z = x * y + 42;
  x = 0;
  y = y + 5;
}
```

ii) Hier wurde $l_{\mathtt{x}}$ durch $l_{\mathtt{x}} + 2 \cdot l_{\mathtt{y}}$ substituiert. Dieses Tripel ist somit gültig.

iii) Das Tripel ist nicht gültig. Gegenbeispiel:

```
void f() {
  z = x * y + 42;
  x = 0;
  n = 10;
}
```

iv) Dieses Tripel ist gültig. Zum einen wurde l_y durch $2 \cdot l_{\mathtt{y}}$ substituiert und zum anderen wurde die Nachbedingung abgeschwächt ($\mathtt{x} \leq 0$).

Aufgabe 5.2 (H)

Gegeben sei folgende MiniJava-Prozedur:

```
void f() {
  while (y > 0) {
    x = x * x;
    y = y - 1;
  }
}
```

Geben Sie ein möglichst aussagekräftiges gültiges Tripel $\{\mathtt{A}\}\ \mathtt{f}();\ \{\mathtt{B}\}$ für die Prozedur `f()` an und zeigen Sie dessen Gültigkeit.

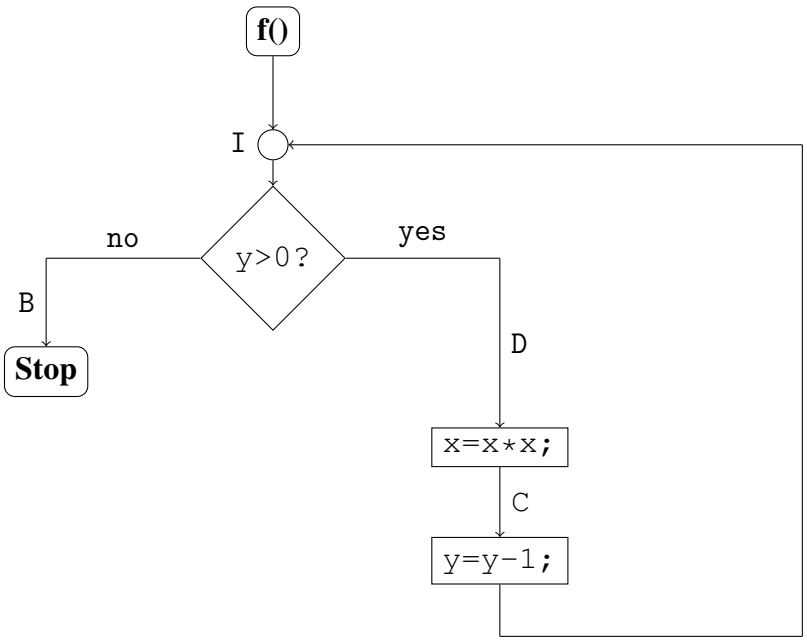
Lösungsvorschlag 5.2

Wir setzen

$$\mathtt{A} :\equiv \mathtt{x} = m \wedge \mathtt{y} = n$$
$$\mathtt{B} :\equiv (n \geq 0 \wedge \mathtt{x} = m^{2^n} \wedge \mathtt{y} = 0) \vee (n < 0 \wedge \mathtt{x} = m \wedge \mathtt{y} = n)$$

Dabei bezeichnen m und n logische Variablen.

Wir erstellen zunächst das Kontrollfluß-Diagramm:



Wir raten die Schleifen-Invariante

$$\mathtt{I} :\equiv (n \geq 0 \wedge \mathtt{x} = m^{2^{n-\mathtt{y}}} \wedge \mathtt{y} \geq 0) \vee (n < 0 \wedge \mathtt{x} = m \wedge \mathtt{y} = n)$$

und stellen erleichtert fest, dass

$$\mathtt{A} \Rightarrow \mathtt{I}$$

gilt. Es ergibt sich:

$$\begin{aligned}\mathbf{WP}[\![\mathtt{x} = \mathtt{x} * \mathtt{x}; \mathtt{y} = \mathtt{y} - 1;]\!](I) &:\equiv (n \geq 0 \wedge \mathtt{x}^2 = m^{2^{n-\mathtt{y}+1}} \wedge \mathtt{y} - 1 \geq 0) \vee (n < 0 \wedge \mathtt{x}^2 = m \wedge \mathtt{y} - 1 = n) \\ &\Leftarrow (n \geq 0 \wedge \mathtt{x} = m^{2^{n-\mathtt{y}}} \wedge \mathtt{y} > 0) \vee (n < 0 \wedge \mathtt{x}^2 = m \wedge \mathtt{y} - 1 = n) \\ &\equiv: \mathtt{D}\end{aligned}$$

$$\begin{aligned}\mathbf{WP}[\![\mathtt{y} > 0]\!](B, D) &:\equiv (\mathtt{y} \leq 0 \wedge ((n \geq 0 \wedge \mathtt{x} = m^{2^n} \wedge \mathtt{y} = 0) \vee (n < 0 \wedge \mathtt{x} = m \wedge \mathtt{y} = n))) \\ &\quad \vee (\mathtt{y} > 0 \wedge ((n \geq 0 \wedge \mathtt{x} = m^{2^{n-\mathtt{y}}} \wedge \mathtt{y} > 0) \vee (n < 0 \wedge \mathtt{x} = m \wedge \mathtt{y} = n))) \\ &\equiv (\mathtt{y} \leq 0 \wedge ((n \geq 0 \wedge \mathtt{x} = m^{2^{n-\mathtt{y}}} \wedge \mathtt{y} \geq 0) \vee (n < 0 \wedge \mathtt{x} = m \wedge \mathtt{y} = n))) \\ &\quad \vee (\mathtt{y} > 0 \wedge ((n \geq 0 \wedge \mathtt{x} = m^{2^{n-\mathtt{y}}} \wedge \mathtt{y} \geq 0) \vee (n < 0 \wedge \mathtt{x} = m \wedge \mathtt{y} = n))) \\ &\equiv (\mathtt{y} \leq 0 \wedge I) \vee (\mathtt{y} > 0 \wedge I) \\ &\equiv (\mathtt{y} \leq 0 \vee y > 0) \wedge I \\ &\equiv I\end{aligned}$$

Damit ist die Korrektheit gezeigt.

Aufgabe 5.3 (P) Rekursive Prozeduren

Gegeben sei folgendes MiniJava-Programm:

```
int n,m,r;

void f() {
  if (n >= m && m >= 0) {
    if (m == 0)
      r = r + 1;
```

```
    else {
      n = n - 1;
      m = m - 1;
      f();
      m = m + 1;
      f();
      n = n + 1;
    }
  }
}
```

a) Erstellen Sie das Kontrollfluss-Diagramm für die Prozedur `f()`!

b) Zeigen Sie die Gültigkeit des Tripels

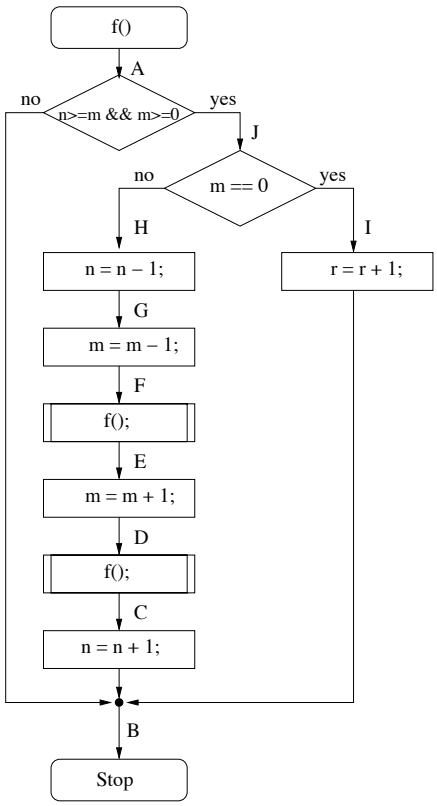
$$\{\mathtt{r} = l_{\mathtt{r}} \wedge \mathtt{m} = l_{\mathtt{m}} \wedge \mathtt{n} = l_{\mathtt{n}}\} \quad \mathtt{f}(); \quad \{\mathtt{r} = l_{\mathtt{r}} + \binom{l_{\mathtt{n}}}{l_{\mathtt{m}}} \wedge \mathtt{m} = l_{\mathtt{m}} \wedge \mathtt{n} = l_{\mathtt{n}}\}.$$

Als Hilfestellung sei an die folgende Identität erinnert:

$$\binom{n}{m} = \begin{cases} 1 & \text{falls } m = 0,\ n \geq 0 \\ \binom{n-1}{m-1} + \binom{n-1}{m} & \text{falls } n \geq m \geq 1 \\ 0 & \text{sonst} \end{cases}, \qquad m, n \in \mathbb{Z}.$$

Lösungsvorschlag 5.3

a) Das Kontrollfluss-Diagramm für die Prozedur `f()`:



b) **Verifikation der Prozedur** `f()`**.** Wir setzen

$$A \;:\equiv\; \mathtt{r} = l_\mathtt{r} \wedge \mathtt{m} = l_\mathtt{m} \wedge \mathtt{n} = l_\mathtt{n} \tag{2}$$

$$B \;:\equiv\; \mathtt{r} = l_\mathtt{r} + \binom{l_\mathtt{n}}{l_\mathtt{m}} \wedge \mathtt{m} = l_\mathtt{m} \wedge \mathtt{n} = l_\mathtt{n} \tag{3}$$

und setzen also die Gültigkeit des Tripels $\{A\}\ \mathtt{f}();\ \{B\}$ voraus. Es ergibt sich:

$$\begin{aligned}
\mathbf{WP}[\![n = n + 1;]\!](B) \;&\equiv\; \mathtt{r} = l_\mathtt{r} + \binom{l_\mathtt{n}}{l_\mathtt{m}} \wedge \mathtt{m} = l_\mathtt{m} \wedge \mathtt{n} + 1 = l_\mathtt{n} \\
&\Leftarrow\; \mathtt{r} = l_\mathtt{r} + \binom{l_\mathtt{n}}{l_\mathtt{m}} \wedge \mathtt{m} = l_\mathtt{m} \wedge \mathtt{n} = l_\mathtt{n} - 1 \\
&\quad\; \wedge l_\mathtt{n} \geq l_\mathtt{m} \geq 1 \\
&\equiv\; \mathtt{r} = l_\mathtt{r} + \binom{l_\mathtt{n} - 1}{l_\mathtt{m} - 1} + \binom{l_\mathtt{n} - 1}{l_\mathtt{m}} \wedge \mathtt{m} = l_\mathtt{m} \wedge \mathtt{n} = l_\mathtt{n} - 1 \\
&\quad\; \wedge l_\mathtt{n} \geq l_\mathtt{m} \geq 1 \\
&\equiv:\; C
\end{aligned}$$

Wir setzen

