TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Lehrstuhl für Sprachen und Beschreibungsstrukturen WS 2008/09
Einführung in die Informatik 2 **Übungsblatt 13**
Prof. Dr. Helmut Seidl, T. M. Gawlitza, S. Pott, M. Schwarz 15.01.2009

Abgabe: keine

Aufgabe 13.1 (P) Verifikation funktionaler Programme

Gegeben sei folgende MiniOCaml-Funktion:

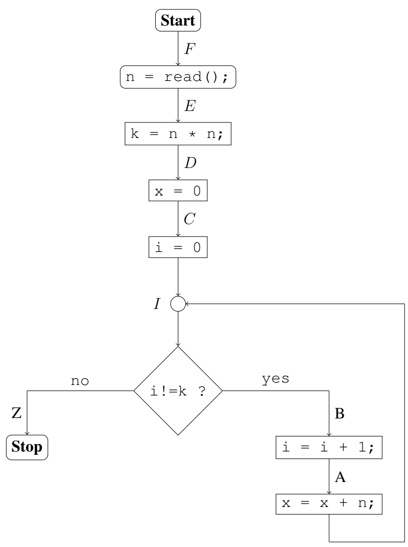
```
let rec f =
    fun a ->
        match a with
            (z,[]) -> []
          | (z,x::xs) -> (z + x)::(f (z,xs))
```

Zeigen Sie mit Hilfe der Big-Step operationellen Semantik, dass der Aufruf `f` $(7, 1)$ für jeden Listenwert `l` terminiert.

Hinweis: Eine Auflistung der Axiome und Regeln der Big-Step operationellen Semantik befindet sich im Anhang.

Aufgabe 13.2 (P) Verifikation

Gegeben sei folgendes Kontroll-Fluß-Diagramm:



a) Zeigen Sie, dass am **Stop**-Knoten die Zusicherung $Z \;:\equiv\; \mathtt{x} = \mathtt{n}^3$ stets erfüllt ist!

b) Führen Sie an geeigneten Stellen im Programm eine Kenngröße `r` ein, die es ermöglicht zu zeigen, dass das Programm stets terminiert. Den Beweis müssen Sie **nicht** führen.

Aufgabe 13.3 (P) Verifikation in Anwesenheit von Prozeduren

In dieser Aufgabe sind $\mathtt{x}, \mathtt{y}$ und $\mathtt{z}$ Programm-Variablen. $l_{\mathtt{x}}, l_{\mathtt{y}}, l_{\mathtt{z}}$ und l sind logische Variablen. Nehmen Sie an, dass folgendes Tripel für die nicht näher spezifizierte Prozedur $\mathtt{f}()$ gültig sei:

$$\{\mathtt{x} = l_{\mathtt{x}} \ \wedge \ \mathtt{y} = l_{\mathtt{y}} \ \wedge \ l_x \cdot l_{\mathtt{y}} \geq 0\} \quad \mathtt{f}(); \quad \{\mathtt{x} = l_{\mathtt{x}} \ \wedge \ \mathtt{y} = l_{\mathtt{x}} \cdot l_{\mathtt{y}}\} \tag{1}$$

a) Wie sind die Werte der Variablen $\mathtt{x}$ und $\mathtt{y}$ nach Ausführung der Prozedur $\mathtt{f}()$, wenn vor Ausführung $\mathtt{x}$ den Wert 2 und $\mathtt{y}$ den Wert 3 hat? (Ohne Begründung!)

b) Wie sind die Werte der Variablen $\mathtt{x}$ und $\mathtt{y}$ nach Ausführung der Prozedur $\mathtt{f}()$, wenn vor Ausführung $\mathtt{x}$ den Wert 2 und $\mathtt{y}$ den Wert -3 hat? (Ohne Begründung!)

c) Welche der folgenden Tripel sind gültig? Begründen Sie Ihre Antwort.

