# Theorie kommunizierender Systeme
# Sequentielle Prozesse und Bisimulation

Ehsan Gholamsaghaee
Betreuer: Andreas Rossberg

Universität des Saarlandes
Department 6.2 - Computer Science
Fakultät für Naturwissenschaft und Technologie I

20. Oktober 2004

# Inhaltsverzeichnis

Automaten beshtehen unter anderem aus einer Menge von Übergänge, die von einem Zustand $q$ zu einem anderen Zustand $q'$ führen. Wenn man jeden Übergang als ein Prozess betrachtet, kann man jeden Automat als ein sequenzieller Prozess (Prozesse, die nacheinander laufen) beschreiben. In diesem Artikel sprechen wir über nicht deterministische sequentielle Prozesse. Zusätzlic wollen wir wiessen, wann zwei solche Prozesse äquivalent sind, anders gesagt, wann zwei sequenzielle Prozesse sich gleich verhalten. Um diese Sachverhalten testen zu können, definieren wir eine Form des Äquivalentzrelation, die wir Bisimulation nennen werden.

# 1 Labelled transition systems

Sequentielle Prozesse können durch beschriftetes Transitionssystem ($LTS$ Labelled Transition System) beschrieben werden. Um dieses zu definieren brauchen wir erst ein paar Notationen. Als Erste definieren wir eine unendliche Menge $\mathcal{N}$ von Namen, deren Elemnte wir inder Regel mit kleinen Buchstaben bezeichnen; in Beispiele werden wir aber hilfreiche Namen wie z.B. ‚tea' oder ‚2p' benutzen. Mit $\overline{\mathcal{N}} = \{\overline{a} | a \in \mathcal{N}\}$ bezeichnen wir die Menge von co-Namen, wie z.B. $\overline{tea}$. Wir nehmen an, dass $\mathcal{N}$ und $\overline{\mathcal{N}}$ disjunkte Mengen sind, und ihre Vereinigung ($\mathcal{N} \cup \overline{\mathcal{N}}$) bezeichen wir mit $\mathcal{L}$ als die Menge von Labels. Vorläufig setzen wir $Act$ die Menge von Actionen gleich wie $\mathcal{L}$ und ihre Element bezeichen wir mit $\alpha, \beta, \ldots$ .

**Definition 1.1 (Labelled Transition System)** Ein $LTS$ über der Menge $Act$ ist ein Tupel $(\mathcal{Q}, \mathcal{T})$ wobei

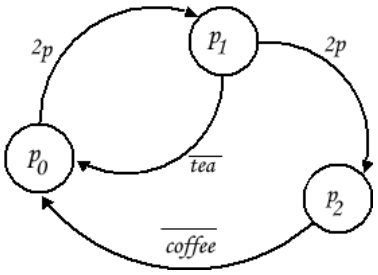
- $\mathcal{Q}$ ist eine Menge der Zuständen und
- $\mathcal{T} \subseteq (\mathcal{Q} \times Act \times \mathcal{Q})$ ist eine tenäre Relation, die als **Transitionsrelation** bezeichnet wird.

Sei $(q, \alpha, q') \in \mathcal{T}$, so schreiben wir $q \xrightarrow{\alpha} q'$, wobei $q$ die Quelle und $q'$ das Ziel ist. Sei $q \xrightarrow{\alpha_1} q_1 \ \ldots \xrightarrow{\alpha_n} q_n$, dann ist $q_n$ die Ableitung von $q$.

Ein $LTS$ kann also als ein Automat ohne Start- und Endzustände betrachtet werden. Wir dürfen davon ausgehen, dass wir jeder Zustand als Startzustand

wählen können. Alle Automaten, die wir in dieser Weise erhalten, sind auf dem selben $LTS$ basiert.

