Technische Universität München
Fakultät für Informatik
Prof. Dr. M. Broy

WS 2002
Übungsblatt 2
1. November 2002

# Übungen zur Einführung in die Informatik III

## Aufgabe 5 Zustandsübergänge des Fahrstuhls

Sei A die Menge der Aktionen aus Aufgabe 1, Blatt 2. Diese werden nun als Zustandsänderungen des Liftsystems weiter detailliert. Dazu werden den beteiligten Einheiten wie Liftkabinen, Etagen und Personen unter anderem folgende Zustandsvariablen zugeordnet, auf denen die Aktionen Zuweisungen ausführen:

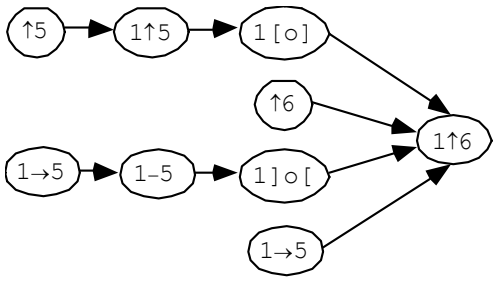
| Einheit | Variable | Variablenwerte | Bedeutung |
|---|---|---|---|
| Liftkabine i | $fr_i$ | $\{\uparrow, \downarrow\}$ | Fahrtrichtungsanzeige in Kabine i |
| | $et_i$ | $2^{[0,m-1]}$ | Menge der gedrückten (beleuchteten) Etagentasten in Kabine i |
| | $akt_i$ | $[0,m-1]$ | aktuelle Etage der Kabine i |
| Etage j | $u_j$ | {**O**, **L**} | Knopf ↑ auf Etage j ist nicht gedrückt / gedrückt ($u_{m-1}$ gibt es nicht) |
| | $d_j$ | {**O**, **L**} | Knopf ↓ auf Etage j ist nicht gedrückt / gedrückt ($d_0$ gibt es nicht) |
| Person k | $l_k$ | $[-1,n-1]$ | Liftkabine, in der sich Person k befindet (−1 = in keiner Kabine) |

**a)** Ordnen Sie den Aktionen in A Zuweisungen an diese Variablen zu.

**b)** Welche Aktionen sind deterministisch, welche total?

**c)** Definieren Sie die Relation $Conflict \subseteq A \times A$, die für jedes Paar von Aktionen angibt, ob ein Konflikt besteht, d.h. ob die Bernsteinbedingung verletzt ist.

**d)** Sei p der nebenstehende Prozeß. Welche parallelen Aktionen stehen in Konflikt?




**e)** Als (Teil-)Zustandsraum S betrachten wir die Menge der Belegungen der Variablen ($l_1$, $l_2$, $l_3$, $u_5$, $u_6$, $akt_1$, $et_1$). Geben Sie für p die Menge aller möglichen Zustandsübergänge $\sigma_0 \longrightarrow^p \sigma_1$ an, wobei $\sigma_0 = (-1, -1, 1, \mathbf{L}, \mathbf{O}, 4, \{5, 6\})$.

**Aufgabe 6 Programmabläufe als Prozesse**

Den Ablauf eines Programms kann man auch als Prozess im Sinne einer Aktionsstruktur deuten.

Gegeben sei die folgende prozedurale Variante des *quicksort* Algorithmus:

```
proc quicksort = (var [1..n] array m a, nat min, nat max):
⌈   var i j x := min+1, max, a[min];
    while i ≤ j do
         if a[i] ≤ x then i := i+1
         elif x ≤ a[j] then j := j-1
         else a[i], a[j] :=a [j], a[i]
         fi;
    od;
    if min < j then a[min], a[j] := a[j], a[min] fi;
    if min < j-1 then quicksort (a, min, j-1) fi;
    if j+1 < max then quicksort (a, j+1, max) fi;
                                                    ⌋
```

Der Ablauf des Programms für eine bestimmte Eingabe kann zunächst als sequentielle Aktionstruktur verstanden werden.

**a)** Geben Sie die Menge der auftretenden Aktionen für die folgenden Eingaben an:

i) `[3, 4, 5, 2, 1]`

ii) `[3, 4, 7, 5, 2, 1, 6]`

Bei genauerer Betrachtung stellt man fest, dass manche Aktionen unabhängig voneinander ausgeführt werden können und andere nicht ohne Konflikt nebeneinander ausgeführt werden können. Das führt zu allgemeineren (nicht mehr unbedingt sequentiellen) Aktionsstrukturen.

**b)** Zwischen welchen Aktionen treten allgemein Konflikte auf?

**c)** Geben Sie eine Aktionsstruktur für die beiden Prozesse an.

**d)** Wie ändert sich die Aktionsstruktur, wenn man anstelle der parallelen Zuweisung

```
a[i], a[j] :=a [j], a[i]
```

die drei Zuweisungen

```
h := a[i]
a[i] := a [j]
a[j] := h
```

verwendet ?

## Aufgabe 7 (H) Menge der Sequentialisierungen

Nach einem Satz der Vorlesung zu endlich fundierten Prozessen lässt sich die Kausalitätsrelation eines Prozesses aus der Menge seiner Sequentialisierungen rekonstruieren. Die folgende Menge besteht aus sämtlichen Sequentialisierungen eines Prozesses. Bestimmen Sie die Kausalitätsrelation.

$\{ e_0 \to e_1 \to e_2 \to e_3 \to e_4 \to e_5,$
$e_0 \to e_1 \to e_2 \to e_4 \to e_3 \to e_5,$
$e_0 \to e_1 \to e_3 \to e_2 \to e_4 \to e_5,$
$e_0 \to e_2 \to e_1 \to e_3 \to e_4 \to e_5,$
$e_0 \to e_2 \to e_1 \to e_4 \to e_3 \to e_5 \}$

## Aufgabe 8 (P) Aktionsstruktur, Fortführung

In der Aufgabe 4, Blatt 1 wird ein Aktionsgraph im Speicher aufgebaut. Wir erweitern nun diesen Graphen um Funktionalität zur Berechnung von vollständigen Sequentialisierungen. Dazu wird ein Referenzzähler eingeführt, die für jeden Knoten die Anzahl der auf ihn gerichteten Pfeile zählt. Ein Knoten ist genau dann ohne Vorgänger, wenn dieser Referenzzähler = 0 ist.

(a) Benutzen Sie diesen Mechanismus, um eine vollständige Sequentialisierung der Aktionsstruktur zu berechnen und in einem Vektor abzulegen (der vom Hauptprogramm ausgegeben wird), indem Sie aus dem Graph sukzessive alle Knoten ohne Vorgänger entfernen.
Erweitern Sie Ihr Programm aus Aufgabe 4 (oder alternativ den angebotenen Lösungsvorschlag (siehe Web) um eine solche Komponente. Versuchen Sie dabei gewinnbringend Vererbung einzusetzen, um z.B. die ursprüngliche Knotenklasse ohne Modifikation wiederzuverwenden.

(b) Erweitern Sie Ihr Programm nun, so dass es alle vollständigen Sequentialisierungen ausgibt.

(c) **Ergänzung** zum Thema Testen: Für das Testen von Programmen gibt es ein kleines Testframework namens „JUnit“. Laden Sie dazu junit3.8.1 von http://www.junit.org und lesen Sie das „JUnit Cookbook“, das unter http://junit.sourceforge.net/doc/cookbook/cookbook.htm zu finden ist. Schreiben Sie damit einige automatisierte Tests für Ihr Programm. Die Musterlösung der Aufgabe 4 enthält ebenfalls ein Grundgerüst für derartige Tests.