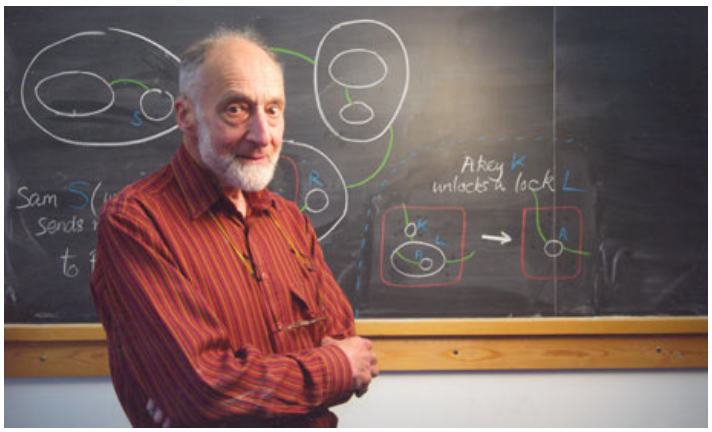
Kapitel 10
Nebenläufige Systeme

Abschnitt 10.1
Einführung

"Concurrency is ubiquitous"

(Nebenläufigkeit ist überall)

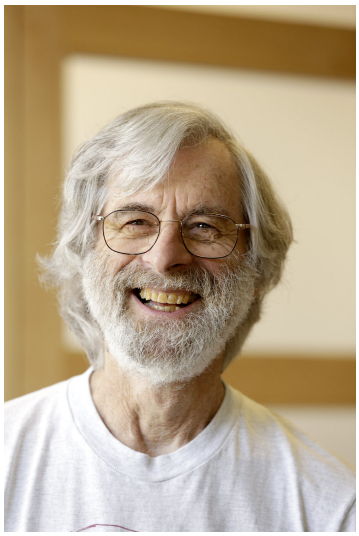
Robin Milner, 1993



Robin Milner
(1934 – 2010)

ACM Turing Award 1991

Robin Milner, *Elements of Interaction (Turing Award Lecture)*,
Communications of the ACM 36:79–89, 1993.



Leslie Lamport
(1941*)

ACM Turing Award 2013

For fundamental contributions to the theory and practice of distributed and concurrent systems, notably the invention of concepts such as causality and logical clocks, safety and liveness, replicated state machines, and sequential consistency.

Sequentielle Prozesse

- Unsere bisherigen Berechnungsmodelle (endliche Automaten, Kellerautomaten, Turingmaschinen) sind Modelle für sequentielle Berechnungen (oder Prozesse).
- All diese Modelle beschreiben Prozesse, die eine Eingabe erhalten, diese verarbeiten, und daraus eine Ausgabe berechnen.
- Abstrakt lässt sich die Semantik sequentieller Programme also als Funktion

  $$\text{Eingabe} \longrightarrow \text{Ausgabe}$$

  beschreiben.

Beispiele sequentieller Prozesse

1. DFA für einen regulären Ausdruck
   Eingabe: Wort w
   Ausgabe: Akzeptiere / verwerfe.
2. Parser für eine kontextfreie Grammatik
   Eingabe: Wort w
   Ausgabe: Ableitungsbaum oder Fehlermeldung
3. Rendering von Webseiten
   Eingabe: HTML-File
   Ausgabe: Bildschirmdarstellung

Nebenläufigkeit

Die Realität ist komplizierter.

Prozesse laufen nicht unabhängig und nacheinander ab, sondern sehr häufig gleichzeitig und mit Wechselwirkungen untereinander.

Wir bezeichnen solche Prozesse als nebenläufig.

Beispiele nebenläufiger Prozesse

1. Verschiedenen Programme, die gleichzeitig auf einem Computer laufen (etwa in verschiedenen Fenstern auf einem gemeinsamen Desktop, aber auch im Hintergrund).
2. Parallele Programme, die auf mehreren Prozessoren mit Zugriff auf einen gemeinsamen Speicher laufen.
3. Verschiedene Threads in einem Java-Programm.
4. Eine Aufzugsteuerung, die mit einem Alarmsystem interagiert.
5. Ein Bankautomat, der mit einem Kunden und einem zentralen Server der Bank interagiert.

Kompositionale Semantik

Das Verhalten von Systemen nebenläufiger Prozesse möchte man in der Regel aus dem Verhalten der einzelnen Komponenten ableiten.

Man sagt: Wir wollen nebenläufigen Systemen eine kompositionale Semantik geben.

Für sequentielle Systeme ist es in der Regel leicht, eine kompositionale Semantik anzugeben. Beispielsweise kann man das Verhalten der Hintereinanderausführung von zwei Programmen leicht aus dem Verhalten der beiden Programme ableiten.

Beispiel 10.1 I

Programme

$$P_1: \quad x \leftarrow y + 2$$
$$P_2: \quad x \leftarrow y; x \leftarrow x + 1; x \leftarrow x + 1$$
$$Q: \quad x \leftarrow 0; y \leftarrow 0$$

Intuitive Bedeutung der Programme

Alle drei Programme manipulieren die Werte zweier Variablen (oder Speicherzellen, Register), die wir mit x und y bezeichnen. Ein Semikolon steht für die Hintereinanderausführung der Anweisungen, ansonsten sind die Programme selbsterklärend.

P_1 und P_2 verhalten sich also gleich — nach ihrer Ausführung hat x den (Anfangs)wert von y plus 2, während y unverändert bleibt.

Nach Ausführung von Q haben x und y den Wert 0.

Formale Semantik

Beispiel 10.1 II

Formal können wir die Bedeutung der Programme als eine Abbildung zwischen Variablenbelegungen (Speicherinhalten) auffassen:

$P_1: \quad x \leftarrow y + 2$ $\qquad [\![P_1]\!]:$ $\begin{array}{c|c} x & m \\ y & n \end{array} \longrightarrow \begin{array}{c|c} x & m+2 \\ y & n \end{array}$

$P_2: \quad x \leftarrow y; x \leftarrow x + 1; x \leftarrow x + 1$ $\qquad [\![P_2]\!]:$ $\begin{array}{c|c} x & m \\ y & n \end{array} \longrightarrow \begin{array}{c|c} x & m+2 \\ y & n \end{array}$

$Q: \quad x \leftarrow 0; y \leftarrow 0$ $\qquad [\![Q]\!]:$ $\begin{array}{c|c} x & m \\ y & n \end{array} \longrightarrow \begin{array}{c|c} x & 0 \\ y & 0 \end{array}$

Hintereinanderausführung

Beispiel 10.1 III

$[\![P_1; Q]\!]$: $\begin{matrix} x & \boxed{m} \\ y & \boxed{n} \end{matrix} \longrightarrow \begin{matrix} x & \boxed{0} \\ y & \boxed{0} \end{matrix}$ $\qquad$ $[\![Q; P_1]\!]$: $\begin{matrix} x & \boxed{m} \\ y & \boxed{n} \end{matrix} \longrightarrow \begin{matrix} x & \boxed{2} \\ y & \boxed{0} \end{matrix}$

$[\![P_2; Q]\!]$: $\begin{matrix} x & \boxed{m} \\ y & \boxed{n} \end{matrix} \longrightarrow \begin{matrix} x & \boxed{0} \\ y & \boxed{0} \end{matrix}$ $\qquad$ $[\![Q; P_2]\!]$: $\begin{matrix} x & \boxed{m} \\ y & \boxed{n} \end{matrix} \longrightarrow \begin{matrix} x & \boxed{2} \\ y & \boxed{0} \end{matrix}$

Nebenläufige Ausführung

Bei nebenläufiger Ausführung der Prozesse P_i und Q ist das Verhalten nicht mehr eindeutig bestimmt. Je nach Zeitpunkt der einzelnen Aktionen der einzelnen Prozesse ergeben sich unterschiedlich Resultate.

Zudem verhalten sich $P_1 \mid Q$ und $P_2 \mid Q$ nicht gleich, obwohl P_1 und P_2 sich gleich verhalten.

Beispiel 10.1 IV

$[\![P_1 \mid Q]\!]$: $\begin{matrix} x & \boxed{m} \\ y & \boxed{n} \end{matrix} \longrightarrow \left\{ \begin{matrix} x & \boxed{0} \\ y & \boxed{0} \end{matrix}, \begin{matrix} x & \boxed{2} \\ y & \boxed{0} \end{matrix}, \begin{matrix} x & \boxed{n+2} \\ y & \boxed{0} \end{matrix} \right\}$

$[\![P_2 \mid Q]\!]$: $\begin{matrix} x & \boxed{m} \\ y & \boxed{n} \end{matrix} \longrightarrow \left\{ \begin{matrix} x & \boxed{0} \\ y & \boxed{0} \end{matrix}, \begin{matrix} x & \boxed{1} \\ y & \boxed{0} \end{matrix}, \begin{matrix} x & \boxed{2} \\ y & \boxed{0} \end{matrix}, \begin{matrix} x & \boxed{n+2} \\ y & \boxed{0} \end{matrix} \right\}$

Schlussfolgerungen

- Das Verhalten nebenläufiger Prozesse komplex, es hängt davon ab, zu welchen Zeitpunkten die Prozesse auf welche Weise miteinander "kommunizieren" (zum Beispiel durch den Lese-/Schreibzugriff auf gemeinsame Variablen).
- Können wir diese Zeitpunkte nicht vorhersehen, so können wir das Verhalten nur als nichtdeterministisch beschreiben, d.h., eine Eingabe kann zu verschiedenen Ausgaben führen.
- Das Verhalten von nebenläufigen Prozessen lässt sich nicht mehr ohne weiteres aus dem Verhalten seiner Komponenten ableiten:
  P_1 und P_2 verhalten sich gleich, $P_1 \mid Q$ und $P_2 \mid Q$ aber nicht.

Schlussfolgerungen (Forts.)

- Es ist sehr schwierig das Verhalten nebenläufiger Systeme zu verstehen. Deswegen ist es auch schwierig, korrekte nebenläufige Systeme zu entwickeln.
- Aus diesem Grund ist es wichtig, formale Beschreibungen nebenläufiger Systeme zu haben, um deren Korrektheit formal (und möglichst automatisch) nachweisen zu können.
- Es gibt eine Vielzahl formaler Modelle von nebenläufigen Systemen, von denen wir in den folgenden Abschnitten einige kennenlernen werden.