**Beispiel 1.2** Wir btrachten folgende Getränkeautomat und definieren wir eine $LTS$ das unseren Automat enthält.

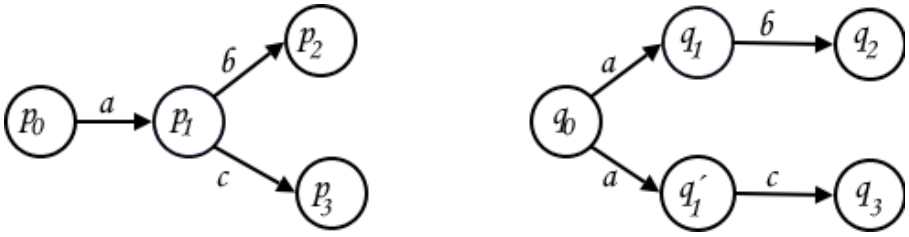



$\mathcal{Q} = \{p_0, p_1, p_2\}$ und
$\mathcal{T} = \{(p_0, 2p, p_1), (p_1, \overline{tea}, p_0), (p_1, 2p, p_2), (p_2, \overline{coffee}, p_0)\}$ ■

# 2 Starke Simulation

Wann sollen zwei Zustände in einem $LTS$ als äquivalent betrachtet werden?

**Beispiel 2.1**




$A$ $B$

Seien $A$ und $B$ zwei Getränkeautomaten. Wie es deutlich zu sehen ist, ist $A$ mächtiger als $B$ (obwohl die beide die gleiche Sprache akzeptieren), denn bei $B$ ist nach dem Übergang von $a$ festgelegt, dass entweder nur $b$ oder $c$ akzeptiert wird; dagegen ist bei $A$ sowohl $b$ als auch $c$ als Übergang möglich.■
Um diesen Sachverhalten näher beschreiben zu können, wird folgende Definition eingeführt:

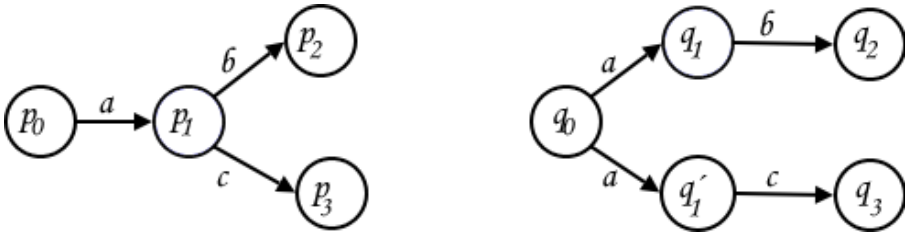
**Definition 2.2 (Starke Simulation)** Sei$(\mathcal{Q}, \mathcal{T})$ ein $LTS$ und sei $S$ eine binäre Relation über $\mathcal{Q}$. Dann wird $S$ eine starke Simulation über $(\mathcal{Q}, \mathcal{T})$ genannt, wenn für alle $pSq$ gilt:

$$\textit{wenn } p \xrightarrow{\alpha} p' \textit{ dann existiert ein } q' \in \mathcal{Q}, \textit{ so dass } q \xrightarrow{\alpha} q' \textit{ und } p'Sq'$$

Wir sagen, dass $q$ $p$ stark simuliert, wenn es eine starke Simulation $S$ gibt, so dass $pSq$.

**Bemerkung**: Wir definieren die Simulation über die Zustände von einem einzigen $LTS$ nicht zwischen den Zuständen von zwei $LTSen$. Oft ist es der Fall, dass ein $LTS$ zwei oder mehrere Automaten enthält wie in Beispiel 2.1

**Beispiel 2.3** Fortsetzung des Beispiels 2.1:




Man zeigt, dass $p_0$ $q_0$ stark simuliert, indem man $S$ wie folgendes definiert

$$S = \{(q_0, p_0), (q_1, p_1), (q_1', p_1), (q_2, p_2), (q_3, P_3), \}$$

Wie es deutlich zu sehen ist, gibt es für jedes Paar $(q, p) \in S$ und die Transition $q \xrightarrow{\alpha} q'$ eine zugehörige Transition $p \xrightarrow{\alpha} p'$. ■

**Übung 2.4** Zeigen Sie bei dem Beispiel 2.3, dass es keine starke Simulation gibt, die das Paar $(p_1, q_1)$ enthält.

Sei $\mathcal{R}$ eine starke Simulation mit $p_1 \mathcal{R} q_1$. Da $p_1 \xrightarrow{c} p_3$ muss es ein $q_i$ geben mit $q_1 \xrightarrow{c} q_i$ und $p_3 \mathcal{R} q_i$, aber es ist überhaupt keine c-Übergang von $q_1$ möglich und damit gibt es auch keine starke Simulation $\mathcal{R}$ mit $p_1 \mathcal{R} q_1$. ■

Wir haben ein Kriterium gefunden, um zwischen unseren beiden Automaten in Beispiel 2.1 zu unterscheiden. Wir sollen jetzt Simulation in eine Äquivalenzrelation verfeinern, in der unsere zwei Getränkeautomaten nicht äquivalent sind.

