Ludwig-Maximilians-Universität München
Institut für Informatik
Prof. Dr. F. Kröger, M. Hammer

WS 2006/2007
Übungsblatt 10

Temporale Logik und Zustandssysteme Lösungsvorschlag

Aufgabe 10-1 **Modellierung und Verifikation eines Fahrstuhls** (14 Punkte)

Ein Aufzug bedient n Stockwerke eines Hauses. Der Aufzugschacht wird auf jedem Stockwerk durch eine elektrische Schiebetür abgeschlossen. Zur Anforderung des Aufzugs befinden sich auf jedem Stockwerk Druckknöpfe (je einer im untersten bzw. obersten Stockwerk, ansonsten je zwei für "Fahrt nach oben" bzw. "Fahrt nach unten"). In der Aufzugkabine befindet sich eine Leiste von Druckknöpfen (je einer pro Stockwerk) zur Eingabe der Fahrtziele.

a) Modellieren Sie die Steuerung für den Aufzug und die Türen durch ein faires Zustandssystem Γ mit Startzuständen (frSTS) bezüglich einer geeigneten Signatur SIG und prädikatenlogischen Struktur S. Berücksichtigen Sie dabei die folgenden Anforderungen:

- Wenn keine Fahrtwünsche vorliegen, bleibt der Aufzug stehen.
- Der Aufzug ändert seine Richtung nur dann, wenn keine weiteren Fahrtwünsche in der bisherigen Richtung vorliegen.
- Erreicht der Aufzug ein Stockwerk, für das eine Anforderung vorliegt, so hält der Aufzug an. Dies gilt nicht, falls der Fahrtwunsch die entgegengesetzte Richtung angibt, aber weitere Fahrtwünsche für die aktuelle Fahrtrichtung anliegen.
- Die Fahrt von einem Stockwerk zum nächsten dauert mehrere Taktzyklen der Steuerung und kann daher nicht als (atomare) Aktion modelliert werden. Insbesondere können während der Fahrt neue Fahrtwünsche eintreffen.
- Der Aufzug darf nur losfahren, wenn die Tür des aktuellen Stockwerks ganz verschlossen ist. Das Öffnen bzw. Schließen einer Tür dauert mehrere Taktzyklen und kann daher nicht als (atomare) Aktion modelliert werden. Während des Schließens einer Tür kann diese durch erneutes Drücken der Anforderungstaste (im Lift oder am Stockwerk) wieder geöffnet werden.

Verwenden Sie dabei die Menge $Act = \{req\_up_i, req\_dn_i, req\_dest_i, start\_lift, pass\_floor, stop\_lift, door\_open, close\_door, door\_closed, reopen\_door \mid i \in \{1, \ldots, n\}\}$ mit folgenden informellen Bedeutungen:

- req_up_i, req_down_i: Im Stockwerk i wird ein Wunsch nach einer Fahrt nach oben/unten angezeigt.
- req_dest_i: Im Fahrstuhl wird das Stockwerk i als Fahrtziel angefordert.
- $start\_lift$: Nach dem Schliessen der Türen setzt sich der Lift in Bewegung.
- $pass\_floor$: Der fahrende Lift passiert ein Stockwerk, ohne anzuhalten.
- $stop\_lift$: Der fahrende Lift erreicht ein Stockwerk, in welchem er anhält.
- $door\_open$: Eine sich öffnende Tür erreicht den Zustand, in dem sie ganz geöffnet ist.
- $close\_door$: Das Schliessen einer offenen Tür wird ausgelöst.
- $door\_closed$: Das Schliessen einer Tür wird abgeschlossen.
- $reopen\_door$: Eine geschlossene oder sich schliessende Tür wird wieder geöffnet, sofern das möglich ist.

b) Geben Sie für jede Aktion von Γ geeignete Ausführbarkeitsbedingungen an und beschreiben Sie die Wirkung der Aktionen durch Formeln von $\mathcal{L}_{\mathrm{TL}\Gamma}$.