$$D \;:\equiv\; \mathtt{r} = l_\mathtt{r} + \binom{l_\mathtt{n} - 1}{l_\mathtt{m} - 1} \wedge \mathtt{m} = l_\mathtt{m} \wedge \mathtt{n} = l_\mathtt{n} - 1 \wedge l_\mathtt{n} \geq l_\mathtt{m} \geq 1 \tag{4}$$

Die Gültigkeit des Tripels $\{D\}\ \ \mathtt{f}();\ \ \{C\}$ ergibt sich aus der Gültigkeit des Tripels $\{A\}\ \mathtt{f}();\ \{B\}$, da

$$D \equiv A[(l_\mathtt{n} - 1)/l_\mathtt{n},\ (l_\mathtt{r} + \binom{l_\mathtt{n} - 1}{l_\mathtt{m} - 1})/l_\mathtt{r}] \wedge l_\mathtt{n} \geq l_\mathtt{m} \geq 1$$

und

$$C \equiv B[(l_\mathtt{n} - 1)/l_\mathtt{n},\ (l_\mathtt{r} + \binom{l_\mathtt{n} - 1}{l_\mathtt{m} - 1})/l_\mathtt{r}] \wedge l_\mathtt{n} \geq l_\mathtt{m} \geq 1.$$

Das Tripel ergibt sich also durch Substitution logischer Variablen und Hinzufügen der Bedingung $l_\mathtt{n} \geq l_\mathtt{m} \geq 1$, die nur über logische Variablen spricht. (Spezialfall der Adaptions-Regel aus der Vorlesung.)

$$\begin{aligned}
\mathbf{WP}[\![m = m + 1;]\!](D) \;&\equiv\; \mathtt{r} = l_\mathtt{r} + \binom{l_\mathtt{n} - 1}{l_\mathtt{m} - 1} \wedge \mathtt{m} = l_\mathtt{m} - 1 \wedge \mathtt{n} = l_\mathtt{n} - 1 \\
&\quad\; \wedge l_\mathtt{n} \geq l_\mathtt{m} \geq 1 \\
&\equiv:\; E
\end{aligned}$$

Analog zum anderen Aufruf von $\mathtt{f}()$ setzen wir:

$$F \;:\equiv\; \mathtt{r} = l_\mathtt{r} \wedge \mathtt{m} = l_\mathtt{m} - 1 \wedge \mathtt{n} = l_\mathtt{n} - 1 \wedge l_\mathtt{n} \geq l_\mathtt{m} \geq 1 \tag{5}$$

Aus der Gültigkeit des Tripels $\{A\}\ \mathtt{f}();\ \{B\}$ ergibt sich wiederum die Gültigkeit des Tripels $\{F\}\ \mathtt{f}();\ \{E\}$.

$$\begin{aligned}
\mathbf{WP}[\![m = m - 1]\!](F) \;&\equiv\; \mathtt{r} = l_\mathtt{r} \wedge \mathtt{m} - 1 = l_\mathtt{m} - 1 \wedge \mathtt{n} = l_\mathtt{n} - 1 \wedge l_\mathtt{n} \geq l_\mathtt{m} \geq 1 \\
&\equiv\; \mathtt{r} = l_\mathtt{r} \wedge \mathtt{m} = l_\mathtt{m} \wedge \mathtt{n} = l_\mathtt{n} - 1 \wedge l_\mathtt{n} \geq l_\mathtt{m} \geq 1 \;\equiv:\; G
\end{aligned}$$

$$\begin{aligned}\mathbf{WP}[\![n = n - 1]\!](G) &\equiv \mathtt{r} = l_{\mathtt{r}} \wedge \mathtt{m} = l_{\mathtt{m}} \wedge \mathtt{n} - 1 = l_{\mathtt{n}} - 1 \wedge l_{\mathtt{n}} \geq l_{\mathtt{m}} \geq 1 \\ &\equiv \mathtt{r} = l_{\mathtt{r}} \wedge \mathtt{m} = l_{\mathtt{m}} \wedge \mathtt{n} = l_{\mathtt{n}} \wedge l_{\mathtt{n}} \geq l_{\mathtt{m}} \geq 1 \equiv: H\end{aligned}$$