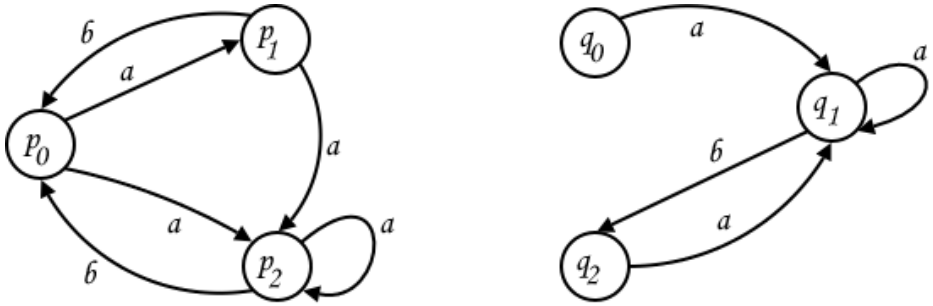
# 3 Starke Bisimulation

**Definition 3.1 (Starke Bisimulation, starke Äquivalent)** Sei $(\mathcal{Q}, \mathcal{T})$ ein $LTS$, und sei $S$ eine binäre Relation über $\mathcal{Q}$. Dann wird $S$ eine starke Bisimulation (starke Äquivalent) über $(\mathcal{Q}, \mathcal{T})$ genannt, wenn $S$ und $S^{-1}$ starke Simulationen sind.

Zwei Zustände $p$ und $q$ sind stark bisimular, wenn es eine starke Bisimulation $S$ gibt mit $pSq$.
Schreibweise: $p \sim q$; $p$ und $q$ sind äquivalent.

**Beispiel 3.2**

Wir betrachten folgendes $LTS$, das aus zwei Automaten besteht

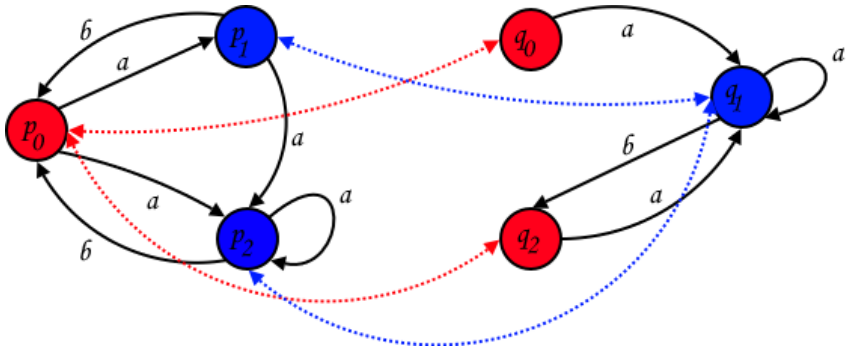



zu zeigen ist: $p_0 \sim q_0$
Um das zu zeigen definiert man $S$ wie folgt:

$$S = \{(p_0, q_0), (p_0, q_2), (p_1, q_1), (p_2, q_1)\}$$

und dann zeigt man, dass $S$ eine Bisimulation ist, da $p_0 S q_0$ folgt die Behauptung. ■

Es ist oft sinnvoll eine Bisimulation graphisch darzustellen. So können wir $S$ aus Beispiel 3.2 wie folgt darstellen:




**Übung 3.3** Zeigen Sie in Beispiel 3.2, dass $S$ eine starke Bisimulation ist.

Zu erst sollen wir zeigen, dass $S$ eine starke Simulation ist, indem wir für jedes Paar $(p_i, q_i) \in S$ testen, ob die Simulationsbedingung erfült ist. Für $(p_0, q_0)$ gilt es; $p_0 \xrightarrow{a} p_1$ und $q_0 \xrightarrow{a} q_1$ mit $(p_1, q_1) \in S$. Analog zeigt man für restliche Paare, dass es sich um eine Simulation handelt. Nun müssen wir zeigen, dass $S^{-1} = \{(q_0, p_0), (q_2, p_0), (q_1, p_1), (q_1, p_2)\}$ auch eine Bisimulation ist . Man geht genau so wie im erstem Teil vor und zeigt für jedes Paar, dass die Simulationbedingug erfühlt ist. Da $S$ und $S^{-1}$ beide stak Simulationen sind folgt, dass $S$ eine starke Bisimulation ist. ■

**Proposition 3.4**

- **(1)** $\sim$ ist eine Äquivalenzrelation d.h.
  - $p \sim p$ (reflexiv)
  - wenn $p \sim q$ dann $q \sim p$ (symmetrisch)
  - wenn $p \sim q$ und $q \sim r$ dann $p \sim r$ (transitiv)
- **(2)** $\sim$ ist selbst eine starke Bisimulation.

**Beweis**

- **(1)** Für Reflexivität genügt es zu zeigen, dass die Identitätsrelation ($Id\mathcal{Q} = \{(p,p) \mid p \in \mathcal{Q}\}$) über $\mathcal{Q}$ eine Bisimulation ist. Das ist trivial. Um Symmetrie zu zeigen, muss man zeigen, wenn $S$ eine Bisimulation ist, dann ist auch $S^{-1}$ eine Bisimulation. Das folgt aus der Definition der Bisimulation. Für Transitivität muss man zeigen, wenn $S_1$ und $S_2$ Bisimulationen sind, so ist ihre Komposition

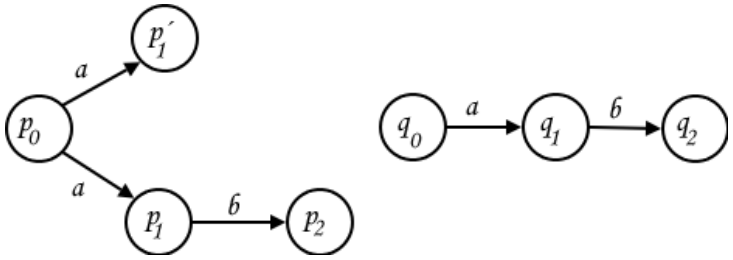
  $$S = \{(p,r) | \exists q.\ pS_1q\ und\ qS_2r\}$$

  auch eine Bisimulation. Es genügt zu zeigen, dass $S$ eine Simulation ist ($S^{-1}$ folgt Analog). Sei $(p,r) \in S$ und $p \xrightarrow{\alpha} p'$. Da es ein $q$ mit $pS_1q$ und $qS_2r$ existiert, gibt es ein $q'$ mit $q \xrightarrow{\alpha} q'$ und $p'S_1q'$, und auch ein $r'$ mit $r \xrightarrow{\alpha} r'$ und $q'S_2r'$. Daraus folgt $(p',r') \in S$.

- **(2)** Sei $p \sim q$. Dann gibt es eine starke Simulation mit $pSq$ und, wenn $p \xrightarrow{\alpha} p'$, gibt es ein $q'$ mit $q \xrightarrow{\alpha} q'$ daraus folgt $p'Sq'$ und damit $p' \sim q'$. Wir haben gezeigt, dass $\sim$ eine Simulation ist, da $\sim$ symmetrisch ist folgt analog, dass ihre Inverse auch eine Simulation ist. □

**Bemerkung**: $p \sim q$ heisst nicht „p simuliert starke q und q simuliert stark p “.

**Übung 3.5** Betrachten Sie folgende Prozesse




Zeigen Sie, dass $p_0$ simuliert stark $q_0$ und umgekehrt; aber: $p_0 \nsim q_0$