Abschnitt 10.2

Synchronisierte Produkte

Transitionssysteme

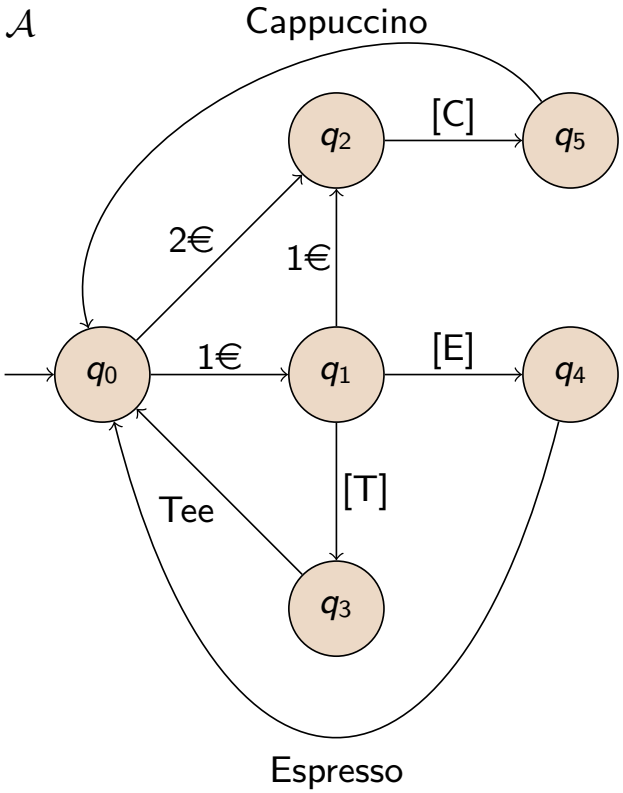
Das vielleicht einfachste Prozessmodell sind ε-NFAs.

- Die Buchstaben des Alphabets interpretieren wir als (beobachtbare) Aktionen.
- ε-Transitionen interpretieren wir als nicht beobachtbare Aktionen.
- Die Endzustände sind häufig irrelevant, weil wir uns vor allem für die Läufe des Systems interessieren.

  Man spricht dann auch von Transitionssystemen.

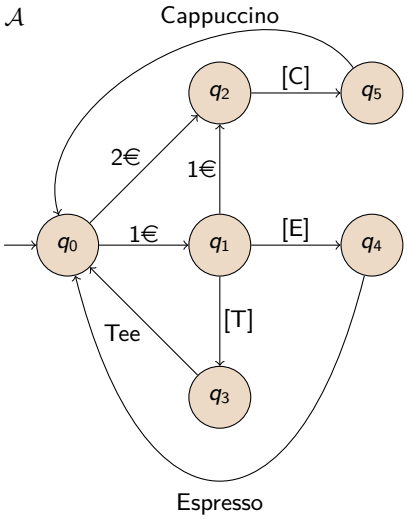
Beispiel 10.2

Als Transitionssystem betrachten wir wieder unser Modell eines Getränkeautomaten:

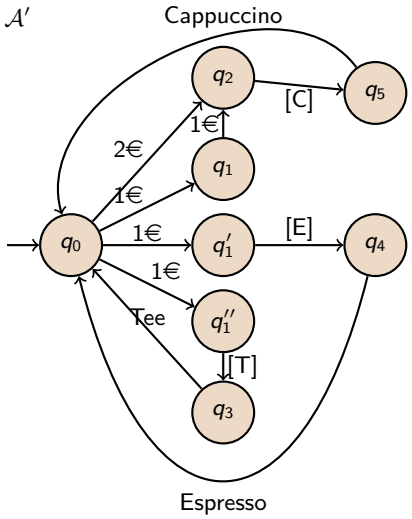


Beispiel 10.2 (Forts.)

Transitionssystem



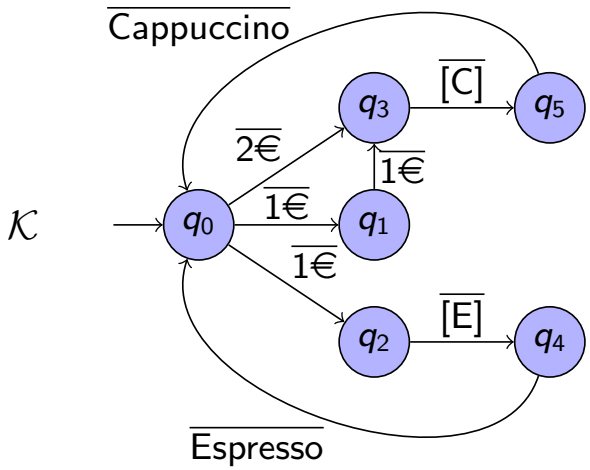
Alternatives Transitionssystem



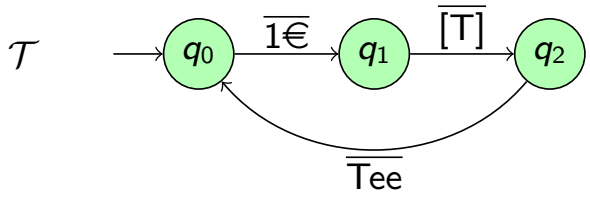
- Bisher haben wir die Semantik von Transitionssystemen (bzw. endlichen Automaten) über die erkannte Sprache definiert.
- Fasst man alle Zustände als akzeptierende Endzustände auf, so erkennen $\mathcal{A}$ und $\mathcal{A}'$ dieselbe Sprache
- Dennoch verhalten sich die beiden Systeme unterschiedlich!
- Die Semantik sollte in diesem Kontext also nicht mehr allein über die Sprache definiert werden.

Zwei weitere Prozesse:

Kaffeetrinker



Teetrinker



Nebenläufige Ausführung

- Wir möchten nun die drei Prozesse Getränkeautomat, Kaffeetrinker, und Teetrinker nebenläufig ausführen.
- Dabei sollen gewisse Aktionen synchronisiert werden, zum Beispiel $\overline{[C]}$ ("Der Kaffeetrinker drückt die Taste [C]") und [C] ("Die Taste [C] wird gedrückt").
- Wir modellieren dies durch eine Produktkonstruktion: Wir bilden den Produktautomaten der drei Systeme, erlauben aber bei manchen Transitionen nur eine synchronisierte Ausführung auf verschiedenen Komponenten.

Definition 10.3

Für $i = 1, \ldots, n$ sei $\mathcal{A}_i = (Q_i, \Sigma_i, q_{0i}, \Delta_i)$ ein Transitionssystem. Das freie Produkt von $\mathcal{A}_1, \ldots, \mathcal{A}_n$ ist das Transitionssystem

$$\prod_{i=1}^{n} \mathcal{A}_i := \mathcal{A}_1 \times \cdots \times \mathcal{A}_n := (Q, \Sigma, q_0, \Delta),$$

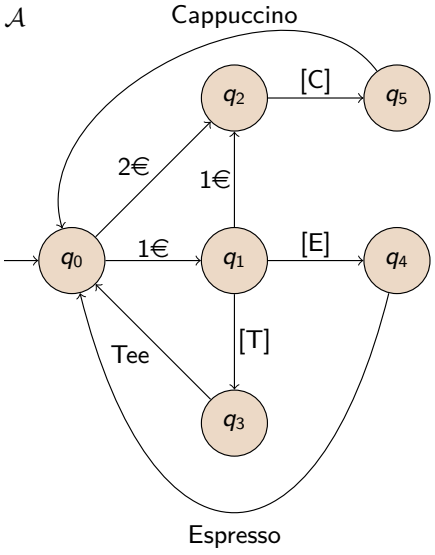
wobei

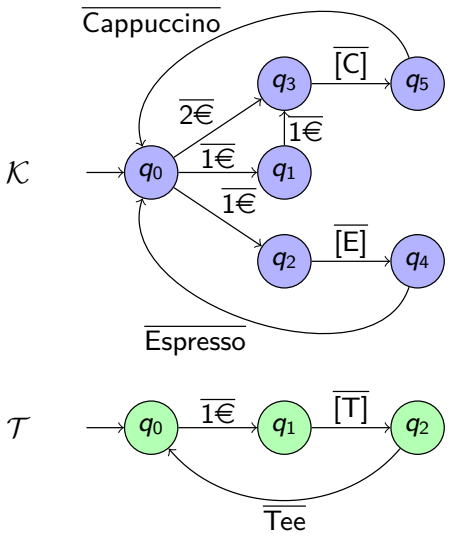
- $Q = \prod_{i=1}^{n} Q_i$,
- $\Sigma = \prod_{i=1}^{n} (\Sigma_i \cup \{\varepsilon\})$,
- $q_0 = (q_{01}, \ldots, q_{0n})$,
- Δ besteht aus allen Transitionen

$$((q_1, \ldots, q_n), (\alpha_1, \ldots, \alpha_n), (q'_1, \ldots, q'_n)) \in Q \times \Sigma \times Q$$

so dass für alle $i \in [n]$ gilt: $\alpha_i = \varepsilon$ und $q_i = q'_i$ oder $\alpha_i = a_i \in \Sigma_i$ und $(q_i, a_i, q'_i) \in \Delta_i$.

Beispiel 10.4





Freies Produkt $\mathcal{A} \times \mathcal{K} \times \mathcal{T}$.

- 108 Zustände und hunderte Transitionen.
- Eine graphische Darstellung des Systems ist deswegen nicht sonderlich aufschlussreich.

Beispiellauf 1 (sinnvoll)

$$(q_0) \xrightarrow{\binom{2€}{2€}} (q_2) \xrightarrow{\binom{[C]}{[C]}} (q_5) \xrightarrow{\binom{\text{Cappuccino}}{\text{Cappuccino}}} (q_0) \xrightarrow{\binom{1€}{\varepsilon}} (q_1)$$

Idee

Erlaube im Produkt nur gewisse Transitionen, um so die Synchronisierung zu steuern und sinnlose Läufe auszuschließen.

Definition 10.5

Seien wieder $\mathcal{A}_i = (Q_i, \Sigma_i, q_{0i}, \Delta_i)$ für $i = 1, \ldots n$ Transitionssysteme, und sei $\prod_{i=1}^{n} \mathcal{A}_i = (Q, \Sigma, q_0, \Delta)$.

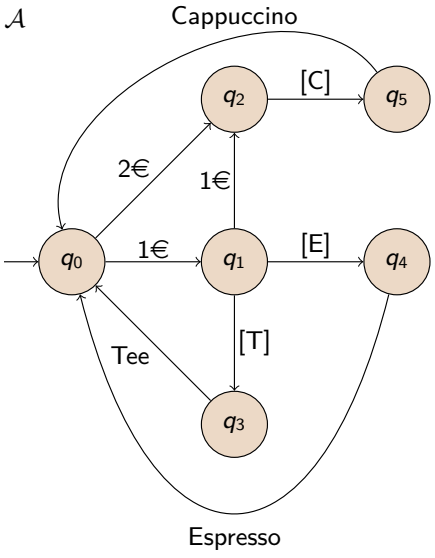
Ein Synchronisierungsschema ist eine Teilmenge $S \subseteq \Sigma$.