$$\mathbf{WP}[\![r = r + 1]\!](B) \equiv \mathtt{r} + 1 = l_{\mathtt{r}} + \binom{l_{\mathtt{n}}}{l_{\mathtt{m}}} \wedge \mathtt{m} = l_{\mathtt{m}} \wedge \mathtt{n} = l_{\mathtt{n}} \equiv: I$$

Wir setzen

$$J :\equiv \mathtt{r} = l_{\mathtt{r}} \wedge \mathtt{m} = l_{\mathtt{m}} \wedge \mathtt{n} = l_{\mathtt{n}} \wedge l_{\mathtt{n}} \geq l_{\mathtt{m}} \geq 0$$

und zeigen, dass

$$J \implies \mathbf{WP}[\![m == 0]\!](H, I) \equiv (\mathtt{m} \neq 0 \wedge H) \vee (\mathtt{m} = 0 \wedge I)$$

gilt. Dazu nehmen wir an, dass J gilt. Falls $\mathtt{m} \neq 0$ gilt offensichtlich H. Nehmen wir also an $\mathtt{m} = 0$. Dann ist $\binom{l_{\mathtt{n}}}{0} = 1$ und deshalb I erfüllt.

Schließlich bleibt zu überprüfen, ob A die Zusicherung

$$\mathbf{WP}[\![n >= m \;\&\&\; m >= 0]\!](B, J) \equiv ((\mathtt{n} < \mathtt{m} \vee \mathtt{m} < 0) \wedge B) \vee (\mathtt{n} \geq \mathtt{m} \wedge \mathtt{m} \geq 0 \wedge J)$$

impliziert. Nehmen wir also an, dass A gilt. Gilt $n \geq m \wedge m \geq 0$ nicht, so gilt $n < m$ oder $m < 0$. In beiden Fällen ist $\binom{l_{\mathtt{n}}}{l_{\mathtt{m}}} = 0$, so dass B trivialer Weise impliziert wird. Im Falle $n \geq m \wedge m \geq 0$ wird J trivialer Weise impliziert.

Aufgabe 5.4 (P) Was ist gleich?

Alternative a	Alternative b	gleich	nicht gleich
int -> (int -> int)	(int -> int) -> int	☐	☐
(int -> int) -> int	int -> int -> int	☐	☐
int -> (int -> int)	int -> int -> int	☐	☐
a b c	(a b) c	☐	☐
a b c	a (b c)	☐	☐
x	((x))	☐	☐
x, y	(x, y)	☐	☐
1::2::3::[]	1::[2;3]::[]	☐	☐
1::2::3::[]	1::[2;3]	☐	☐

Lösungsvorschlag 5.4

Alternative a	Alternative b	gleich	nicht gleich
int -> (int -> int)	(int -> int) -> int	☐	☑
(int -> int) -> int	int -> int -> int	☐	☑
int -> (int -> int)	int -> int -> int	☑	☐
a b c	(a b) c	☑	☐
a b c	a (b c)	☐	☑
x	((x))	☑	☐
x, y	(x, y)	☑	☐
1::2::3::[]	1::[2;3]::[]	☐	☑
1::2::3::[]	1::[2;3]	☑	☐

Aufgabe 5.5 (P) OCaml vs. Java!

Übersetzen Sie die statischen Methoden der folgenden Java-Klasse möglichst genau in OCaml.