Lösung: (Teilaufgabe a und b): Für diese Aufgabe gibt es verschiedene Lösungsmöglichkeiten. Wir legen eine Signatur mit den Sorten $FLOOR$, $FLOORS$ und $DOOR$ (für Stockwerke, Mengen von Stockwerken und Status der Türen) zu Grunde, die in der Struktur S durch $|\mathsf{S}|_{FLOOR} = \{1, \ldots, N\}$, $|\mathsf{S}|_{FLOORS} = \mathcal{P}(\{1, \ldots, N\})$, also Mengen von Stockwerken, und $|\mathsf{S}|_{DOOR} = \{\text{“open”}, \text{“closed”}, \text{“opening”}, \text{“closing”}\}$ interpretiert werden.

Als Funktions- und Prädikatensymbole führen wir ein:

- die Konstanten $1^{(\varepsilon,FLOOR)}, \ldots, N^{(\varepsilon,FLOOR)}$ mit offensichtlicher Interpretation,
- die Funktionen $succ^{(FLOOR,FLOOR)}$ und $pred^{(FLOOR,FLOOR)}$ mit der Interpretation

$$\mathsf{S}(succ(i)) = \left\{ \begin{array}{ll} i+1 & \text{falls } i < N \\ N & \text{falls } i = N \end{array} \right\} \qquad \text{bzw.} \qquad \mathsf{S}(pred(i)) = \left\{ \begin{array}{ll} i-1 & \text{falls } i > 1 \\ 1 & \text{falls } i = 1 \end{array} \right\}$$

- die Konstante $empty^{(\varepsilon,FLOORS)}$ mit der Interpretation $\mathsf{S}(empty) = \emptyset$,
- die Funktionen $insert^{(FLOORS\ FLOOR,FLOORS)}$ und $remove^{(FLOORS\ FLOOR,FLOORS)}$ mit den Interpretationen $\mathsf{S}(insert(i,s)) = s \cup \{i\}$ und $\mathsf{S}(remove(i,s)) = s \setminus \{i\}$,
- die Konstanten $open^{(\varepsilon,DOOR)}$, $closed^{(\varepsilon,DOOR)}$, $opening^{(\varepsilon,DOOR)}$ und $closing^{(\varepsilon,DOOR)}$ mit offensichtlicher Interpretation,
- das Prädikatszeichen $<^{(FLOOR\ FLOOR)}$ mit natürlicher Interpretation ($i > j$ steht für $j < i$),
- das Prädikatszeichen $\in^{(FLOOR\ FLOORS)}$ (in Infix-Schreibweise), das durch die Element-Relation interpretiert wird.

Der Aufzug wird nun modelliert durch ein frSTS $\Gamma = (X, V, Z, T, Act, start, \mathcal{E})$. Die Menge X enthält folgende Systemvariablen (Sorte jeweils mit angegeben):

$floor^{FLOOR}$: aktuelles bzw. (bei fahrendem Aufzug) zuletzt besuchtes Stockwerk,

$reqs\_up^{FLOORS}, reqs\_dn^{FLOORS}, reqs\_dest^{FLOORS}$: registrierte Fahrtwünsche: $reqs\_up$ und $reqs\_dn$ repräsentieren die noch unerfüllten Anforderungen von außen (jeweils für Auf- oder Abwärtsfahrt), $reqs\_dest$ diejenigen Anforderungen, die von innerhalb der Kabine eingegeben wurden,

$door^{DOOR}$: Stellung der Tür am aktuellen (bzw. zuletzt besuchten) Stockwerk.

Hinzu kommen noch die folgenden Booleschen Systemvariablen in V:

$stopped$: gibt an, ob der Lift derzeit angehalten ist,

$going\_up$: gibt an, ob der Lift aktuell nach oben fährt (bzw. zuletzt nach oben fuhr).

