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

WS 2002
Lösungsblatt 2
1. November 2002

Übungen zur Einführung in die Informatik III

Aufgabe 5 Zustandsübergänge des Fahrstuhl

a) Den Aktionen ordnen wir folgende Zuweisungen zu

↑j	□j	i↑j	i□j
u_j:= **L**	d_j:= **L**	u_j:= **O**; akt_i:= j	d_j:= **O**; akt_i:= j

i□ j	i-j	i↑	i□
et_i:= et_i□{j}	et_i:= et_i\{j}; akt_i:= j	fr_i:= ↑	fr_i:= □

i[o]	i]o[
l_k:= i für ein k mit l_k=-1	l_k:= -1 für ein k mit l_k=i

b) Alle Aktionen sind deterministisch außer i[o] und i]o[. Diese beiden Aktionen sind nichtdeterministisch, da sie verschiedene Übergänge zulassen, wenn es mehr als eine Person gibt, die bei einer Liftkabine einsteigen bzw. aussteigen kann.
Alle Aktionen sind auch total außer i[o] und i]o[, da die betreffenden Zuweisungen immer ausführbar sind und einen definierten Wert liefern. Bei den Aktionen i[o] und i]o[ist dies nicht der Fall, falls es kein l_k mit l_k=-1 bzw. kein l_k mit l_k=i gibt.

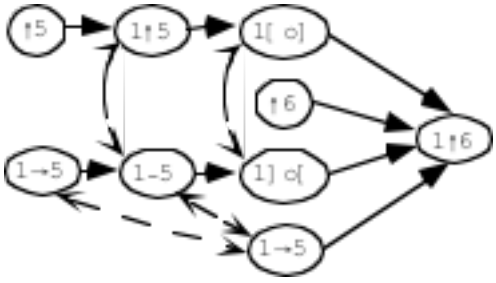
c) Conflict = {(a,a) | a□A} □

{

(↑j, i↑j), (i_1↑j, i_2↑j),	wg. der gemeinsamen Variable u_j
(□j, i□j), (i_1□j, i_2□j),	wg. der gemeinsamen Variable d_j
(i↑j_1, i□j_2), (i↑j_1, i↑j_2), (i□j_1, i□j_2),	wg. der gemeinsamen Variable akt_i
(i□j_1, i□j_2), (i□j_1, i↑j_2), (i□j_1, i□j_2),	wg. der gemeinsamen Variable akt_i
(i□ j_1, i-j_2), (i□ j_1, i□ j_2), (i-j_1, i-j_2),	wg. der gemeinsamen Variable et_i
(i↑, i□),	wg. der gemeinsamen Variable fr_i
(i[o], i]o[)	wg. der gemeinsamen Variablen l_1, l_2, ...

}

d) In dem Prozess p stehen die gekennzeichneten parallelen Paare in Konflikt, da sie auf die gemeinsame Variable zugreifen.
Alle übrigen in Konflikt stehenden Paare sind nicht parallel.



e) Die möglichen Zustandübergänge $\sigma_0 \xrightarrow{p} \sigma_1$ von `p` ergeben sich aus den Übergängen $\sigma_0 \xrightarrow{s} \sigma_1$ der Spuren `s` von `p`. Da für konfliktfreie Aktionen `a`, `b` stets gilt $\sigma_0 \xrightarrow{<a\ b>} \sigma_1 \xrightarrow{<b\ a>} \sigma_0$, genügt es, die möglichen Sequentialisierungen der in **d)** gefunden kritischen Paare zu untersuchen. Wir stellen deren Zustandübergänge nur für die betroffenen Variablen dar.
Die Sequentialisierungen von `1[o]` und `1]o[` ergeben:

```
(-1, -1, 1, ...)   --<1[o] 1]o[>--   (1, -1, -1, ...)
                                     (-1, 1, -1, ...)
                                     (-1, -1, 1, ...)
(-1, -1, 1, ...)   --<1]o[ 1[o]>--   (1, -1, -1, ...)
                                     (-1, 1, -1, ...)
                                     (-1, -1, 1, ...)
```

Bei der Sequentialisierung von `1↔5` und `1↔5` spielt die Reihenfolge keine Rolle.
Die Sequentialisierung von `1↔5` und `1-5` ergeben:

```
(...,{5,6})   --<1-5 1↔5>--   (...,{5,6})
(...,{5,6})   --<1↔5 1-5>--   (...,{6})
```