```
public class Java {
  public static int sign(int a) {
    if (a > 0)
      return 1;
    else if (a < 0)
      return -1;
    return 0;
  }

  public static int sum_n(int n) {
    if (n == 0)
      return 0;
    return n + sum_n(n - 1);
  }

  public static int sum_n_2(int n) {
    int erg = 0;
    while (n > 0) {
      erg = erg + n;
      n = n - 1;
```

```
    }
    return erg;
  }
}
```

Lösungsvorschlag 5.5

```
let sign a =
  if a > 0 then
    1
  else if a < 0 then
    - 1
  else
    0

let rec sum_n n =
  if n = 0 then
    0
  else
    n + (sum_n (n - 1))

let rec sum_n_2 erg n =
  if n > 0 then
    let erg = erg + n in
    let n = n - 1 in
    sum_n_2 erg n
  else
    erg

let sum_n_2 = sum_n_2 0
```

Aufgabe 5.6 (P) Finanzgeschäfte

In dieser Aufgabe soll die Situation in einer Bank betrachtet werden.

a) Definieren Sie einen Datentyp `wertpapier` zur Repräsentation von Wertpapieren, wie z.B. Aktien oder Zertifikate. Ein Wertpapier hat einen Namen und einen Wert.

b) Des Weiteren wird ein Datentyp `kunde` zur Repräsentation von Kunden benötigt. Jeder Kunde hat einen Namen und ein Portfolio, welches eine Liste von, aus einer Anzahl und einem Wertpapier bestehenden, Paaren ist.

c) Schreiben Sie eine Funktion, die den Gesamtwert eines Portfolios bestimmt.

d) Schreiben Sie eine Funktion, die das in Wertpapieren angelegte „Vermögen" eines Kunden ermittelt.

e) Schreiben Sie eine Funktion `anzahl : (kunde -> bool) -> kunde list -> int`, für die der Aufruf `anzahl p l` die Anzahl der Kunden in der Liste `l`, die das Prädikat `p` erfüllen, ergibt.

f) Schreiben Sie eine Funktion `filter : (kunde -> bool) -> kunde list -> kunde list`, für die der Aufruf `filter p l` die Liste der Kunden aus der Liste `l`, die das Prädikat `p` erfüllen, ergibt.

g) Schreiben Sie eine Funktion `grosskunden : kunde list -> kunde list`, für die der Aufruf `grosskunden l` die Liste derjenigen Kunden aus der Liste `l`, deren Portfolio mindestens einen Gesamtwert von 1000000 $ hat, ergibt.

Lösungsvorschlag 5.6

```
type wertpapier = { bez : string; wert : int }

type kunde = { name : string; portfolio : (int * wertpapier) list }

(* drei moegliche Loesungen fuer Teil c) *)

let rec portfoliowert portfolio =
  match portfolio with
      [] -> 0
    | (a,w)::ws -> a * w.wert + portfoliowert ws

let rec portfoliowert = function
    [] -> 0
  | (a,w)::ws -> a * w.wert + portfoliowert ws

let rec portfoliowert = List.fold_left (fun s (a,w) -> s + a * w.wert) 0

let kundenwert k = portfoliowert k.portfolio

let rec anzahl praedikat kunden_liste =
  match kunden_liste with
      [] -> 0
  | k::ks ->
      anzahl praedikat ks +
```