Die Menge S der Systemzustände wird hier nicht weiter eingeschränkt, die Übergangsrelation T wird durch die axiomatische Beschreibung der Systemaktionen (vgl. unten) gegeben. Die Startbedingung ist

$$\begin{array}{rcl} start & \equiv & \wedge\ reqs\_up = empty \wedge reqs\_dn = empty \wedge reqs\_dest = empty \\ & & \wedge\ door = closed \wedge stopped \end{array}$$

Zur Spezifikation der Übergangsrelation definieren wir für die folgenden Systemaktionen jeweils ihre Wirkung und ihre Ausführbarkeitsbedingungen. Dabei schreiben wir als Abkürzung $\text{UNCHANGED}(x_1, \ldots, x_n)$ für die Formel $x_1' = x_1 \wedge \ldots \wedge x_n' = x_n$ (für Boolesche Systemvariablen eigentlich $x_i' \leftrightarrow x_i$).

Die erste Gruppe von Systemaktionen betrifft die Eingabe von Fahrtwünschen, entweder in der Kabine oder von außen. Falls der Aufzug angehalten wurde, wird der Wunsch an das aktuelle Stockwerk zu fahren, ignoriert. (Das Drücken der Anforderungstaste zum Öffnen der Kabinentür wird weiter unten behandelt.)

$$\begin{array}{rcll} exec\, req\_up_i & \rightarrow & \wedge\ floor = i \rightarrow \neg stopped & (i = 1, \ldots, N-1) \\ & & \wedge\ reqs\_up' = insert(reqs\_up, i) & \\ & & \wedge\ \text{UNCHANGED}(reqs\_dn, reqs\_dest, floor, door, stopped, going\_up) & \\ \text{enabled}_{req\_up_i} & \equiv & floor = i \rightarrow \neg stopped & (i = 1, \ldots, N-1) \\ exec\, req\_dn_i & \rightarrow & \wedge\ floor = i \rightarrow \neg stopped & (i = 2, \ldots, N) \\ & & \wedge\ reqs\_dn' = insert(reqs\_dn, i) & \\ & & \wedge\ \text{UNCHANGED}(reqs\_up, reqs\_dest, floor, door, stopped, going\_up) & \\ \text{enabled}_{req\_dn_i} & \equiv & floor = i \rightarrow \neg stopped & (i = 2, \ldots, N) \\ exec\, req\_dest_i & \rightarrow & \wedge\ floor = i \rightarrow \neg stopped & (i = 1, \ldots, N) \\ & & \wedge\ reqs\_dest' = insert(reqs\_dest, i) & \\ & & \wedge\ \text{UNCHANGED}(reqs\_up, reqs\_dn, floor, door, stopped, going\_up) & \\ \text{enabled}_{req\_dest_i} & \equiv & floor = i \rightarrow \neg stopped & (i = 1, \ldots, N) \end{array}$$

Die folgende Gruppe von Aktionen modelliert die Bewegung des Aufzugs (Losfahren, Durchfahren eines Stockwerks und Anhalten). Die Richtung, in die der Aufzug fährt, wird beim Losfahren neu bestimmt. Beim Anhalten des Aufzugs wird gleichzeitig das Öffnen der Kabinentür veranlasst. Wir schreiben im folgenden **if** P **then** A **else** B für die Formel $(P \rightarrow A) \wedge (\neg P \rightarrow B)$.