   Für ein Tripel dessen Gültigkeit nicht durch die Gültigkeit des Tripels in (1) impliziert wird ist ein Gegenbeispiel anzugeben.

   i) $\{\mathtt{x} = \mathtt{y} = l\} \quad \mathtt{f}(); \quad \{\mathtt{y} = l^2\}$

   ii) $\{\mathtt{x} = l_{\mathtt{x}} \ \wedge \ \mathtt{y} = l_{\mathtt{y}} \ \wedge \ l_x \cdot l_{\mathtt{y}} \geq 0 \ \wedge \ \mathtt{z} = l_{\mathtt{z}}\} \quad \mathtt{f}(); \quad \{\mathtt{x} = l_{\mathtt{x}} \ \wedge \ \mathtt{y} = l_{\mathtt{x}} \cdot l_{\mathtt{y}} \ \wedge \ \mathtt{z} = l_{\mathtt{z}}\}$

   iii) $\{\mathtt{x} = l_{\mathtt{x}} \ \wedge \ \mathtt{y} = l + l_{\mathtt{x}}^2 \ \wedge \ l_x \geq 0 \ \wedge \ l_{\mathtt{x}} \cdot (l + l_{\mathtt{x}}^2) \geq 0\} \quad \mathtt{f}(); \quad \{\mathtt{x} = l_{\mathtt{x}} \ \wedge \ \mathtt{y} = l_{\mathtt{x}} \cdot l + l_{\mathtt{x}}^3\}$

Aufgabe 13.4 (P) OCaml

a) Definieren Sie eine OCaml-Funktion

$$\mathtt{jedes\_n\_te} \ : \ \mathtt{int} \ {-}{>} \ {'}\mathtt{a} \ \mathtt{list} \ {-}{>} \ {'}\mathtt{a} \ \mathtt{list}$$

   Ein Aufruf `jedes_n_te n l` soll jedes `n`-te Element aus der Liste `l` nehmen und aus diesen Elementen eine Liste konstruieren und zurückliefern.

   Beispiel: `jedes_n_te 3` $[1;2;3;4;5;6;7] = [3;6]$

b) Definieren Sie eine OCaml-Funktion

$$\mathtt{m} \ : \ ({'}\mathtt{a} \ {-}{>} \ {'}\mathtt{b} \ {-}{>} \ {'}\mathtt{c}) \ \mathtt{list} \ {-}{>} \ {'}\mathtt{a} \ \mathtt{list} \ {-}{>} \ {'}\mathtt{b} \ \mathtt{list} \ {-}{>} \ {'}\mathtt{c} \ \mathtt{list}$$

   Für

$$\mathtt{fs} = [\mathtt{f}_1; \ldots; \mathtt{f}_n], \quad \mathtt{xs} = [\mathtt{x}_1; \ldots; \mathtt{x}_n] \quad \text{und} \quad \mathtt{ys} = [\mathtt{y}_1; \ldots; \mathtt{y}_n]$$

   soll der Aufruf

$$\mathtt{m \ fs \ xs \ ys}$$

   das gleiche Ergebnis liefern wie die Auswertung des Ausdrucks

$$[\mathtt{f}_1 \ \mathtt{x}_1 \ \mathtt{y}_1; \ldots; \ \mathtt{f}_n \ \mathtt{x}_n \ \mathtt{y}_n].$$

   Beispiel: $\mathtt{m} \ [(\mathtt{fun \ x \ y} \ {-}{>} \ \mathtt{x} + \mathtt{y}); \ (\mathtt{fun \ x \ y} \ {-}{>} \ \mathtt{x} - \mathtt{y})] \ [2;2] \ [2;2] = [4;0]$

Aufgabe 13.5 (P) Rekursive Datentypen in OCaml

In dieser Aufgabe soll ein OCaml-Programm zur Verwaltung von Abteilungsstrukturen in Firmen entwickelt werden. Dabei sollen die hierarchische Gliederung der Abteilungen abgebildet und Angestellte den verschiedenen Abteilungen zugeordnet werden können.

a) Zu jedem Angestellten einer Firma soll der **Name** (als `string`), das **Alter** (als `int`) sowie die **Sozialversicherungsnummer** (als `string`) gespeichert werden.

   i) Definieren Sie einen geeigneten OCaml-Typ `ang` zur Repräsentation von Angestellten.

   ii) Definieren Sie eine OCaml-Funktion `alter_of_ang : ang -> int`, die das Alter eines Angestellten zurückliefert.

b) Es gibt zwei Sorten von Abteilungen: Eine Abteilung vom Typ **A** hat

   - einen **Namen**;
   - einen Angestellten in der Funktion des **Abteilungsleiters**;
   - beliebig viele der Abteilung angehörige **normale Angestellte**.