Seien $S_1$:=$\{(q_0,p_0),(q_1,p_1),(q_2,p_2)\}$ und $S_2$:=$\{(p_0,q_0),(p_1,q_1),(p_1',q_1),(p_2,q_2)\}$. Es ist einfach zu zeigen, dass $S_1$ und $S_2$ starke Simulationen sind (aber keine Bisimulation); daraus folg die erste Behauptung. Wir nehmen an: $p_0Sq_0$ für eine starke Simulation $S$. Da $p_0 \xrightarrow{a} p_1'$, muss es ein $q_i$ geben nämlich $q_1$ mit $q_0 \xrightarrow{a} q_1$ und $p_1'Sq_1$. So gilt $q_1S^{-1}p_1'$; aber für die Transition $q_1 \xrightarrow{b} q_2$ gibt es

keine passende Transition von $p_1'$. Daraus folgt $S^{-1}$ ist keine starke Simulation.

# 4 Sequentielle Prozessausdrücke

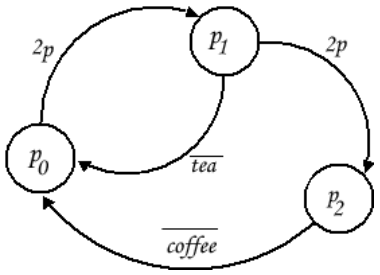
Prozessausdrücke tragen Informationen über das Verhalten und die Struktur eines Systems. Im Falle eines sequentiellen Prozessausdruckes beschreibt der Prozessausdruck genau die Transitionen eines Automaten. Um die sequentielle Prozessausdrücke zu definieren, führen wir folgende Basisnotationen ein; $Id$ ist eine unendliche Menge von Prozessbezeichnern, deren Elemente wir mit Grußbuchstaben wie z.B. $A, B, ...$ bezeichnen. Es ist aber sinnvoll in Beispiele hilfreiche Namen wie ‚Getränkeautomat'oder ‚Buffer'zu benutzen. Wir betrachten Prozesse die durch Parametern beschrieben werden können; z.B. wir schreiben $A\langle a, b, c\rangle$ um auszudrücken, dass der Prozess $A$ durch die Parametern $a, b, c$ beschrieben wird. Mit $\vec{a}$ zeigen wir eine Folge $a_1, ..., a_n$ von Namen. Wenn $\vec{a}$ und $\vec{b}$ Folgen der Namen, die Längen $n$ und $P$ ein Prozessausdruck sind, dann ist $\{\vec{a}/\vec{b}\}P$ das Resultat des Ersetzens von $a_i$ durch $b_i$ in $P$ $(1 \leq i \leq n)$. Die Menge der Namen, die in einem Prozessausdrück $P$ vorkommen, bezeichnen wir mit $fn(P)$ ('$fn$ ‚steht für frei Namen, später werden wir unsere Prozessausdrücke so erweitern, dass die auch gebundene d.h. nicht frei Namen bekommen, und dann enthält $fn(P)$ nur frei Namen, die in $P$ vorkommen).

**Definition 4.1 (Sequentiellen Prozessausdrücken)** Die Menge $\mathcal{P}^{seq}$ von sequentiellen Prozessausdrücken wird durch folgenden Syntax definiert:

$$P ::= A\langle a_1, ..., a_n\rangle | \sum_{i \in I} \alpha_i.P_i$$

Wobei $I$ eine endliche Indexmenge ist.

**Beispiel 4.2** Wir betrachten nochmal unseren Getränkeautomat.




Den können wir wie folgt als eine Prozessausdrück zusammenfassen:
Aktionen : $2p, \overline{tea}, \overline{coffee}$
Prozess : $B_1$

$$B_1 =_{def} 2p.(\overline{tea}.B_1 \ + \ 2p.\overline{coffee}.B_1)$$

■

Wir nennen die leere Summe ($\sum_{i \in \emptyset} P_i$) Null und gehen davon aus, dass jeder Prozessbezeichner $A$ durch folgende Gleichung definiert werden kann

$$A(\vec{a}) =_{def} P_A$$

Wobei $P_A$ ist eine Summe und $\vec{a} = a_1, ..., a_n$ enthält $fn(P_A)$. Nun $A\langle\vec{b}\rangle$ hat die gleiche Bedeutung wie $\{\vec{b}/\vec{a}\}P_A$, wenn $\vec{b}$ und $\vec{a}$ die gleiche Länge haben. Diese Absicht erfassen wir durch folgende Definition:

**Definition 4.3 (Strukturelle Kongruenz)** Zwei sequentielle Prozesse $P$ und $Q$ sind strukturell kongruent ($P \equiv Q$) , wenn wir den einen in den anderen umwandeln können, indem wir die Auftreten von $A\langle\vec{b}\rangle$ durch $\{\vec{b}/\vec{a}\}P_A$ oder umgekehrt ersetzen, wenn $A(\vec{a}) =_{def} P_A$

**Beispiel 4.4** Seien

$$A_0(a,b,c) =_{def} a.A_1\langle a,b,c\rangle$$
$$A_1(a,b,c) =_{def} \overline{b}.A_0\langle a,b,c\rangle + a.A_2\langle a,b,c\rangle$$
$$A_2(a,b,c) =_{def} \overline{c}.A_0\langle a,b,c\rangle$$

Wir zeigen:

$$a.A_1\langle a,b,c\rangle \equiv a.(\overline{b}.A_0\langle a,b,c\rangle + a.A_2\langle a,b,c\rangle)$$

Es genügt, dass man $A_1\langle a,b,c\rangle$ durch $\overline{b}.A_0\langle a,b,c\rangle + a.A_2\langle a,b,c\rangle$ ersetzt. ■

Hier lässt sich ein $LTS$ definieren welches sequentielle Prozesse beinhaltet.

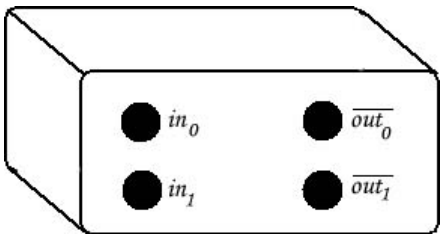
**Definition 4.5 ($LTS$ von sequentiellen Prozessen)** Ein $LTS$ von sequentiellen Prozessen über einer Menge $Act$ ist durch die Menge $\mathcal{P}^{seq}$ von Zuständen definiert und hat folgende Transitionen:

$$wenn\ P \equiv \sum_{i\in I} \alpha_i.P_i,\ dann\ gibt\ es\ für\ alle\ j \in I, P \xrightarrow{\alpha_i} P_j.$$

Im folgenden Abschnitt geben wir paar Beispiele von sequentiellen Prozessen.

# 5 Beispiele

**Beispiel 5.1 (Boolscher Buffer)**




$Buff^{(2)}$

Wir wollen nun einen boolschen Puffer konstruieren, welcher in der Lage ist die Sequenz $s \in \{0,1,00,01,10,11\}$ zu speichern und in der gleichen Reihenfolge auszugeben. Als Parameter bekommt er die Namen $in_i$, $out_i$ $(i \in \{0,1\})$. Dazu benötigt man sieben Prozessidentifiers: $Buff^{(2)}, Buff_0^{(2)}, Buff_1^{(2)}, Buff_{00}^{(2)}, ...,$ die durch folgende Prozessausdrücke festgelegt sind:

$$Buff^{(2)} =_{def} \sum_{i\in\{0,1\}} in_i.Buff_i^{(2)} \ (nur\ Einlesen\ möglich)$$

$$Buff_i^{(2)} =_{def} \ \overline{out_i}.Buff^{(2)} + \sum_{j\in\{0,1\}} in_j.Buff_{ji}^{(2)} \ (Einlesen\ oder\ Ausgeben)$$

$$Buff_{ij}^{(2)} =_{def} \ \overline{out_j} \ (nur\ Ausgeben\ möglic).$$

Später werden wir sehen, wie die Puffer kombiniert werden können, um Puffer von der größeren KapazitÄt zu bilden, und wie $Buff^{(2)}$ in einfachere Prozesse zerlegt werden kann. ■

**Übung 5.2 (a)** Schreiben Sie eine analoge Definition eines $Buffer^{(3)}$ von Kapazität drei. **(b)** Ändern Sie die definierenden Gleichungen des $Buffer^{(2)}$, um zu erlauben, dass die gespeicherten Werte in jedem möglichem Auftrag ausgegeben werden. **(c)** Zeichnen Sie das Übergangsdiagramm von $Buffer^{(2)}$.