$$
\begin{array}{rcl}
exec\, start\_lift & \rightarrow & \wedge\ stopped \wedge door = closed \\
& & \wedge\ \vee\ need\_go\_up \wedge going\_up' \\
& & \quad \vee\ need\_go\_dn \wedge \neg going\_up' \\
& & \wedge\ \neg stopped' \\
& & \wedge\ \text{UNCHANGED}(reqs\_up, reqs\_dn, reqs\_dest, floor, door) \\
\text{enabled}_{start\_lift} & \equiv & stopped \wedge door = closed \wedge (need\_go\_up \vee need\_go\_dn) \\
exec\, pass\_floor & \rightarrow & \wedge\ \neg stopped \\
& & \wedge\ \textbf{if}\ going\_up\ \textbf{then}\ floor' = succ(floor)\ \textbf{else}\ floor' = pred(floor) \\
& & \wedge\ \neg need\_stop(floor') \\
& & \wedge\ \text{UNCHANGED}(reqs\_up, reqs\_dn, reqs\_dest, going\_up, stopped, door) \\
\text{enabled}_{pass\_floor} & \equiv & \neg stopped \wedge (\textbf{if}\ going\_up\ \textbf{then} \neg need\_stop(succ(floor))\ \textbf{else}\ \neg need\_stop(pred(floor))) \\
exec\, stop\_lift & \rightarrow & \wedge\ \neg stopped \\
& & \wedge\ \textbf{if}\ going\_up\ \textbf{then}\ floor' = succ(floor)\ \textbf{else}\ floor' = pred(floor) \\
& & \wedge\ need\_stop(floor') \wedge stopped' \wedge door' = opening \\
& & \wedge\ \text{UNCHANGED}(reqs\_up, reqs\_dn, reqs\_dest, going\_up) \\
\text{enabled}_{stop\_lift} & \equiv & \neg stopped \wedge (\textbf{if}\ going\_up\ \textbf{then}\ need\_stop(succ(floor))\ \textbf{else}\ need\_stop(pred(floor)))
\end{array}
$$

Die vorstehenden Aktionen benutzen die Makros $need\_go\_up$ bzw. $need\_go\_dn$, die bestimmen, ob ein Fahrtwunsch nach oben bzw. unten vorliegt, sowie das Makro $need\_stop(f)$ zur Entscheidung, ob der Aufzug am Stockwerk f anhalten soll. Diese sind folgendermaßen definiert:

$$
\begin{array}{rcl}
high\_req(f) & \equiv & \exists i(i > f \wedge (i \in reqs\_dest \vee i \in reqs\_up \vee i \in reqs\_dn)) \\
low\_req(f) & \equiv & \exists i(i < f \wedge (i \in reqs\_dest \vee i \in reqs\_up \vee i \in reqs\_dn)) \\
need\_go\_up & \equiv & high\_req(floor) \wedge (going\_up \vee \neg low\_req(floor)) \\
need\_go\_dn & \equiv & low\_req(floor) \wedge (\neg going\_up \vee \neg high\_req(floor)) \\
need\_stop(f) & \equiv & \vee\ f \in reqs\_dest \\
& & \vee\ going\_up \wedge f \in reqs\_up \\
& & \vee\ \neg going\_up \wedge f \in reqs\_dn \\
& & \vee\ f \in reqs\_dn \wedge \neg high\_req(f) \\
& & \vee\ f \in reqs\_up \wedge \neg low\_req(f)
\end{array}
$$

Die Definitionen von $need\_go\_up$ und $need\_go\_dn$ sind so gewählt, dass der Aufzug nur dann wendet, wenn in der aktuellen Richtung keine Fahrtwünsche mehr vorliegen.

Schließlich haben wir noch eine Gruppe von Systemaktionen zur Steuerung der Tür (am aktuellen Stockwerk). Beim vollständigen Öffnen der Tür werden Fahrtwünsche für das aktuelle Stockwerk und die aktuelle

(genauer: voraussichtlich nächste) Richtung gelöscht.