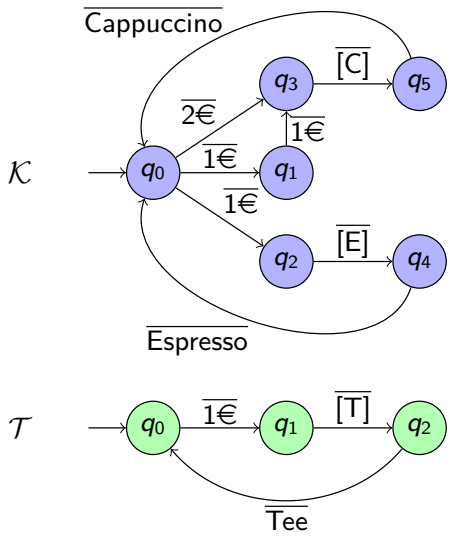
Das synchronisierte Produkt von $\mathcal{A}_1, \ldots, \mathcal{A}_n$ mit Synchronisierungsschema S ist das Transitionssystem

$$\Big(\prod_{i=1}^{n} \mathcal{A}_i \;\Big|\; S\Big) \qquad \text{(oder auch } (\mathcal{A}_1 \times \cdots \times \mathcal{A}_n \mid S)\text{)},$$

dass aus dem freien Produkt $\prod_{i=1}^{n} \mathcal{A}_i$ entsteht, wenn man nur Transitionen (q, a, q') mit $a \in S$ zulässt.

Beispiel 10.6





Synchronisiertes Produkt $\mathcal{B} := (\mathcal{A} \times \mathcal{K} \times \mathcal{T} \mid S)$ mit

$$S = \left\{ \begin{pmatrix} 2€ \\ \overline{2€} \\ \varepsilon \end{pmatrix}, \begin{pmatrix} 1€ \\ \overline{1€} \\ \varepsilon \end{pmatrix}, \begin{pmatrix} 1€ \\ \varepsilon \\ \overline{1€} \end{pmatrix}, \begin{pmatrix} [C] \\ \overline{[C]} \\ \varepsilon \end{pmatrix}, \begin{pmatrix} [E] \\ \overline{[E]} \\ \varepsilon \end{pmatrix}, \begin{pmatrix} [T] \\ \varepsilon \\ \overline{[T]} \end{pmatrix}, \right.$$
$$\left. \begin{pmatrix} \text{Cappuccino} \\ \overline{\text{Cappuccino}} \\ \varepsilon \end{pmatrix}, \begin{pmatrix} \text{Espresso} \\ \overline{\text{Espresso}} \\ \varepsilon \end{pmatrix}, \begin{pmatrix} \text{Tee} \\ \varepsilon \\ \overline{\text{Tee}} \end{pmatrix}, \right\}$$

Beispiel 10.6 (Forts)

Beispiellauf 1

$$\begin{pmatrix} q_0 \\ q_0 \\ q_0 \end{pmatrix} \xrightarrow{\begin{pmatrix} 2€ \\ \overline{2€} \\ \varepsilon \end{pmatrix}} \begin{pmatrix} q_2 \\ q_3 \\ q_0 \end{pmatrix} \xrightarrow{\begin{pmatrix} [\mathrm{C}] \\ \overline{[\mathrm{C}]} \\ \varepsilon \end{pmatrix}} \begin{pmatrix} q_5 \\ q_5 \\ q_0 \end{pmatrix} \xrightarrow{\begin{pmatrix} \mathrm{Cappuccino} \\ \overline{\mathrm{Cappuccino}} \\ \varepsilon \end{pmatrix}} \begin{pmatrix} q_0 \\ q_0 \\ q_0 \end{pmatrix} \xrightarrow{\begin{pmatrix} 1€ \\ \varepsilon \\ \overline{1€} \end{pmatrix}} \begin{pmatrix} q_1 \\ q_0 \\ q_1 \end{pmatrix}$$

Beispiellauf 2

$$\begin{pmatrix} q_0 \\ q_0 \\ q_0 \end{pmatrix} \xrightarrow{\begin{pmatrix} 1€ \\ \varepsilon \\ \overline{1€} \end{pmatrix}} \begin{pmatrix} q_1 \\ q_1 \\ q_0 \end{pmatrix} \xrightarrow{\begin{pmatrix} 1€ \\ \overline{1€} \\ \varepsilon \end{pmatrix}} \begin{pmatrix} q_2 \\ q_1 \\ q_0 \end{pmatrix}$$

Im Gegensatz zu den sinnlosen Läufen, die es im freien Produkt gab, ist dieser Lauf durchaus möglich und weist auf einen Fehler im System hin.

Beispiel 10.7

Betrachte den Prozess, der dem Ablauf des folgenden Programmsegments entspricht:

```
1: while true do
2:     if not b then
3:         b ← true
4:         sleep {oder mach irgendetwas anderes}
5:         b ← false
6:     end if
7: end while
```

Wir modellieren den Prozess mit 4 Zuständen, die den Zeilen 2,3,4,5 entsprechen und fünf Aktionen:

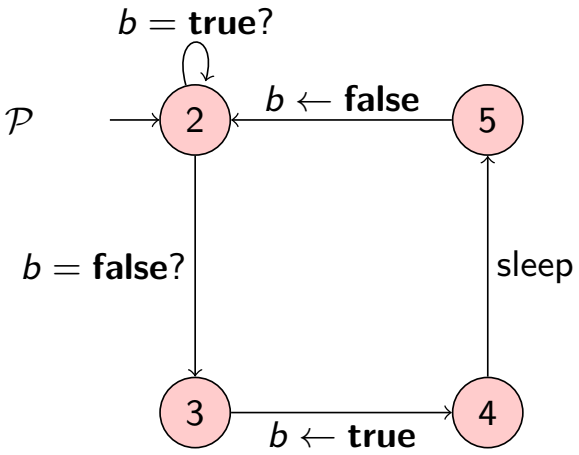
- $b = \textbf{true}$? ("b wird abgefragt, und der Wert ist **true**.")
- $b = \textbf{false}$?
- $b \leftarrow \textbf{true}$ ("b wird auf **true** gesetzt.")
- $b \leftarrow \textbf{false}$
- sleep

Beispiel 10.7 (Forts.)

Programmfragment

```
1: while true do
2:     if not b then
3:         b ← true
4:         sleep
5:         b ← false
6:     end if
7: end while
```

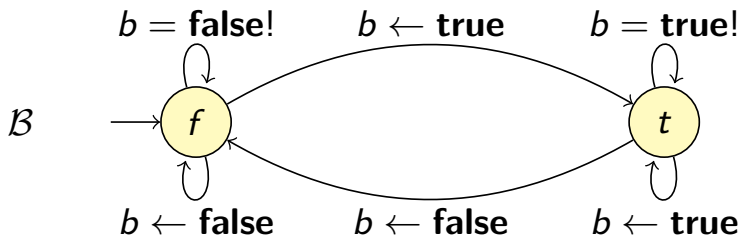
Transitionssystem



Beispiel 10.7 (Forts.)

Oft ist es sinnvoll, die Variable b als eigenen Prozess zu modellieren, zum Beispiel wenn noch andere Prozesse darauf zugreifen.

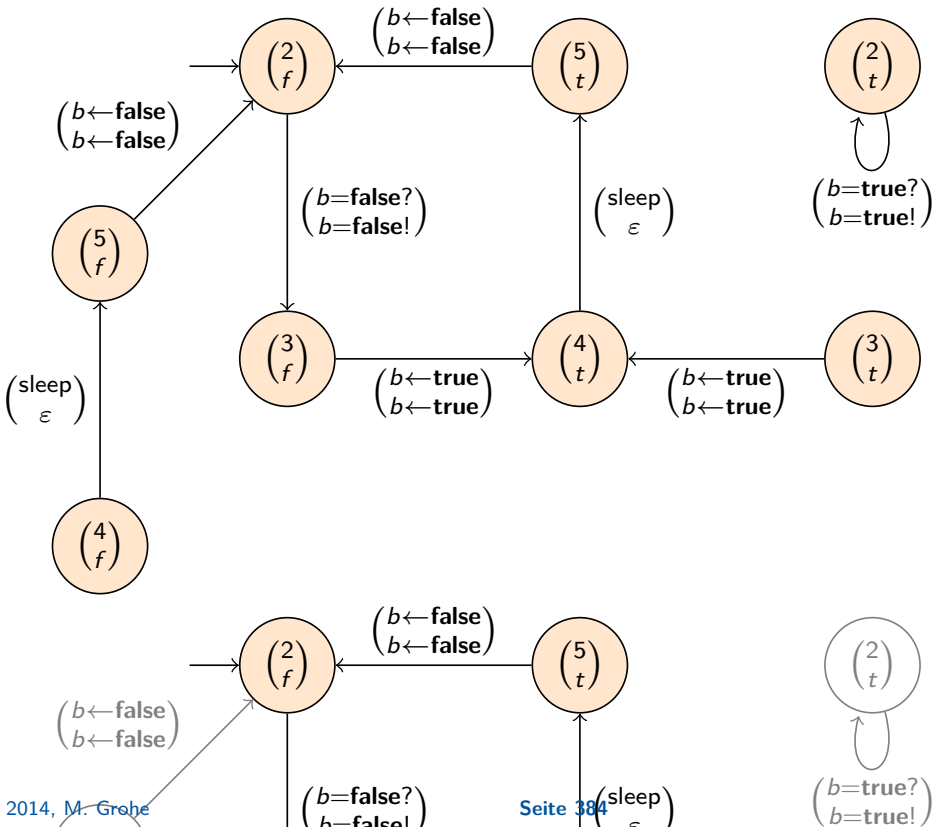
Transitionssystem für Boolesche Variable b



Beispiel 10.7 (Forts.)

Synchronisiertes Produkt $(\mathcal{P} \times \mathcal{B} \mid S)$ mit

$$S = \left\{ \binom{b{=}\textbf{true}?}{b{=}\textbf{true}!}, \binom{b{=}\textbf{false}?}{b{=}\textbf{false}!}, \binom{b{\leftarrow}\textbf{true}}{b{\leftarrow}\textbf{true}}, \binom{b{\leftarrow}\textbf{false}}{b{\leftarrow}\textbf{false}}, \binom{\text{sleep}}{\varepsilon} \right\}$$



Verwendung der ε-Transitionen

Betrachten wir ein synchronisiertes Produkt $(\mathcal{A} \times \mathcal{B} \mid S)$ und a aus dem Alphabet von $\mathcal{A}$ sowie b aus dem Alphabet von $\mathcal{B}$

Enthält S die Transitionen $\binom{a}{\varepsilon}$, $\binom{\varepsilon}{b}$, so können wir die Transition $\binom{a}{b}$ hinzufügen oder weglassen, ohne das System wesentlich zu verändern. Auf diese Weise können wir die Synchronisierungsschemata verkleinern.