   Eine Abteilung vom Typ **B** hat **zusätzlich genau zwei** Abteilungen, die der Abteilung unmittelbar untergeordnet sind.

   i) Definieren Sie einen OCaml-Typ `abt` zur Repräsentation von Abteilungen.

   ii) Definieren Sie eine OCaml-Funktion `zaehle : abt -> int`, die zu einer Abteilung die **Anzahl** der unmittelbaren und mittelbaren **Unterabteilungen** bestimmt.

   iii) Definieren Sie eine OCaml-Funktion `alle : abt -> ang list`, die zu einer Abteilung die Liste **aller Angestellten** inclusive Abteilungsleiter bestimmt. Dabei sind auch die Angestellten in ummittelbaren und mittelbaren Unterabteilungen zu berücksichtigen.

   iv) Definieren Sie eine OCaml-Funktion `aelter : abt -> true`. Der Aufruf `aelter a` soll überprüfen, ob in der Abteilung `a` und in den mittelbaren sowie unmittelbaren Unterabteilungen stets der Abteilungsleiter älter ist als die in der Abteilung `a` tätigen normalen Angestellten. Ist dies erfüllt, so soll `true` und andernfalls `false` zurückgeliefert werden.

Hinweis: Sie dürfen den Operator `@` verwenden.

Big-Step Operationelle Semantik

Axiome: $v \Rightarrow v$ für jeden Wert v

Tupel:
$$\frac{e_1 \ \Rightarrow \ v_1 \qquad \dots \qquad e_k \ \Rightarrow \ v_k}{(e_1, \dots, e_k) \ \Rightarrow \ (v_1, \dots, v_k)} \ (T)$$

Listen:
$$\frac{e_1 \ \Rightarrow \ v_1 \qquad e_2 \ \Rightarrow \ v_2}{e_1 \texttt{ :: } e_2 \ \Rightarrow \ v_1 \texttt{ :: } v_2} \ (L)$$

Globale Definitionen:
$$\frac{f = e \qquad e \ \Rightarrow \ v}{f \ \Rightarrow \ v} \ (GD)$$

Lokale Definitionen:
$$\frac{e_1 \ \Rightarrow \ v_1 \qquad e_0[v_1/x] \ \Rightarrow \ v_0}{\texttt{let } x = e_1 \texttt{ in } e_0 \ \Rightarrow \ v_0} \ (LD)$$

Funktionsaufrufe:
$$\frac{e_1 \ \Rightarrow \ \texttt{fun } x \texttt{ -> } e_0 \qquad e_2 \ \Rightarrow \ v_2 \qquad e_0[v_2/x] \ \Rightarrow \ v_0}{e_1 \ e_2 \ \Rightarrow \ v_0} \ (App)$$

Pattern Matching:
$$\frac{e_0 \ \Rightarrow \ v' \equiv p_i[v_1/x_1, \dots, v_k/x_k] \qquad e_i[v_1/x_1, \dots, v_k/x_k] \ \Rightarrow \ v}{\texttt{match } e_0 \texttt{ with } p_1 \texttt{ -> } e_1 \texttt{ | } \dots \texttt{ | } p_m \texttt{ -> } e_m \ \Rightarrow \ v} \ (PM)$$

— sofern v' auf keines der Muster $p_1, \dots, p_{i-1}$ passt

Eingebaute Operatoren:
$$\frac{e_1 \ \Rightarrow \ v_1 \qquad e_2 \ \Rightarrow \ v_2 \qquad v_1 \ \mathsf{op} \ v_2 \Rightarrow v}{e_1 \ \mathsf{op} \ e_2 \ \Rightarrow \ v} \ (Op)$$

— Unäre Operatoren werden analog behandelt.

Substitutionslemma

$$\frac{e_1 = e_2}{e[e_1/x] = e[e_2/x]}$$