$$
\begin{array}{rcl}
exec\, door\_open & \rightarrow & \wedge\ door = opening \wedge door' = open \\
& & \wedge\ reqs\_dest' = remove(reqs\_dest, floor) \\
& & \wedge\ \textbf{if}\ going\_up \vee \neg low\_req(floor)) \\
& & \quad \textbf{then}\ reqs\_up' = remove(reqs\_up, floor)\ \textbf{else}\ reqs\_up' = reqs\_up \\
& & \wedge\ \textbf{if}\ \neg going\_up \vee \neg high\_req(floor) \\
& & \quad \textbf{then}\ reqs\_dn' = remove(reqs\_dn, floor)\ \textbf{else}\ reqs\_dn' = reqs\_dn \\
& & \wedge\ \text{UNCHANGED}(floor, stopped, going\_up) \\
\text{enabled}_{door\_open} & \equiv & door = opening \\
exec\, close\_door & \rightarrow & \wedge\ door = open \wedge door' = closing \\
& & \wedge\ \text{UNCHANGED}(reqs\_up, reqs\_dn, reqs\_dest, floor, going\_up, stopped) \\
\text{enabled}_{close\_door} & \equiv & door = open \\
exec\, door\_closed & \rightarrow & \wedge\ door = closing \wedge door' = closed \\
& & \wedge\ \text{UNCHANGED}(reqs\_up, reqs\_dn, reqs\_dest, floor, going\_up, stopped) \\
\text{enabled}_{door\_closed} & \equiv & door = closing \\
exec\, reopen\_door & \rightarrow & \wedge\ stopped \wedge (door = closing \vee door = closed) \\
& & \wedge\ door' = opening \\
& & \wedge\ \text{UNCHANGED}(reqs\_up, reqs\_dn, reqs\_dest, floor, going\_up, stopped) \\
\text{enabled}_{reopen\_door} & \equiv & stopped \wedge (door = closing \vee door = closed)
\end{array}
$$

Einige Entwurfsentscheidungen, die dieser Modellierung zu Grunde liegen, sind diskussionswürdig. Zum Beispiel kann es vorkommen, dass der Aufzug auf der Fahrt nach oben anhält, um einen Fahrtwunsch nach unten zu befriedigen (da momentan weiter oben kein Fahrtwunsch ansteht), aber dann doch nach oben weiterfährt, weil mittlerweile weiter oben ein Fahrtwunsch eingetroffen ist. In "realen Projekten" stellt man häufig fest, dass die "Kundenanforderungen" nicht genau genug spezifiziert sind. Um zu gewährleisten, dass die Anforderungen erfüllt werden, müssen formale Beschreibungen validiert werden, zum Beispiel durch Simulation. Erst dann ist es sinnvoll mit Verifikation zu beginnen.

c) Beweisen Sie folgende Aussagen für die Aufzugsteuerung:

1. Wenn der Aufzug in Bewegung ist, liegen Fahrtwünsche in der aktuellen Richtung vor.
   Lösung: Wir verstärken die Aussage um die Behauptung, dass der Aufzug steht, wann immer die Tür nicht geschlossen ist:

$$
\begin{array}{rcll}
I_1 & \equiv & \neg stopped \rightarrow & \wedge\ going\_up \rightarrow high\_req(floor) \\
& & & \wedge\ \neg going\_up \rightarrow low\_req(floor) \\
& & & \wedge\ door = closed
\end{array}
$$

   Wir benutzen (natürlich) die Regel (inv').