Diese Beobachtung lässt sich auf synchronisierte Produkte beliebig vieler Prozesse verallgemeinern

Beispiel 10.8: Mutual Exclusion Protokoll

Nebenläufige Prozesse $\mathcal{P}_1$, $\mathcal{P}_2$ mit Zugriff auf gemeinsame Variablen

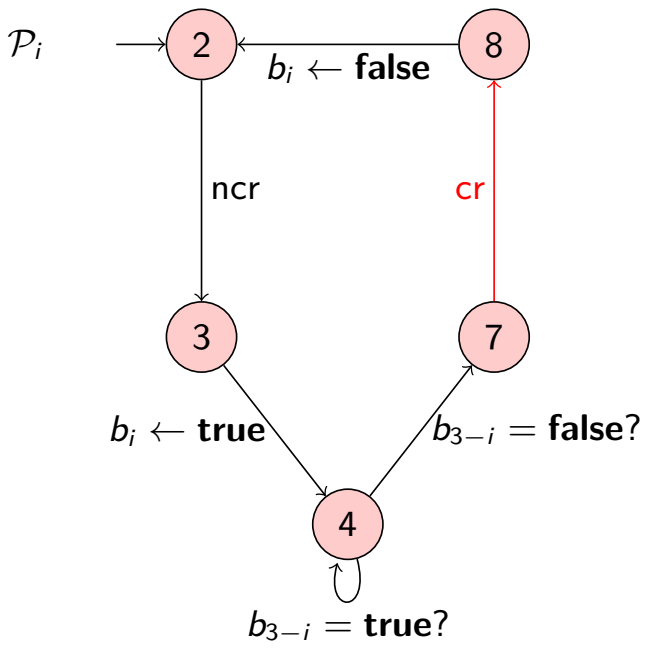
```
1: while true do
2:     {nicht kritisch}
3:     b1 ← true
4:     while b2 do
5:         {Warte}
6:     end while
7:     {kritischer Bereich}
8:     b1 ← false
9: end while
```

```
1: while true do
2:     {nicht kritisch}
3:     b2 ← true
4:     while b1 do
5:         {Warte}
6:     end while
7:     {kritischer Bereich}
8:     b2 ← false
9: end while
```

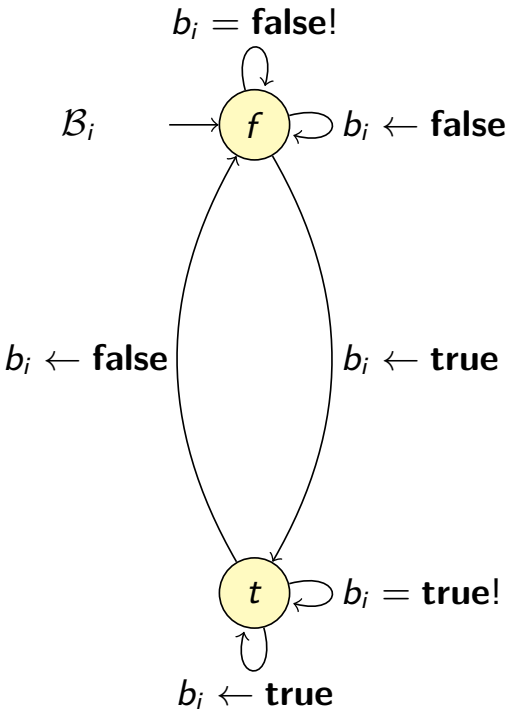
Nur ein Prozess darf sich im kritischen Bereich befinden.

Beispiel 10.8 (Forts.): Modellierung

Prozess $\mathcal{P}_i$



Variable $\mathcal{B}_i$



Beispiel 10.8 (forts.): Produkt

Synchronisiertes Produkt $(\mathcal{P}_1 \times \mathcal{P}_2 \times \mathcal{B}_1 \times \mathcal{B}_2 \mid S)$, wobei S folgende Aktionen erlaubt:

- $\begin{pmatrix} b_1 = \textbf{true}? \\ \varepsilon \\ b_1 = \textbf{true}! \\ \varepsilon \end{pmatrix}, \begin{pmatrix} b_1 = \textbf{false}? \\ \varepsilon \\ b_1 = \textbf{false}! \\ \varepsilon \end{pmatrix}, \begin{pmatrix} b_2 = \textbf{true}? \\ \varepsilon \\ \varepsilon \\ b_2 = \textbf{true}! \end{pmatrix}, \begin{pmatrix} b_2 = \textbf{false}? \\ \varepsilon \\ \varepsilon \\ b_2 = \textbf{false}! \end{pmatrix},$
  $\begin{pmatrix} \varepsilon \\ b_1 = \textbf{true}? \\ b_1 = \textbf{true}! \\ \varepsilon \end{pmatrix}, \begin{pmatrix} \varepsilon \\ b_1 = \textbf{false}? \\ b_1 = \textbf{false}! \\ \varepsilon \end{pmatrix}, \begin{pmatrix} \varepsilon \\ b_2 = \textbf{true}? \\ \varepsilon \\ b_2 = \textbf{true}! \end{pmatrix}, \begin{pmatrix} \varepsilon \\ b_2 = \textbf{false}? \\ \varepsilon \\ b_2 = \textbf{false}! \end{pmatrix},$
- $\begin{pmatrix} b_1 \leftarrow \textbf{true} \\ \varepsilon \\ b_1 \leftarrow \textbf{true} \\ \varepsilon \end{pmatrix}, \begin{pmatrix} b_1 \leftarrow \textbf{false} \\ \varepsilon \\ b_1 \leftarrow \textbf{false} \\ \varepsilon \end{pmatrix}, \begin{pmatrix} \varepsilon \\ b_2 \leftarrow \textbf{true} \\ \varepsilon \\ b_2 \leftarrow \textbf{true} \end{pmatrix}, \begin{pmatrix} \varepsilon \\ b_2 \leftarrow \textbf{false} \\ \varepsilon \\ b_2 \leftarrow \textbf{false} \end{pmatrix},$
- $\begin{pmatrix} \text{ncr} \\ \varepsilon \\ \varepsilon \\ \varepsilon \end{pmatrix}, \begin{pmatrix} \text{cr} \\ \varepsilon \\ \varepsilon \\ \varepsilon \end{pmatrix}, \begin{pmatrix} \varepsilon \\ \text{ncr} \\ \varepsilon \\ \varepsilon \end{pmatrix}, \begin{pmatrix} \varepsilon \\ \text{cr} \\ \varepsilon \\ \varepsilon \end{pmatrix}.$

Beispiel 10.8 (Forts.) I

Ein Lauf des Systems $(\mathcal{P}_1 \times \mathcal{P}_2 \times \mathcal{B}_1 \times \mathcal{B}_2 \mid S)$:

$$\begin{pmatrix} 2 \\ 2 \\ f \\ f \end{pmatrix} \xrightarrow{\begin{pmatrix} \text{ncr} \\ \varepsilon \\ \varepsilon \\ \varepsilon \end{pmatrix}} \begin{pmatrix} 3 \\ 2 \\ f \\ f \end{pmatrix} \xrightarrow{\begin{pmatrix} \varepsilon \\ \text{ncr} \\ \varepsilon \\ \varepsilon \end{pmatrix}} \begin{pmatrix} 3 \\ 3 \\ f \\ f \end{pmatrix} \xrightarrow{\begin{pmatrix} \varepsilon \\ b_2 \leftarrow \textbf{true} \\ \varepsilon \\ b_2 \leftarrow \textbf{true} \end{pmatrix}} \begin{pmatrix} 3 \\ 4 \\ f \\ t \end{pmatrix}$$

$$\xrightarrow{\begin{pmatrix} \varepsilon \\ b_1 = \textbf{false}? \\ b_1 = \textbf{false}! \\ \varepsilon \end{pmatrix}} \begin{pmatrix} 3 \\ 7 \\ f \\ t \end{pmatrix} \xrightarrow{\begin{pmatrix} \varepsilon \\ \text{cr} \\ \varepsilon \\ \varepsilon \end{pmatrix}} \begin{pmatrix} 3 \\ 8 \\ f \\ t \end{pmatrix} \xrightarrow{\begin{pmatrix} \varepsilon \\ b_2 \leftarrow \textbf{false} \\ \varepsilon \\ b_2 \leftarrow \textbf{false} \end{pmatrix}} \begin{pmatrix} 3 \\ 2 \\ f \\ f \end{pmatrix}$$

$$\xrightarrow{\begin{pmatrix} b_1 \leftarrow \textbf{true} \\ \varepsilon \\ b_1 \leftarrow \textbf{true} \\ \varepsilon \end{pmatrix}} \begin{pmatrix} 4 \\ 2 \\ t \\ f \end{pmatrix} \xrightarrow{\begin{pmatrix} b_2 = \textbf{false}? \\ \varepsilon \\ \varepsilon \\ b_2 = \textbf{false}! \end{pmatrix}} \begin{pmatrix} 7 \\ 2 \\ 2 \\ f \end{pmatrix} \xrightarrow{\begin{pmatrix} \text{cr} \\ \varepsilon \\ \varepsilon \\ \varepsilon \end{pmatrix}} \begin{pmatrix} 3 \\ 8 \\ f \\ t \end{pmatrix}$$

Beispiel 10.8 (Forts.) II

Ein weiterer Lauf des Systems:

$$\begin{pmatrix}2\\2\\f\\f\end{pmatrix} \xrightarrow{\begin{pmatrix}\text{ncr}\\ \varepsilon\\ \varepsilon\\ \varepsilon\end{pmatrix}} \begin{pmatrix}3\\2\\f\\f\end{pmatrix} \xrightarrow{\begin{pmatrix}\varepsilon\\ \text{ncr}\\ \varepsilon\\ \varepsilon\end{pmatrix}} \begin{pmatrix}3\\3\\f\\f\end{pmatrix} \xrightarrow{\begin{pmatrix}\varepsilon\\ b_2 \leftarrow \textbf{true}\\ \varepsilon\\ b_2 \leftarrow \textbf{true}\end{pmatrix}} \begin{pmatrix}3\\4\\f\\t\end{pmatrix} \xrightarrow{\begin{pmatrix}b_1 \leftarrow \textbf{true}\\ \varepsilon\\ b_1 \leftarrow \textbf{true}\\ \varepsilon\end{pmatrix}} \begin{pmatrix}4\\4\\t\\t\end{pmatrix}$$

An dieser Stelle sind nur noch die Transitionen

Beispiel 10.8 (Forts.) III

$$\begin{pmatrix}4\\4\\t\\t\end{pmatrix} \xrightarrow{\begin{pmatrix}b_2 = \textbf{true}?\\ \varepsilon\\ \varepsilon\\ b_2 = \textbf{true}!\end{pmatrix}} \begin{pmatrix}4\\4\\t\\t\end{pmatrix} \quad \text{und} \quad \begin{pmatrix}4\\4\\t\\t\end{pmatrix} \xrightarrow{\begin{pmatrix}\varepsilon\\ b_1 = \textbf{true}?\\ b_1 = \textbf{true}!\\ \varepsilon\end{pmatrix}} \begin{pmatrix}4\\4\\t\\t\end{pmatrix}$$

möglich, die den Zustand nicht verändern.

