Technische Universität München
Fakultät für Informatik
Prof. Dr. A. Knoll

SS 2006
Übungsblatt 6
16. Juni 2006

Übungen zu Einführung in die Informatik I

Aufgabe 21 Terminierung

Für welche Eingabewerte terminieren die folgenden Funktionen? Beweisen Sie Ihre Aussagen!

a)
```
let rec fun f_a (m,n) =
          if (n = 0) then 1
          else f_a (m+1,n-1) + f_a (m+2,n-1);;
```

b)
```
let rec f_b (m,n) =
          if (m = n) then 1
          else f_b (m,n-1) + f_b (m+1,n);;
```

c)
```
let rec f_c (m,n) =
          if (m = n) then 1
          else f_c (m+1,n-1);;
```

d)
```
let rec f_d n =
          if (n < 100) then f_d (200-n-1)
          else if (n > 100) then f_d (200-n+1)
          else n;;
```

Aufgabe 22 Terminierung

Für welche Eingabewerte terminiert die folgende Funktion? Beweisen Sie Ihre Aussage!

```
let rec f (n,m,k) =
        if (n=m) then k
        else if (n<m) then f (2*m-n-1,m,k+1)
        else f (2*m-n+1,m,k+1);;
```

Aufgabe 23 Ordnungen über Zeichensequenzen

Gegeben sei ein Zeichenvorrat C. Ein Wort y heißt ein Präfix eines Wortes $x \in C^*$, wenn es ein $u \in C^*$ gibt mit $x = y \circ u$. Wir sagen: $y \sqsubseteq x$ genau dann, wenn y ein Präfix von x ist.

a) Beweisen Sie, daß C^* bezüglich der Präfixrelation $\sqsubseteq$ partiell geordnet ist.

b) Zeigen Sie, dass die Präfixrelation

keine totale Ordnung,

eine noethersche (fundierte), jedoch keine wohlgeordnete Ordnung ist.

c) Beweisen Sie, dass die lexikographische Ordnung über C^* nicht noethersch (fundiert) ist. Die lexikographische Ordnung ist dabei wie folgt definiert: Sei $(C, \leq_C)$ ein total geordneter, endlicher Zeichenvorrat. Bezeichne $\sqsubseteq$ die oben definierte Präfixordnung. Dann ist die lexikographische Ordnung $\leq$ auf C definiert durch

(i) $\varepsilon \leq x$ für alle $x \in C^*$

(ii) $x \leq y$ für $x, y \in C^*$ und $x \sqsubseteq y$

(iii) $xay \leq xby'$ für $x, y, y' \in C^*$, $a, b \in C$ und $a \neq b$ und $a \leq_C b$.

d) Geben Sie eine fundierte, totale Ordnung über C^* an.

Aufgabe 24 Vergleich von Bäumen

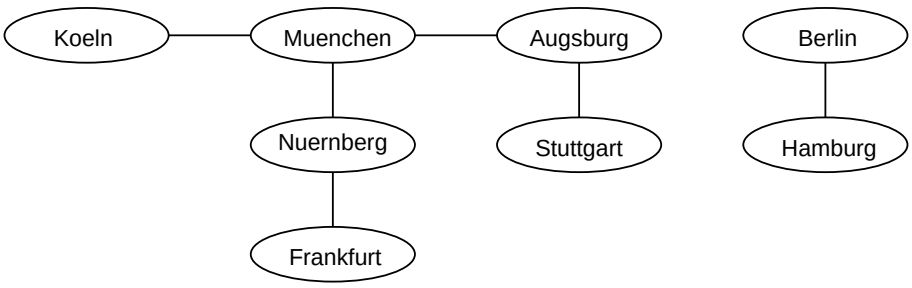
In dieser Aufgabe wenden wir uns nochmals sortierten binären Bäumen zu. Zwei sortierte Binärbäume können strukturell verschieden sein, obwohl sie die gleichen Elemente enthalten. Schreiben Sie eine Funktion `compare_tree` die zwei Binärbäume sowohl strukturell als auch inhaltlich vergleicht und `true` zurückliefert, falls beide Bäume identisch sind.

a) Implementieren Sie die Funktion `compare_tree` indem Sie aus dem Baum zuerst eine Liste erstellen und diese dann überprüfen.

b) (Optional) Implementieren Sie die Funktion `compare_tree` indem Sie rekursiv den Baum durchlaufen.

Aufgabe 25 (H) Tiefensuche in Graphen und Straßennetze (10 Punkte)

Wie findet man in einer Straßenkarte den Weg von Stadt A nach B? Aus der Sicht der Informatik handelt es sich bei einem Straßennetz um einen ungerichteten Graphen mit Städten als Knoten und den Straßen als Kanten. Mathematisch läßt sich ein ungerichteter Graph G durch eine endliche Menge von Knoten E und eine endliche Menge von Kanten $K = \{(u,v) | u, v \in E \wedge u \neq v \wedge \texttt{Es gibt eine Verbindung zwischen } u \texttt{ und } v\}$ beschreiben. Anschaulich ergibt sich beispielsweise für ein (vereinfachtes) Straßennetz der Bundesrepublik folgender Graph:



Mathematisch formuliert, ergibt sich:

$$E = \{M, K, N, F, A, S, B, H\}$$

$$K = \{(M,K), (M,A), (M,N), (A,S), (F,N), (H,B)\}.$$

Unter Verwendung der so genannten Tiefensuche läßt sich nun ein Pfad zwischen zwei Knoten berechnen. Der Algorithmus hat folgende informelle Beschreibung:

```
Speichere Anfangsknoten auf einer stackartigen Datenstruktur
Solange der Stack nicht leer
  Entferne Knoten vom Stack
  Falls der Knoten nicht der Zielknoten ist
    Markiere ihn als besucht
    Speichere alle unbesuchten Nachbarn des Knotens auf dem Stack
```

Bei der Suche werden Knoten entlang eines Pfades solange in die Tiefe expandiert, bis der betrachtete Knoten keinen unbesuchten Nachfolger mehr hat. Dann wird mit Hilfe des Stack von einem anderen Knoten aus ein anderer Pfad exploriert. Sie können bei Ihrer Lösung folgendermaßen vorgehen:

- Implementieren Sie die Zugriffsoperationen der *LIFO*-Datenstruktur `Stack` unter Zuhilfenahme von Listen.
- Definieren Sie entsprechende Datenstrukturen für Knoten und Kanten. Bedenken Sie, daß ein Knoten als besucht markiert werden soll.
- Implementieren Sie eine Funktion, die eine Liste der Nachbarn eines Knotens liefert.
- Implementieren Sie eine Funktion, die in einer Liste von Knoten einen Knoten als besucht markiert und eine Funktion, die den Zustand eines Knotens (d.h. besucht oder nicht) zurückgibt.
- Implementieren Sie den eigentlichen Algorithmus. Testen Sie Ihr Programm indem Sie überprüfen, ob man von Stuttgart nach Frankfurt bzw. Berlin kommt.