(1)	$start \rightarrow I_1$	(taut)
(2)	$exec\ req\_up_i \wedge high\_req(floor) \rightarrow \bigcirc high\_req(floor)$	(data)
(3)	$exec\ req\_up_i \wedge low\_req(floor) \rightarrow \bigcirc low\_req(floor)$	(data)
(4)	$exec\ req\_up_i \rightarrow \text{UNCHANGED}(stopped, door)$	(taut)
(5)	I_1 **invof** req_up_i	(2)–(4)
(6)	I_1 **invof** req_dn_i	(genauso)
(7)	I_1 **invof** req_dest_i	(genauso)
(8)	$exec\ start\_lift \wedge going\_up' \rightarrow \bigcirc high\_req(floor)$	(data)
(9)	$exec\ start\_lift \wedge \neg going\_up' \rightarrow \bigcirc low\_req(floor)$	(data)
(10)	$exec\ start\_lift \rightarrow door' = closed$	(pred)
(11)	$exec\ start\_lift \rightarrow \bigcirc I_1$	(8)(9)(10)
(12)	$high\_req(floor) \wedge \neg need\_stop(succ(floor)) \rightarrow high\_req(succ(floor))$	(data)
(13)	$low\_req(floor) \wedge \neg need\_stop(prev(floor)) \rightarrow low\_req(prev(floor))$	(data)
(14)	$exec\ pass\_floor \rightarrow door' = door$	(taut)
(15)	I_1 **invof** $pass\_floor$	(12)(13)(14)
(16)	$exec\ stop\_lift \rightarrow \bigcirc stopped$	(taut)
(17)	$exec\ stop\_lift \rightarrow \bigcirc I_1$	(16)
(18)	$I_1 \wedge exec\ door\_open \rightarrow stopped \wedge \bigcirc stopped$	(data)
(19)	I_1 **invof** $door\_open$	(18)
(20)	$I_1 \wedge close\_door \rightarrow stopped \wedge \bigcirc stopped$	(data)
(21)	I_1 **invof** $close\_door$	(20)
(22)	$I_1 \wedge door\_closed \rightarrow stopped \wedge \bigcirc stopped$	(data)
(23)	I_1 **invof** $door\_closed$	(22)
(24)	$I_1 \wedge reopen\_door \rightarrow stopped \wedge \bigcirc stopped$	(data)
(25)	I_1 **invof** $reopen\_door$	(24)
(26)	I_1	(inv')(1)(5)(6)(7)(11)(15)(17)(19)(21)(23)(25)

2. Der Aufzug hält an einem Stockwerk nur dann an, wenn eine Anforderung für dieses Stockwerk vorliegt.
   Lösung: Wir formalisieren die Aussage durch die Formel

$$
\begin{array}{rcl}
I_2 & \equiv & \text{enabled}_{stop\_lift} \rightarrow \\
 & & \textbf{if } going\_up \\
 & & \textbf{then } succ(floor) \in reqs\_up \vee succ(floor) \in reqs\_dn \vee succ(floor) \in reqs\_dest \\
 & & \textbf{else } pred(floor) \in reqs\_up \vee pred(floor) \in reqs\_dn \vee pred(floor) \in reqs\_dest
\end{array}
$$

   Zum Beweis benutzen wir die Definition von $\text{enabled}_{stop\_lift}$ und zeigen

$$need\_stop(f) \rightarrow f \in reqs\_up \vee f \in reqs\_dn \vee f \in reqs\_dest$$

   was unmittelbar prädikatenlogisch aus der Definition von $need\_stop$ folgt.

Aufgabe 10-2 **Dining Philosophers** (keine Abgabe)

Die dinierenden Philosophen sind ein klassisches Modellierungsproblem von E. Dijkstra, gestellt als Klausuraufgabe unter dem Namen „dining quintuple“ und von T. Hoare in „dining philosophers“ umgetauft.

An einem runden Tisch sitzen $n > 1$ Philosophen. Philosophen verbringen ihr Leben fortwährend mit Essen und Denken (was beides nicht zur gleichen Zeit stattfinden kann). Vor jedem Philosophen steht eine Schale Reis; zwischen je zwei Philosophen liegt ein Stäbchen. Um essen zu können, benötigt der Philosoph sowohl sein rechtes als auch sein linkes Stäbchen. Dabei gilt:

- Will ein Philosoph essen, so nimmt er zuerst sein rechtes, dann sein linkes Stäbchen.

- Wird ein Stäbchen vom jeweiligen Nachbarn benutzt, so wartet der Philosoph, bis der Nachbar das Stäbchen freigibt.
- Nach dem Essen legt der Philosoph ohne Unterbrechung beide Stäbchen ab und denkt wieder.
- Ein aufgenommenes Stäbchen legt ein Philosoph nicht wieder zurück, bis er das Essen beendet hat.
- Essen und Denken kann beliebig lange andauern, das Aufnehmen und Ablegen der Stäbchen hingegen wird schnellstmöglich ausgeführt.

a) Modellieren Sie die n Philosophen in einem rlSTS Γ. Geben Sie für jede Aktion von Γ geeignete Ausführbarkeitsbedingungen an, und beschreiben Sie die Wirkung der Aktionen durch Formeln von $\mathcal{L}_{\mathrm{TL}\Gamma}$.