Das System befindet sich in einem Deadlock!

Diskussion

- Unsere Analyse hat gezeigt, dass das Mutual Exclusion Protokoll fehlerhaft ist — ein Deadlock darf nicht entstehen.
- Sobald ein formales Modell des Systems erstellt ist, kann man die Analyse und Fehlersuche, etwa die Suche nach Deadlocks, automatisieren.
  Noch besser ist es, das Systemmodell automatisch aus dem Code zu generieren, so dass der ganze Analyseprozess automatisiert wird.
- Ein entscheidendes praktisches Problem ist allerdings, dass synchronisierte Produkte und andere Modelle oft zu groß werden: Das synchronisierte Produkt von n Systemen mit je k Zuständen hat k^n Zustände ("State Explosion Problem").

Mehr dazu in der Vorlesung Model Checking.

Petersons Mutual Exclusion Protokoll

```
 1: while true do
 2:     {nicht kritisch}
 3:     b1 ← true
 4:     p ← 1
 5:     while b2 and p = 1 do
 6:         {Warte}
 7:     end while
 8:     {kritischer Bereich}
 9:     b1 ← false
10: end while
```

```
 1: while true do
 2:     {nicht kritisch}
 3:     b2 ← true
 4:     p ← 2
 5:     while b1 and p = 2 do
 6:         {Warte}
 7:     end while
 8:     {kritischer Bereich}
 9:     b2 ← false
10: end while
```

Lässt sich auf ähnliche Weise als synchronisiertes Produkt modellieren.
Man kann dann nachweisen, dass das System deadlockfrei ist.

Abschnitt 10.3

Prozesskalküle

Prozesskalküle

- Wie synchronisierte Produkte dienen Prozesskalküle der Modellierung nebenläufiger Prozesse.
- In ihrer einfachsten Form lassen sie sich als eine algebraische Darstellung von endlichen Automaten (durch Terme und Gleichungen) auffassen.
- Die Kalküle bieten aber in der Regel zusätzliche Möglichkeiten, etwa das Lokalisieren (Verstecken) von Variablen, um den Komponenten klar definierte Schnittstellen zu geben.
  Auch funktioniert die Synchronisierung etwas anders.
- Es gibt zahlreiche Prozesskalküle, in der Theorie sind die wichtigsten CCS ("Calculus of Communicating Systems"), CSP ("Communicating Sequential Processes") und der π-Kalkül.

Algebraische Beschreibung von Transitionssystemen

Sei $\mathcal{A} = (Q, \Sigma, q_0, \Delta)$ ein Transitionssystem.

Wir beschreiben die Transitionsrelation als System von Gleichungen. Für $q \in Q$ seien $(q, a_1, q_1), \ldots, (q, a_n, q_n)$ alle Transitionen, die von q ausgehen. Wir fügen folgende Gleichung hinzu:

$$q \equiv a_1.q_1 + \ldots + a_n.q_n$$

Alternativ schreiben wir $q \equiv \sum_{i=1}^{n} a_i.q_i$.

Man beachte, dass die Gleichungen auch rekursiv sein können.

Beispiel 10.9

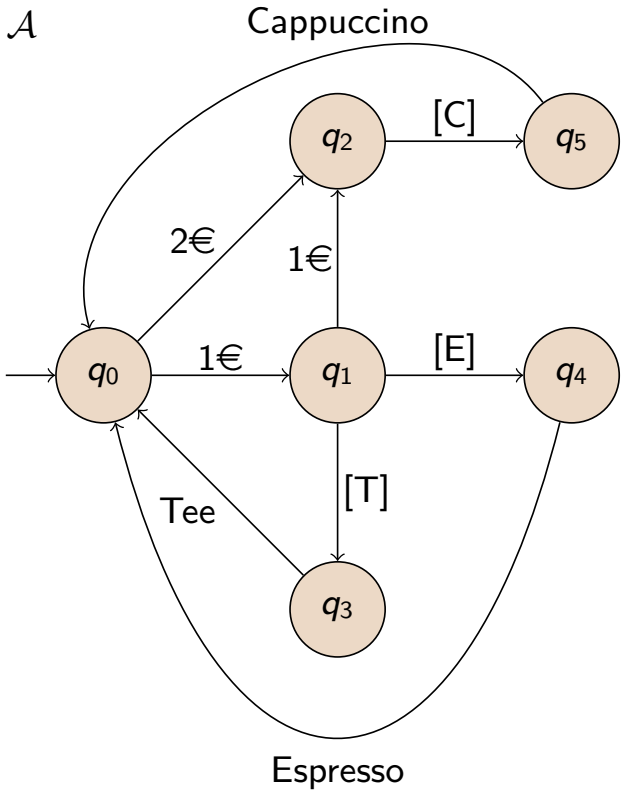
Die Gleichung

$$q \equiv a.q + a'.q',$$

besagt, dass es vom Zustand q aus eine a-Transition nach q und eine a'-Transition nach q' gibt.

Beispiel

Transitionssystem



Algebraische Beschreibung

$$q_0 \equiv 2€.q_2 + 1€.q_1$$
$$q_1 \equiv 1€.q_2 + [\mathrm{E}].q_4 + [\mathrm{T}].q_3$$
$$q_2 \equiv [\mathrm{C}].q_5$$
$$q_3 \equiv \mathrm{Tee}.q_0$$
$$q_4 \equiv \mathrm{Espresso}.q_0$$
$$q_5 \equiv \mathrm{Cappuccino}.q_0$$

Algebra

Wir interpretieren die Terme und Gleichungen in einer algebraischen Struktur, deren Elemente die Zustände des Systems repräsentieren und in der wir folgende Operationen haben:

- für jede Aktion $a \in \Sigma$ eine einstellige Operation $a.$;
- eine zweistellige Operation $+$.
  Dabei ist $+$ assoziativ und kommutativ, deswegen müssen wir in Gleichungen $q \equiv \sum_{i=1}^{n} a_1.q_i$ nicht klammern, und die Reihenfolge der Summanden spielt keine Rolle.

Wir erlauben es, dass so ein System unendlich viele Zustände hat.

Vergleiche dies mit einem Vektorraum über einem Körper K. Dort haben wir ebenfalls eine assoziative und kommutative zweistellige Operation $+$ auf den Elementen (Vektoren) sowie für alle $a \in K$ eine einstellige Operation $a\cdot$ (Skalarmultiplikation).

Wir können also den Vektorraum als spezielles Transitionssystem mit Aktionsalphabet K auffassen. (Das bringt allerdings nichts.)

Notation

Die für Prozesskalküle übliche Notation unterscheidet sich von unser Notation für Automaten.

- Zustände, die auch "Agenten" oder "Prozesse" genannt werden, werden mit Grossbuchstaben oder groß geschriebenen Wörtern bezeichnet.
- Aktionen werden mit Kleinbuchstaben oder klein geschrieben Wörtern bezeichnet.
- Wenn sich dies anbietet, verwenden wir auch Zahlen und andere Zeichen, sowohl für Zustände als auch für Aktionen

Kaffeeautomat

Zustände: KA, $EinE$, $ZweiE$, Cap, Esp, Tee

Aktionen: $1€$, $2€$, $capkn$, $espkn$, $teekn$, cap, esp, tee

Gleichungen:

$$\begin{aligned} KA &\equiv 1€.EinE + 2€.ZweiE \\ EinE &\equiv 1€.ZweiE + espkn.Esp + teekn.Tee \\ Tee &\equiv tee.KA \\ Esp &\equiv esp.KA \\ ZweiE &\equiv capkn.Cap \\ Cap &\equiv cap.KA \end{aligned}$$

Rechnen in der Algebra

Aus den Gleichungen, die ein System beschreiben, können wir durch Einsetzen weitere Gleichungen ableiten.

Beispiel 10.10 (Forts.)

$$\begin{aligned} KA &\equiv 1€.EinE + 2€.ZweiE \\ EinE &\equiv 1€.ZweiE + espkn.Esp + teekn.Tee \\ ZweiE &\equiv capkn.Cap \end{aligned}$$

implizieren

$$\begin{aligned} EinE &\equiv 1€.\ capkn.Cap + espkn.Esp + teekn.Tee \\ KA &\equiv 1€.(1€.\ capkn.Cap + espkn.Esp + teekn.Tee) + 2€.capkn.Cap \end{aligned}$$

Achtung

Es gilt kein Distributivgesetz, d.h.,

$$a.(B + C) \not\equiv aB + aC \qquad \text{(im allgemeinen).}$$

Wenn wir Prozesse spezifizieren, müssen wir nicht allen Teilprozessen eigene Namen geben.

Beispiel 10.11

- Teetrinker

$$TT \equiv \overline{1€}.\overline{teekn}.\overline{tee}.TT$$

- Kaffeetrinker

$$KT \equiv \overline{2€}.CapT + \overline{1€}.\ \overline{1€}.CapT + \overline{1€}.\overline{espkn}.\overline{esp}.KT$$
$$CapT \equiv \overline{capkn}.\overline{cap}.KT$$

Semantik

- Wir beschreiben Prozesse als Terme in einer Algebra. Um diese Terme mit Prozessen zu assoziieren, müssen wir eine Semantik definieren.
- Wir legen für jeden Term fest, welche Transitionen möglich sind.
- Damit können wir dann, rekursiv, jedem Term ein Transitionssystem zuordnen, dass den zughörigen Prozess darstellt.
- Wir schreiben $A \xrightarrow{a} A'$, wenn der Prozess A die Aktion a durchführen und in den Prozess A' übergehen kann.

Aktionen

- $a.A \xrightarrow{a} A$ für alle Aktionen a und Prozesse A.

Nichtdeterministische Auswahl (Summe)

- Wenn $A \xrightarrow{a} A'$, dann $A + B \xrightarrow{a} A'$, für alle Aktionen a und Prozesse A, A', B.

Äquivalenz

- Wenn $A \equiv B$ und $A \xrightarrow{a} A'$, dann $B \xrightarrow{a} A'$.