```
    if praedikat k then
      1
    else
       0

(* drei moegliche Loesungen fuer Teil f) *)

let rec filter praedikat kunden_liste =
  match kunden_liste with
      [] -> []
  | k::ks ->
    if praedikat k then
       k::filter praedikat ks
    else
       filter praedikat ks

let filter praedikat kunden_liste = List.filter praedikat kunden_liste

let filter = List.filter

let grosskunden = filter (fun k -> kundenwert k >= 1000000)
```

Erklärung zu den Lösungen:

a) Ein Beispiel für ein Wertpapier ist: `{bez="RWE";wert=66}`

b) Ein Beispiel für einen Kunden ist:
`{kunde="Seidl";portfolio=[(2,{bez="RWE";wert=66});`
`(911,{bez="PORSCHE";wert=60})]}`

c)
- Die erste Lösung der Funktion `portfoliowert` vergleicht den Parameter `portfolio` mit den pattern `[]` und `(a,w)`.
  Ein leeres Portfolio ist eine leere Liste `[]`. Es hat den Wert 0. Ansonsten besteht ein Protfolio aus einem ersten Element `(a,w)` und einer weiteren Liste `ws`. Das erste Element besteht aus einer Anzahl `a` und einem Wertpapier `w`. `w` ist also `{bez=w.bez; wert=w.wert}`. Für das erste Element wird der Wert berechnet `a * w.wert` und darauf der Gesamtwert (`portfoliowert`) der restlichen Liste `ws` addiert.
- Die zweite Lösung ist wie die erste, nur das hier das pattern matching über eine unbenannte Funktion definiert wird. Diese Funktion wird auf den Parameter `portfolio` angewendet und damit wird `portfolio` wie bei der ersten Lösung mit den pattern `[]` und `(a,w)` verglichen.
- Die dritte Lösung verwendet die Funktion `List.fold_left`. Diese nimmt eine Funktion mit zwei Parametern und wendet diese sukzessive links beginnend auf das bisherige Ergebnis und das jeweils nächste Element der Liste an und berechnet so schließlich ein Gesamtergebnis. Der erste Parameter der Funktion (hier `s`) akkumuliert die bisherigen Ergebnisse, der zweite ist das aktuelle Listenelement. `List.fold_left` benötigt neben dieser Funktion noch einen Startwert für `s`. In unserem Fall wird also zunächst die Funktion `(fun s (a,w) -> s + a * w.wert)` mit dem Wert `0` für `s` auf das erste (linke) Listenelement angewendet. Als nächstes wird sie auf das zweite Listenelement angewendet, wobei der Wert von `s` das Ergebnis des vorigen Aufrufes ist, also `0 + a1 * w1.wert`, wenn `(a1,w1)` das erste Listenelement war. Wurde die gesamte Liste durchlaufen wird das Ergebnis des letzten Aufrufs und damit der Gesamtwert des Portfolios zurückgegeben.

d) Das gesuchte Vermögen eines Kunden entspricht dem Gesamtwert seines Portfolios.

e) Diese Funktion ist wieder über pattern matching (s. c erste Lösung) realisiert. Falls die Kundenliste `kunden_liste` nicht leer ist (Fall `k::ks`) wird die Funktion rekursiv auf den Rest der Liste `ks` angewendet und auf das Ergebnis 1 oder 0 addiert, je nachdem ob der erste Kunde `k` das Prädikat `praedikat` erfüllt (1) oder nicht (0).

f)
- Die erste Lösung überprüft, ob die Kundenliste leer ist (`[]`) oder ein erstes Element hat (`k::ks`). Dieser erste Kunde `k` wird vor das Ergebnis des rekursiven Aufrufs `filter praedikat ks` gehängt, falls `k` das Prädikat `praedikat` erfüllt. Andernfalls wird nur die Funktion rekursiv auf den Rest der Liste `ks` angewendet.
- Die gesuchte Funktion entspricht der Funktion `List.filter`.
- Dabei müssen die Parameter nicht übergeben werden, da `List.filter` bereits den richtigen Typ hat.

g) Die Funktion `grosskunden` entspricht einer Anwendung der vorigen Funktion `filter`. Das Prädikat ist die Funktion `(fun k -> kundenwert k >= 1000000)`, also die Eigenschaft eines Kunden `k` ein Portfolio mit einem Gesamtwert (`kundenwert k`) von 1000000 $ oder mehr zu besitzen.