**(a)**

$$Buff^{(3)} =_{def} \sum_{i\in\{0,1\}} in_i.Buff_i^{(3)}$$

$$Buff_j^{(3)} =_{def} \ \overline{out_j}.Buff^{(3)} + \sum_{i\in\{0,1\}} in_i.Buff_{ij}^{(3)}$$

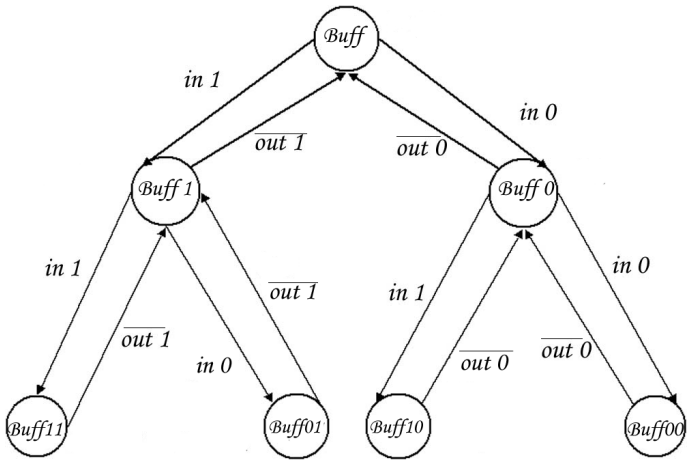
$$Buff_{jk}^{(3)} =_{def} \ \overline{out_k}.Buff_j^{(3)} + \sum_{i\in\{0,1\}} in_i.Buff_{ijk}^{(3)}$$

$$Buff_{ijk}^{(3)} =_{def} \ \overline{out_k}.Buff_{ij}^{(3)}$$

**(b)** Nur die Definition von $Buff_{ij}^{(2)}$ muss geändert werden:

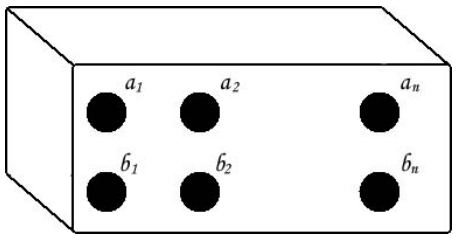
$$Buff_{ij}^{(2)} =_{def} \ \overline{out_i}.Buff_j^{(2)} + \overline{out_j}.Buff_i^{(2)}$$

**(c)**




■

**Beispiel 5.3 Scheduler** Nehmen Sie an, dass wir eiene Folge von Agenten $P_1, ..., P_n$ haben. Jeder Agent möchte eine Aufgabe wiederholt durchführen, und ein Scheduler wird angefordert, um sicherzugehen, dass sie die Aufgabe im zyklischen Auftrag anfangen und mit $P_1$ anfangen. Die unterschiedlichen Aufgabe-Durchführungen können mit in der Zeit sich decken - z.B. $P_2$ kann anfangen bevor $P_1$ fertig ist - aber der Scheduler muss sichergehen, dass jedes Agent eine Durchführung beendet hat, bevor der mit einer neuen anfängt. Nehmen Sie an, dass $P_i$ um eine Aufgabe bittet, indem er die Taste $a_i$ auf dem Scheduler betätigt, und signalisiert die Erfühlung der Aufgabe durch Betätigen von $b_i$.




Informel ist die Spezifikation von Scheduler wie folgt:

- (1) der muss $a_1, ..., a_n$ erfordern, dass die zyklisch auftreten und mit $a_1$ anfangen;

- (2) für jedes $i$ muss der fordern, dass $a_i$ und $b_i$ wechselnd auftreten und mit $a_i$ anfangen;
- (3) der muss zulassen, dass jede seiner Tasten zur jeder Zeit betätigt werden können. (1) und (2) sollen nicht verletzt werden.