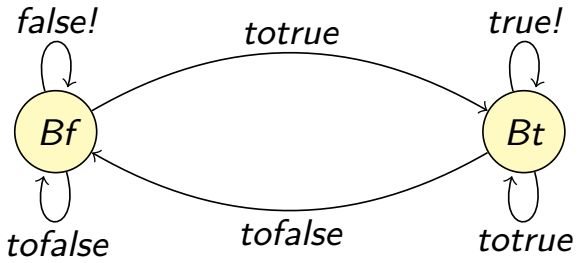
Neben den explizit angegebenen Gleichungen berücksichtigen wir dabei auch implizite Gleichungen wie $A + B \equiv B + A$ und $(A + B) + C \equiv A + (B + C)$.

Beispiel 10.12

Algebraische Spezifikation

$$Bf \equiv false!.Bf + tofalse.Bf + totrue.Bt$$
$$Bt \equiv true!.Bt + totrue.Bt + tofalse.Bf$$

Transitionssystem



Nebenläufigkeit

Bisher können wir mit unserer Prozessalgebra sequentielle Prozesse bzw. Transitionssysteme modellieren.

Um Nebenläufigkeit modellieren zu können, fügen wir eine neue zweistellige Operation $\mid$ hinzu:

$A \mid B$ bedeutet, dass A und B nebenläufig ausgeführt werden.

Wie $+$ ist $\mid$ assoziativ und kommutativ.

Synchronisation

Der Synchronisationsmechanismus in unserer Prozessalgebra unterscheidet sich deutlich von dem der synchronisierten Produkte.

- Für jede Aktion a führen wir eine komplementäre Aktion $\overline{a}$ ein. Dabei ist $\overline{\overline{a}}$ wieder a.
- Werden nun Prozesse $A \equiv a.A'$ und $B \equiv \overline{a}.B'$ nebenläufig ausgeführt, so können sie die komplementären Aktionen $a, \overline{a}$ synchron ausführen.
- Das ergibt eine Transition von $A \mid B$ nach $A' \mid B'$.
- Wir fassen diese Transition als eine interne, von außen nicht beobachtbare Transition auf. Formal bezeichnen wir sie als τ-Transition.
  Wir schreiben $A \mid B \xrightarrow{\tau} A' \mid B'$.

Allgemeiner legen wir die Semantik wie folgt fest:

Nebenläufige Ausführung

- Wenn $A \xrightarrow{a} A'$ und $B \xrightarrow{\overline{a}} B'$, so $A \mid B \xrightarrow{\tau} A' \mid B'$.
- Wenn $A \xrightarrow{a} A'$, dann $A \mid B \xrightarrow{a} A' \mid B$.

Außerdem die Gleichungen $A \mid B \equiv B \mid A$ und $A \mid (B \mid C) \equiv (A \mid B) \mid C$.

Beispiel 10.13

Kaffeeautomat

$$\begin{aligned} KA &\equiv 1€.EinE + 2€.ZweiE \\ EinE &\equiv 1€.ZweiE + espkn.Esp + teekn.Tee \\ Tee &\equiv tee.KA \\ Esp &\equiv esp.KA \\ ZweiE &\equiv capkn.Cap \\ Cap &\equiv cap.KA \end{aligned}$$

Teetrinker

$$TT \equiv \overline{1€}.\overline{teekn}.\overline{tee}.TT$$

Nebenläufige Ausführung

$$KA \mid TT \xrightarrow{\tau} EinE \mid \overline{teekn}.\overline{tee}.TT \xrightarrow{\tau} Tee \mid \overline{tee}.TT \xrightarrow{\tau} KA \mid TT$$

- Die Möglichkeit einer "beobachtbaren" Transition $A \xrightarrow{a} A'$ fassen wir als ein "Angebot" des Prozesses A auf, sich mit ihm über die Aktion a zu synchronisieren.

  Alternativ können wir es uns als Angebot des Agenten A vorstellen, mit ihm über den "Kanal" a zu kommunizieren.

- Wird dieses Angebot vom Prozess B wahrgenommen, so besteht es danach für keinen anderen Prozess mehr.

- Die Synchronisation ist dann eine interne Aktion des nebenläufigen Systems $A \mid B$.

  Weil wir uns vor allem für die Wechselwirkung zwischen Prozessen interessieren, ist diese Transition, wenn man den Prozess $(A \mid B)$ von außen betrachtet, nicht relevant.

Beispiel 10.14 I

Kaffeeautomat

$$\begin{aligned} KA &\equiv 1€.EinE + 2€.ZweiE \\ EinE &\equiv 1€.ZweiE + espkn.Esp + teekn.Tee \\ Tee &\equiv tee.KA \\ Esp &\equiv esp.KA \\ ZweiE &\equiv capkn.Cap \\ Cap &\equiv cap.KA \end{aligned}$$

Teetrinker

$$TT \equiv \overline{1€}.\overline{teekn}.\overline{tee}.TT$$

Kaffeeautomat und Teetrinker

$$KAT \equiv KA \mid TT$$

Beispiel 10.14 II

Kaffeetrinker

$$KT \equiv \overline{2€}.CapT + \overline{1€}.\ \overline{1€}.CapT + \overline{1€}.\overline{espkn}.\overline{esp}.KT$$
$$CapT \equiv \overline{capkn}.\overline{cap}.KT$$

Transitionen von KAT

$$KAT \xrightarrow{\tau} EinE \mid \overline{teekn}.\overline{tee}.TT$$
$$KAT \xrightarrow{1€} EinE \mid TT$$
$$KAT \xrightarrow{2€} ZweiE \mid TT$$
$$KAT \xrightarrow{\overline{1€}} KA \mid \overline{teekn}.\overline{tee}.TT$$

Transitionen von KT

$$KT \xrightarrow{\overline{2€}} CapT$$
$$KT \xrightarrow{\overline{1€}} \overline{1€}.CapT$$
$$KT \xrightarrow{\overline{1€}} \overline{espkn}.\overline{esp}.KT$$

Beispiel 10.14 III

Transitionen von KAT | KT

$$KAT \mid KT \xrightarrow{\tau} EinE \mid \overline{teekn}.\overline{tee}.TT \mid KT$$
$$KAT \mid KT \xrightarrow{1€} EinE \mid TT \mid KT$$
$$KAT \mid KT \xrightarrow{2€} ZweiE \mid TT \mid KT$$
$$KAT \mid KT \xrightarrow{\overline{1€}} KA \mid \overline{teekn}.\overline{tee}.TT \mid KT$$

$$KAT \mid KT \xrightarrow{\overline{2€}} KAT \mid CapT$$
$$KAT \mid KT \xrightarrow{\overline{1€}} KAT \mid \overline{1€}.CapT$$
$$KAT \mid KT \xrightarrow{\overline{1€}} KAT \mid \overline{espkn}.\overline{esp}.KT$$

$$KAT \mid KT \xrightarrow{\tau} ZweiE \mid TT \mid CapT$$
$$KAT \mid KT \xrightarrow{\tau} EinE \mid TT \mid 1€.CapT$$
$$KAT \mid KT \xrightarrow{\tau} EinE \mid TT \mid \overline{espkn}.\overline{esp}.KT$$

Wir haben die wesentlichen Elemente des Calculus of Communicating Systems (CCS) kennengelernt.

- Grundbestandteile sind Terme über einer Algebra, in der wir die Aktionen, nichtdeterministische Wahl und Nebenläufige Ausführung darstellen können. Die Terme selbst fassen wir als Prozesse auf.
- Wir erlauben Gleichungen zwischen den Termen und können dadurch rekursives Verhalten modellieren.
- Wir führen einen Synchronisationsmechanismus ein, bei dem zwei komplementäre Aktionen synchronisiert werden und damit zu einer "internen" Transition des Gesamtsystems werden.
- Die Semantik ist über die Transitionen zwischen Prozessen gegeben.

Abschnitt 10.4

Petrinetze

Carl Adam Petri, *Kommunikation mit Automaten*
(Dissertation 1962)

Grundprinzipien

Petrinetze sind ein weiteres wichtiges Modell nebenläufiger Prozesse.

Idee

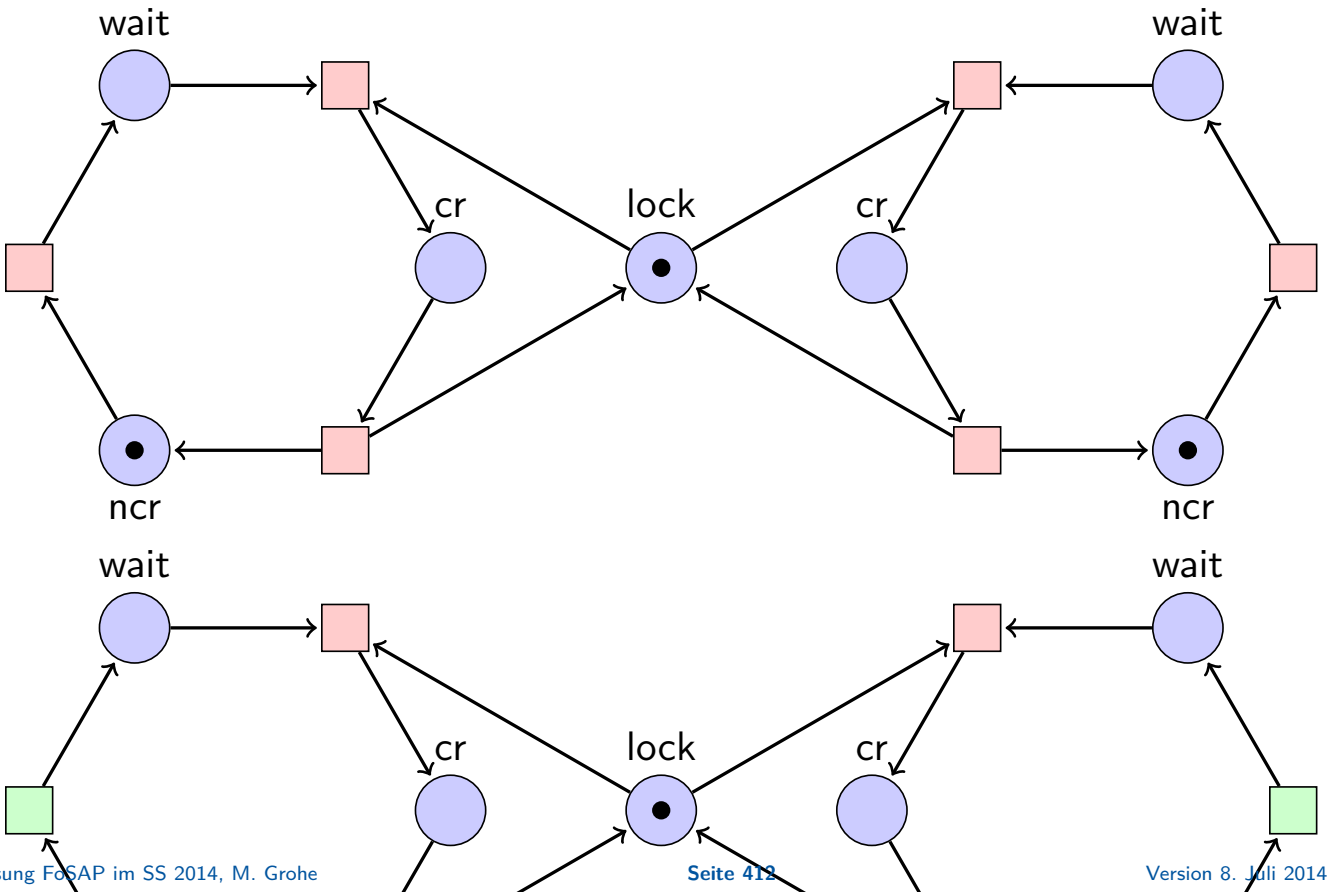
- System erlaubt Aktionen unter Verwendung von Ressourcen
- Eine Aktion verbraucht (und erzeugt) lokal Ressourcen.