Insgesamt ergeben sich also 6 mögliche Übergänge:

```
(-1, -1, 1, L, O, 4, {5,6})   --p--> (1, -1, -1, O, O, 6, {5,6})
                              --p--> (1, -1, -1, O, O, 6, {6})
                              --p--> (-1, 1, -1, O, O, 6, {5,6})
                              --p--> (-1, 1, -1, O, O, 6, {6})
                              --p--> (-1, -1, 1, O, O, 6, {5,6})
                              --p--> (-1, -1, 1, O, O, 6, {6})
```

Aufgabe 6 (T) Programmabläufe als Prozesse

a) Die auftretenden Aktionen sind parallele Zuweisungen der Form
`a[i], a[j] := a [j], a[i],` die hier abgekürzt werden durch `i↔ j`.

i)

Aktion	Zustand von a
	[3, 4, 5, 2, 1]
1↔ 4	[3, 1, 5, 2, 4]
2↔ 3	[3, 1, 2, 5, 4]
0↔ 2	[2, 1, 3, 5, 4]
0↔ 1	[1, 2, 3, 5, 4]
3↔ 4	[1, 2, 3, 4, 5]

ii)

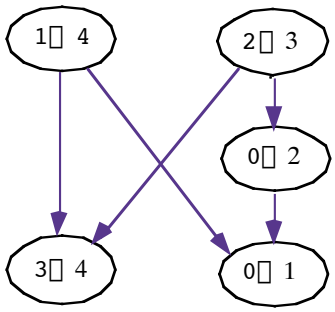
Aktion	Zustand von a
	[3, 4, 7, 5, 2, 1, 6]
1↔ 5	[3, 1, 7, 5, 2, 4, 6]
2↔ 4	[3, 1, 2, 5, 7, 4, 6]
0↔ 2	[2, 1, 3, 5, 7, 4, 6]
0↔ 1	[1, 2, 3, 5, 7, 4, 6]
4↔ 5	[1, 2, 3, 5, 4, 7, 6]
3↔ 4	[1, 2, 3, 4, 5, 7, 6]
5↔ 6	[1, 2, 3, 4, 5, 6, 7]

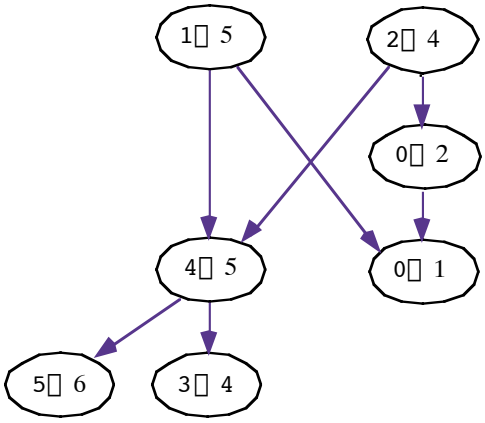
b) Zwischen zwei Aktionen $i_1 \leftrightarrow j_1$ und $i_2 \leftrightarrow j_2$ tritt ein Konflikt auf, wenn

$$\{i_1, j_1\} \cap \{i_2, j_2\} \neq \{\}$$

(Bernsteinbedingung)

c) Die Abhängigkeiten zwischen den Aktionen lassen sich durch die folgenden Aktionsdiagramme darstellen:

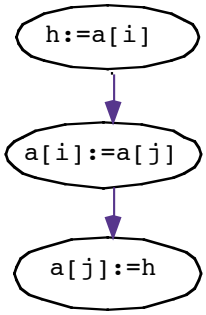




d) Anstelle einer Aktion `i j` treten jetzt die drei Aktionen

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

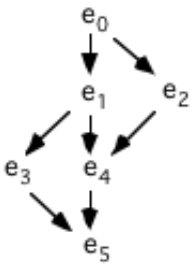
mit den folgenden Abhängigkeiten:



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.

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



In der ursprünglichen Kausalitätsrelation $\prec_0$ liegt ein Ereignis vor einem anderen, wenn dies in allen Sequentialisierungen der Fall ist, sonst nicht. Das führt auf die oben stehende Relation $\prec_0$.

Aufgabe 8 (P) Aktionsstruktur, Fortführung

a) In Abb. 8.1 ist die Speichersituation zu Beginn des Programmablaufs mit einem bereits in der Lösung zur Aufgabe 4 genutzten Beispiel skizziert. Während des Programmablaufs erhalten wir die in Tabelle 8.2 angegebenen Situationen.