Wir spezifizieren den Scheduler in sequentielle Prozess $Sched_{i,X}$, wobei $1 \leq i \leq n$ und $X \subseteq \{1, ..., n\}$. Der Parameter $i$ zeigt an, dass es $P_i$ ist, der die nächste Aufgabe einführen will, und $X$ ist die menge aller Agenten, die gerade beschäftigt sind. $Sched_{i,X}$ representiert das Scheduler in dem Zustand, wo:

- nur $P_i$ kann mit einer Aufgabe anfangen (vorausgesetzt, dass $i \notin X$);
- alle Agente $\{P_j : j \in X)$ führen grade ihre Aufgabe durch und können damit fertig werden.

Wir definieren Scheduler durch folgende Gleichungen:

$$Scheduler =_{def} \ Sched_{1,\emptyset}$$

$$Sched_{i,X} =_{def} \begin{cases} \sum_{j \in X} b_j.Sched_{i,X-j} & (i \in X) \\ \sum_{j \in X} b_j.Schedi, X - j + a_i.Sched_{i+1,X \cup i} & (i \notin X) \end{cases}$$

wobei $i+1$ als modulo $n$ zu interpretieren ist. Wir schreiben $X \cup i$ und $X - i$ anstatt $X \cup \{i\}$ und $X - \{i\}$.

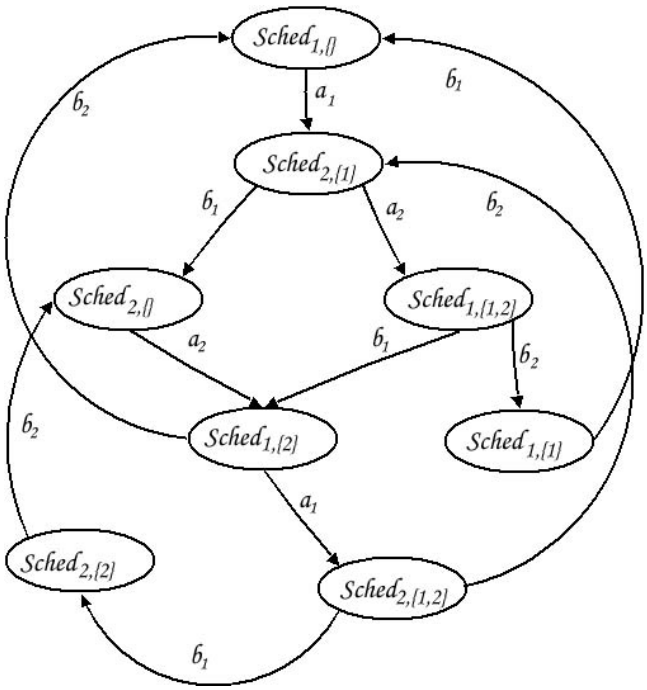
**Übung 5.4 (a)** Behaupten Sie, dass Scheduler nie zum Stillstand kommt, d.h. er ermöglicht immer mindestens eine Action. **(b)** Zeichnen Sie das Übergansdiagramm von Scheduler wenn $n = 2$ **(c)** was ist, wenn wir $Sched_{i,X}$ wie folgt definieren:

$$Sched_{i,X} =_{def} \sum_{j \in X} b_j.Sched_{i,X-j} + a_i.Sched_{i+1,X \cup i}$$

**(a)** Wir können zeigen, dass der Scheduler nicht zum Stillstand kommt indem wir einen zufälligen Zustand $Sched_{i,X}$ betracheten. Wenn $X = \emptyset$, dann ist die Action $a_i$ möglich. Wenn $j \in X$ für ein $j$, bedeutet das $j$ führt seine Aufgabe durch. In dem Fall kann $j$ seine Aufgabe Beenden, d.h. $b_j$ ist möglich. In beiden Fällen ist eine Action möglich daraus folgt, dass es keinen

Stillstand gibt.

**(b)**




**(c)** Wenn wir die Änderung vornehmen, kann ein Prozess eine neue Aufgabe beginnen, bevor er eine frühere Aufgabe beendet hat.

■

# Literaturverzeichnis

Robin Milner, „Communicating and Mobile Systems: the Pi-Calculus “ Cambridge University Press 1999 (Kapitel 3)