Modellierung

- Ressourcen: "Marken"
- Ressourcenorte: "Plätze" mit Marken
- Aktionen: "Transitionen", die Marken erzeugen und verbrauchen
- Globaler Zustand: Verteilung von Marken auf Plätze
- Keine globale Synchronisation

Beispiel 10.15

Petrinetzmodellierung eines Mutual Exclusion Protokolls



Netze

Definition 10.16

Ein Petrinetz ist eine Tripel $\mathcal{N} = (P, T, F)$, bestehend aus

- einer nichtleeren endlichen Mengen P von Plätzen,
- einer zu P disjunkten nichtleeren endlichen Menge T von Transitionen,
- einer Flussrelation $F \subseteq (P \times T) \cup (T \times P)$.

Graphische Darstellung

- Plätze durch Kreise,
- Transitionen durch Kästchen,
- Flussrelation durch Pfeile.

Definition 10.17

Sei $\mathcal{N} = (P, T, F)$ ein Petrinetz.

Der Vorbereich einer Transition $t \in T$ ist

$$^\bullet t := \{p \in P \mid (p, t) \in F\}$$

Der Nachbereich von t ist

$$t^\bullet := \{p \in P \mid (t, p) \in F\}.$$

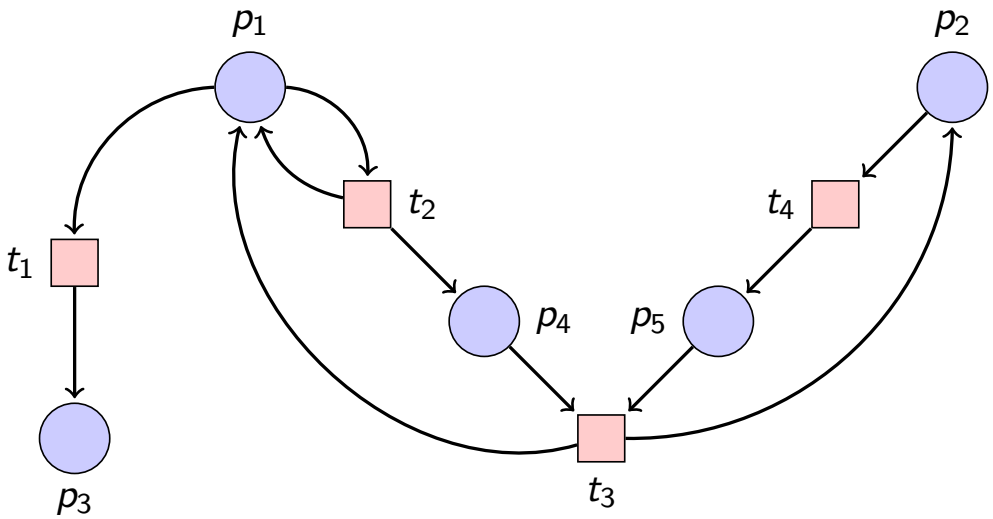
Analog für Plätze: Der Vorbereich eines Platzes $p \in P$ ist

$$^\bullet p := \{t \in T \mid (t, p) \in F\},$$

und der Nachbereich von p ist

$$p^\bullet := \{t \in T \mid (p, t) \in F\}.$$

Beispiel 10.18



Vorbereiche

$^\bullet p_1 = \{t_2, t_3\}$

$^\bullet t_2 = \{p_1\}$

Nachbereiche

$p_1^\bullet = \{t_1, t_2\}$

$t_2^\bullet = \{p_1, p_4\}$

Definition 10.19

Eine Markierung eines Petrinetzes $\mathcal{N} = (P, T, F)$ ist eine Funktion

$$M : P \to \mathbb{N},$$

die jedem Platz p eine Markenzahl $M(p)$ zuordnet.

Intuitive Bedeutung

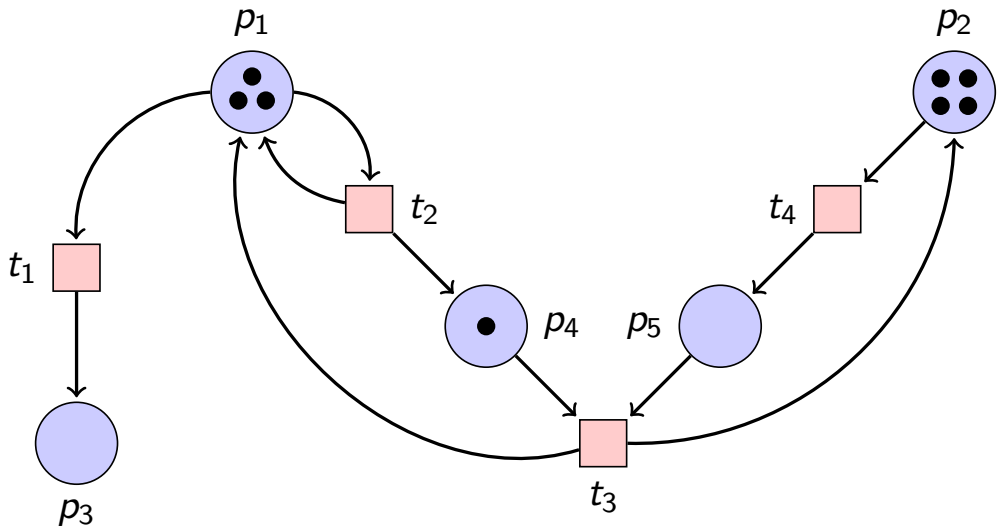
Markierungen sind die globalen Zustände von Petrinetzen.

Markierungen als Vektoren

Haben wir eine Aufzählung $P = \{p_1, \ldots, p_n\}$ festgelegt, so können wir Markierungen als Vektoren in $\mathbb{R}^n$ auffassen:

$$M = \big(M(p_1), M(p_2), \ldots, M(p_n)\big).$$

Beispiel 10.20



Markierung

$$M = (3, 4, 0, 1, 0)$$

Idee der Schaltregel

- Damit eine Transition t "schalten" kann, muss auf allen Vorplätzen von t eine Marke liegen.
- Schaltet t, so wird von allen Vorplätzen von t eine Marke weggenommen und zu allen Nachplätzen eine Marke hinzugefügt.

Schaltregel

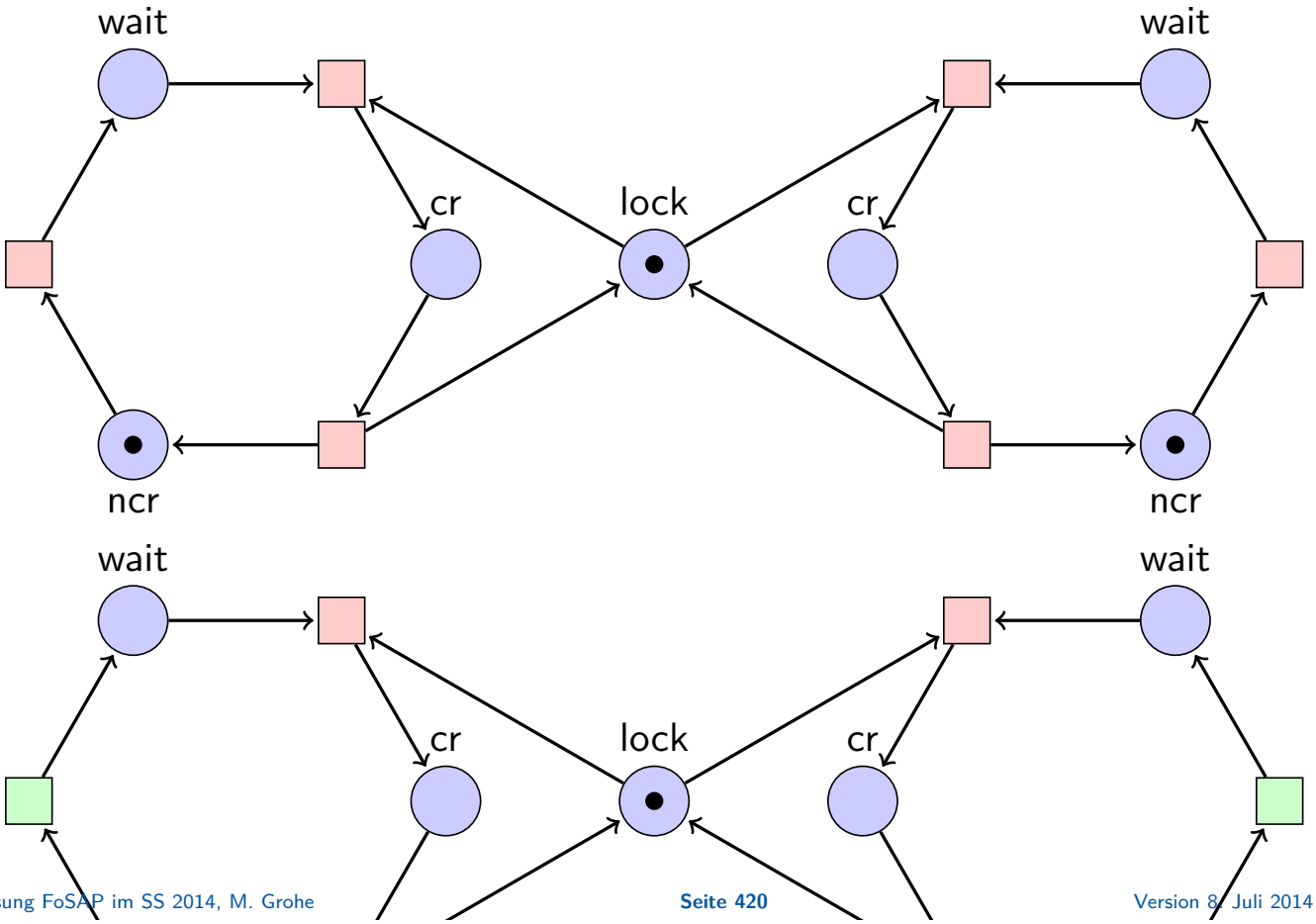
Definition 10.21

Seien $\mathcal{N} = (P, T, F)$ ein Petrinetz, M, M' Markierungen von $\mathcal{N}$, und $t \in T$.