Lösung: Wir definieren ein lrSTS Γ_{DP} über der Signatur und dem Standardmodell der natürlichen Zahlen:

$$\Gamma_{DP}(SIG_{NAT}, \mathsf{N}) = (X, V, Z, T, start, Act, \mathcal{E})$$

wobei X für jeden Philosophen einen Zustandszähler enthält:

$$X = X_{Nat} = \{p_i \mid i \in \{1, \ldots, n\}\}$$

und V eine Menge von Individuensymbolen V', die den Status der Stäbchen beschreiben:

$$V' = \{f_i \mid i \in \{1, \ldots, n\}\}$$

Als Aktionen definieren wir uns für jeden Philosophen sechs Zustandsübergänge:

$$Act = \{think_i, hungry_i, take_{i,r}, take_{i,l}, eat_i, drop_i \mid i \in \{1, \ldots, n\}\}$$

Dabei soll $hungry_i$ das „Eintreten des Hungers“ für Philosoph i darstellen, wonach mit den beiden $take_{i,x}$-Transitionen jeweils ein Stäbchen aufgenommen wird. Bei $drop_i$ werden beide Stäbchen gleichzeitig zurückgelegt.

Insgesamt ergibt sich damit V als

$$V = V' \cup \{exec\lambda \mid \lambda \in Act\}$$

Als Zustände definieren wir

$$\begin{aligned} Z = \{\eta : X \cup V \to \{\mathsf{tt}, \mathsf{ff}\} \cup \mathbb{N} \mid & \eta \text{ zulässig}, \\ & \eta(f_i) \in \{\mathsf{tt}, \mathsf{ff}\}, \eta(p_i) \in \{1, \ldots, 4\} \text{ für alle } 1 \leq i \leq n, \\ & \eta(exec_\lambda) \in \{\mathsf{tt}, \mathsf{ff}\}, \\ & \eta(exec_\lambda) = \mathsf{tt} \text{ für maximal ein } \lambda \in Act\} \end{aligned}$$

Transitionen sind in unserem Falle dann ausführbar, wenn sich der Philosoph im passenden Zustand befindet, und ggf. die benötigten Stäbchen frei ist. Wir definieren $i \ominus 1 \equiv (i + n - 1) \mod n$.

$$\begin{aligned} \mathcal{E} = & \{enabled_\lambda \mid \lambda \in Act\} \text{ mit} \\ & enabled_{think_i} \equiv p_i = 1 \\ & enabled_{hungry_i} \equiv p_i = 1 \\ & enabled_{take_{i,r}} \equiv p_i = 2 \wedge f_i \\ & enabled_{take_{i,l}} \equiv p_i = 3 \wedge f_{i \ominus 1} \\ & enabled_{eat_i} \equiv p_i = 4 \\ & enabled_{drop_i} \equiv p_i = 4 \end{aligned}$$

Die Zustandsübergänge sind somit durch die Transitionen schon fast vollständig definiert; es fehlt nur noch deren (offensichtliche) Wirkung. Dabei ist dafür Sorge zu tragen, daß wir sogenannte „frame conditions“ einhalten; dies bedeutet, daß wir nicht nur die Werte der Variablen in η' angeben müssen, die verändert werden, sondern auch die, die gleich bleiben. Um uns lästige Schreibarbeit zu sparen, definieren wir eine Abkürzung

$\mathrm{UC}(v_1, \ldots, v_n) \equiv \eta'(v_1) = \eta(v_1) \wedge \ldots \wedge \eta'(v_n) = \eta(v_n)$, sowie $p_{\neq i} \equiv (p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n)$, $f_{\neq i} = (f_1, \ldots, f_n) \setminus \{f_i, f_{i\ominus 1}\}$ und $f = (f_1, \ldots, f_n)$.