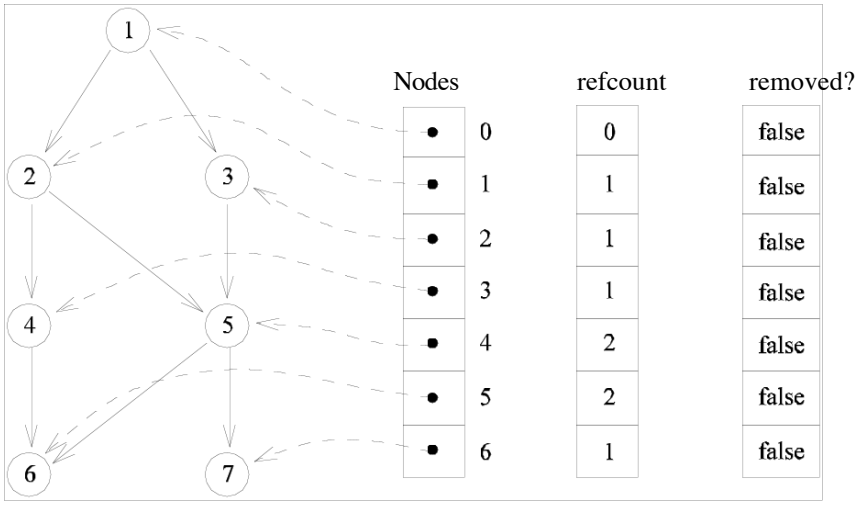


Abb. 8.1 Speichersituation zu Beginn des Programmablaufs

In dem Vector `ergebnis` steht am Ende eine von mehreren vollständigen Sequentialisierungen des Aktionsgraphen. In der Die Knoteninhalte werden in einer Subklasse `AcNode2` erweitert. Neu hinzugekommen sind der Referenzzähler und ein Boolesches Flag, das angibt, ob ein Knoten bereits aus dem Graph genommen wurde. Die Methode `isRemovable` testet ob ein Knoten ohne Vorgänger ist und entfernt ihn gegebenenfalls.

Start	ergebnis	*	*	*	*	*	*	*
	refcount	0	1	1	1	2	2	1
	removed	f	f	f	f	f	f	f
Step 1	ergebnis	1	*	*	*	*	*	*
	refcount	-	0	0	1	2	2	1
	removed	t	f	f	f	f	f	f
Step 2	ergebnis	1	3	2	*	*	*	*
	refcount	-	-	-	0	0	2	1
	removed	t	t	t	f	f	f	f
Step 3	ergebnis	1	3	2	5	4	*	*
	refcount	-	-	-	-	-	0	0
	removed	t	t	t	t	t	f	f
Step 4	ergebnis	1	3	2	5	4	7	6
	refcount	-	-	-	-	-	-	-
	removed	t	t	t	t	t	t	t

Tab. 8.2 Speichersituationen während des Programmablaufs

b) In der Weiterentwicklung des Programms zur Knotenelimination aus der letzten Teilaufgabe wird die lineare Suche nach freien Knoten durch ein Backtracking-Verfahren ersetzt. Das Ergebnis wird nicht in einem Vektor von Aktionen, sondern einer Menge solcher Vektoren abgelegt.

c) Die entstehende Aufgabe ist komplex genug, um Tests notwendig zu machen. Tests können manuell erfolgen, indem mehrere Programmläufe geprüft werden. Ein solcher Test ist aber zeitaufwändig und wird daher ungern wiederholt. Wird das Wissen über einen Programmlauf in einen automatisierten Test abgelegt, so kann dieser jederzeit wiederholt werden, ohne weitere Prüfung durch den Anwender zu erfordern. Typischerweise wird für ausgesuchte Aktionsstrukturen das Ergebnis geprüft, indem etwa die Anzahl der Sequentialisierungen oder deren Länge geprüft wird. Es bietet sich an, Randfälle zu prüfen: einelementige Aktionsstruktur, eine Aktionsstruktur ohne Abhängigkeiten (maximal parallel) oder eine bereits vollständig sequentialisierte Form, sowie zum Beispiel die oben gezeigte. Zur Implementierung eines Tests ist einmalig das erwartete Ergebnis auszurechnen und im Test zu kodieren. Das Framework Junit bietet eine ideale Umgebung Tests zu definieren.

((Java Code zur Aufgabe kommt noch))