$$
\begin{aligned}
T' = \{(\eta,\eta') \in Z \times Z \mid\ & \\
& \eta(exec_{think_i}) = \mathsf{tt} \rightarrow \mathrm{UC}(p_{\neq i}, p_i, f) \\
\wedge\, & \eta(exec_{hungry_i}) = \mathsf{tt} \rightarrow \eta'(p_i) = 2 \wedge \mathrm{UC}(p_{\neq i}, f) \\
\wedge\, & \eta(exec_{take_{i,r}}) = \mathsf{tt} \rightarrow \eta'(p_i) = 3 \wedge \eta'(f_i) = \mathsf{ff} \wedge \mathrm{UC}(p_{\neq i}, f_{\neq i}, f_{i\ominus 1}) \\
\wedge\, & \eta(exec_{take_{i,l}}) = \mathsf{tt} \rightarrow \eta'(p_i) = 4 \wedge \eta'(f_{i\ominus 1}) = \mathsf{ff} \wedge \mathrm{UC}(p_{\neq i}, f_{\neq i}, f_i) \\
\wedge\, & \eta(exec_{eat_i}) = \mathsf{tt} \rightarrow \mathrm{UC}(p_{\neq i}, p_i, f) \\
\wedge\, & \eta(exec_{drop_i}) = \mathsf{tt} \rightarrow \eta'(p_i) = 1 \wedge \eta'(f_i) = \mathsf{tt} \wedge \eta'(f_{i\ominus 1}) = \mathsf{tt} \wedge \mathrm{UC}(p_{\neq i}, f_{\neq i})\}
\end{aligned}
$$

$$T = tot(T')$$

Letztendlich definieren wir den (einzigen) Startzustand, wo jeder Philosoph denkt und alle Gabeln unbenutzt sind:

$$start \equiv \bigwedge_{i \in \{1,\ldots,n\}} p_i = 1 \wedge f_i$$

b) Zeigen Sie: Es gibt einen Ablauf, in dem kein Philosoph jemals isst.

Lösung: Die „berühmte Verklemmung“ kommt zustande, wenn alle Philosophen gleichzeitig hungrig werden. Ein möglicher Ablauf für $n = 3$ einer solchen Verklemmung ist wie folgt (der Zustand wird als $[p_1, \ldots, p_n, f_1, \ldots, f_n]$ angegeben):

$$
\begin{aligned}
& [1,1,1,\mathsf{tt},\mathsf{tt},\mathsf{tt}] \overset{hungry_1}{\longrightarrow} [2,1,1,\mathsf{tt},\mathsf{tt},\mathsf{tt}] \overset{hungry_2}{\longrightarrow} [2,2,1,\mathsf{tt},\mathsf{tt},\mathsf{tt}] \overset{take_{2,l}}{\longrightarrow} [2,3,1,\mathsf{tt},\mathsf{ff},\mathsf{tt}] \\
\overset{hungry_3}{\longrightarrow} & [2,3,2,\mathsf{tt},\mathsf{ff},\mathsf{tt}] \overset{take_{3,l}}{\longrightarrow} [2,3,3,\mathsf{tt},\mathsf{ff},\mathsf{ff}] \overset{take_{1,l}}{\longrightarrow} [3,3,3,\mathsf{ff},\mathsf{ff},\mathsf{ff}] \overset{nil_\Gamma}{\longrightarrow} \ldots
\end{aligned}
$$

Abhilfe kann z.B. dadurch geschaffen werden, daß der ein einzelner Philosoph mit seiner rechten Gabel beginnt (interessanterweise schließt dies weitere Verklemmungen bereits aus), oder dadurch, daß Philosophen, die ihr zweites Stäbchen nicht aufnehmen können, wieder in den „Denk“-Zustand zurückkehren und das bereits aufgenommene Stäbchen wieder ablegen. In beiden Varianten ist es aber ohne zusätzliche Anforderungen an das System nicht möglich, ein sprichtwörtliches „Verhungern“ eines Philosophen auszuschließen.

Abgabe: Nach den Ferien: Mittwoch, den 10.1.2007, vor der Übung.