1. t ist aktiviert in M, wenn $M(p) > 0$ für alle $p \in {}^\bullet t$.
2. t schaltet von M nach M' (wir schreiben $M \xrightarrow{t}_{\mathcal{N}} M'$), wenn t in M aktiviert ist und

$$M'(p) = \begin{cases} M(p) - 1 & \text{wenn } p \in {}^\bullet t \setminus t^\bullet \\ M(p) + 1 & \text{wenn } p \in t^\bullet \setminus {}^\bullet t \\ M(p) & \text{sonst.} \end{cases}$$

Beispiel 10.15 (Forts.)



Läufe und Erreichbarkeit

Definition 10.22

Seien $\mathcal{N} = (P, T, F)$ ein Petrinetz und M, M' Markierungen.

Ein Lauf von M nach M' der Länge ℓ ist eine Folge $M_0, t_1, M_1, t_2, \ldots, t_\ell, M_\ell$ von Transitionen und Markierungen, so dass $\ell \geq 0$ und $M = M_0$ und $M' = M_\ell$ und

$$M_0 \xrightarrow{t_1}_{\mathcal{N}} M_1 \xrightarrow{t_2}_{\mathcal{N}} M_2 \xrightarrow{t_3}_{\mathcal{N}} \ldots \xrightarrow{t_n}_{\mathcal{N}} M_\ell.$$

Notation

- Wir schreiben $M \to_{\mathcal{N}} M'$, wenn es eine Transition $t \in T$ gibt, so dass $M \xrightarrow{t}_{\mathcal{N}} M'$.
- Wir schreiben $M \xrightarrow{*}_{\mathcal{N}} M'$, wenn es einen Lauf von M nach M' gibt.
- Wir lassen den Index $_{\mathcal{N}}$ weg, wenn $\mathcal{N}$ aus dem Kontext hervorgeht.

Oft gibt man ein Netz $\mathcal{N}$ zusammen mit einer Anfangsmarkierung M_0 an.

Definition 10.23

Sei $\mathcal{N} = (P, T, F)$ ein Petrinetz und M_0 eine Anfangsmarkierung.

1. Ein Lauf von $(\mathcal{N}, M_0)$ ist eine endliche oder unendliche Folge $M_0, t_1, M_1, t_2, \ldots$, so dass für alle i das Anfangsstück $M_0, t_1, M_1, t_2, \ldots, t_i, M_i$ ein Lauf von M_0 nach M_i in $\mathcal{N}$ ist.
2. Eine Markierung M ist erreichbar in $(\mathcal{N}, M_0)$, wenn $M_0 \xrightarrow{*} M$.

Bemerkung 10.24

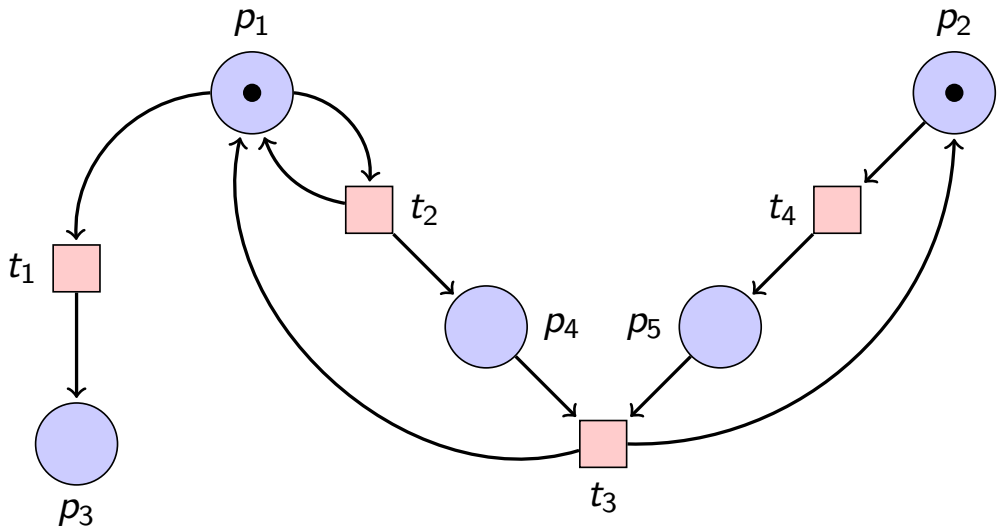
Wir gehen in unserer Definition von Läufen davon aus, dass zu jedem Zeitpunkt nur eine Transition schaltet. Man kann auch kompliziertere Abläufe von Netzen betrachten, in denen man es Transitionen mit disjunkten Vorbereichen erlaubt, unabhängig voneinander gleichzeitig zu schalten.

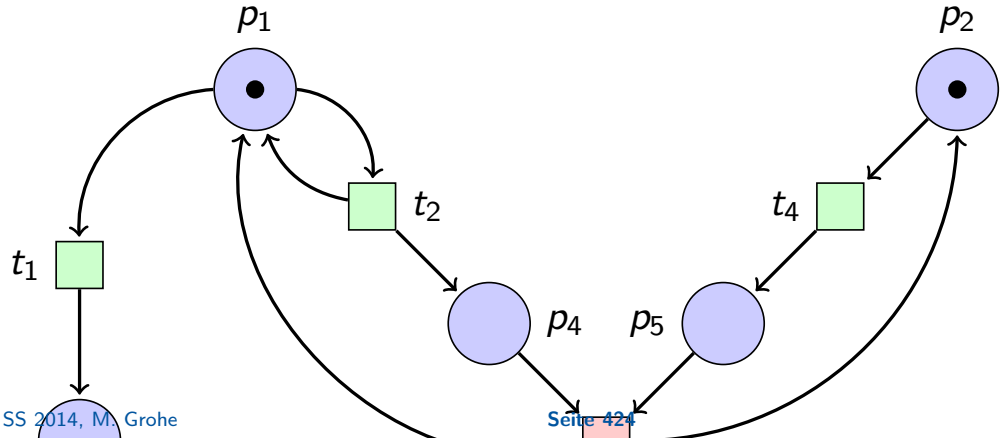
Definition 10.25

Sei $\mathcal{N} = (P, T, F)$ ein Petrinetz und M eine Markierung.

1. M ist eine Verklemmung von $\mathcal{N}$ (oder ein Deadlock, oder auch, $(\mathcal{N}, M)$ ist verklemmt), wenn es keine Transition $t \in T$ gibt, die in M aktiviert ist.
2. $(\mathcal{N}, M)$ ist beschränkt, wenn nur endlich viele Markierungen erreichbar sind.
   Sonst ist $\mathcal{N}, M$ unbeschränkt.

Beispiel 10.26





Analyse von Petrinetzen

Kernfrage jeder Systemanalyse

Wird das System in einem Initialzustand gestartet, kann es dann gewisse (gewünschte oder unerwünschte) Zustände erreichen?

Eine einfaches Werkzeug zur Analyse von Petrinetzen ist der Erreichbarkeitsbaum.

Definition 10.27

Sei $\mathcal{N} = (P, T, F)$ ein Petrinetz und M_0 eine Anfangsmarkierung.

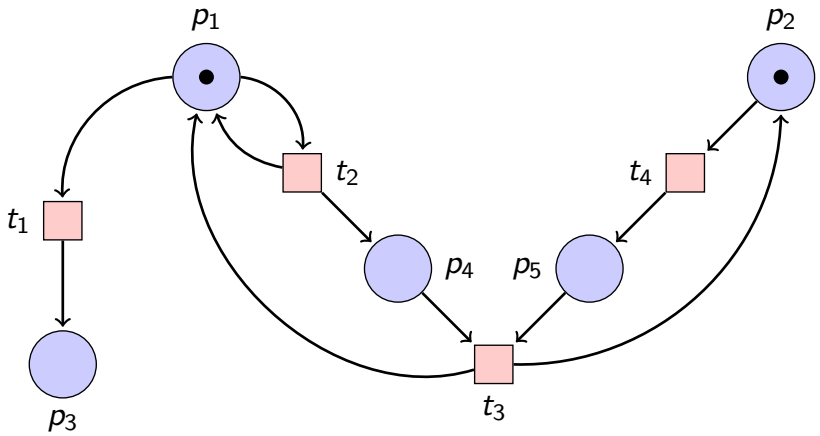
Der Erreichbarkeitsbaum $T[\mathcal{N}, M_0]$ ist wie folgt definiert:

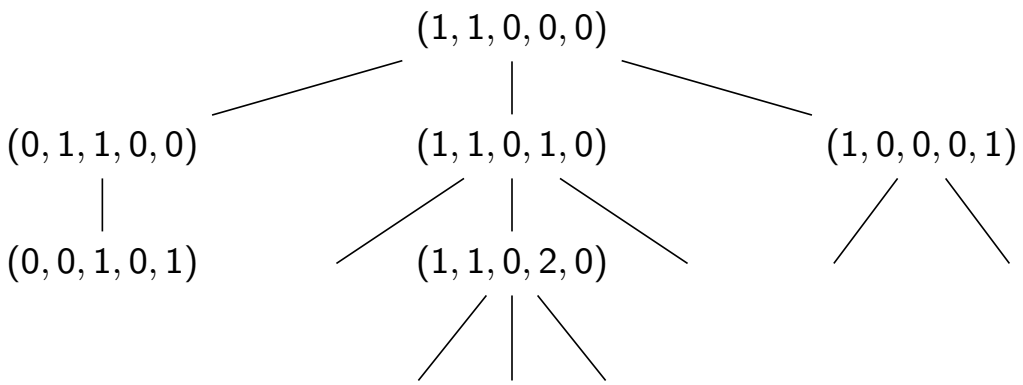
- $T[\mathcal{N}, M_0]$ ist ein endlicher oder unendlicher Baum, dessen Knoten mit Markierungen von $\mathcal{N}$ beschriftet sind.
- Die Wurzel von $T[\mathcal{N}, M_0]$ ist mit M_0 beschriftet.
- Ein mit M beschrifteter Knoten hat ein mit M' beschriftetes Kind für alle M' mit $M \to M'$.

Bemerkung 10.28

Der Erreichbarkeitsbaum $T[\mathcal{N}, M_0]$ enthält genau die in $(\mathcal{N}, M_0)$ erreichbaren Markierungen.

Beispiel 10.26 (Forts.)

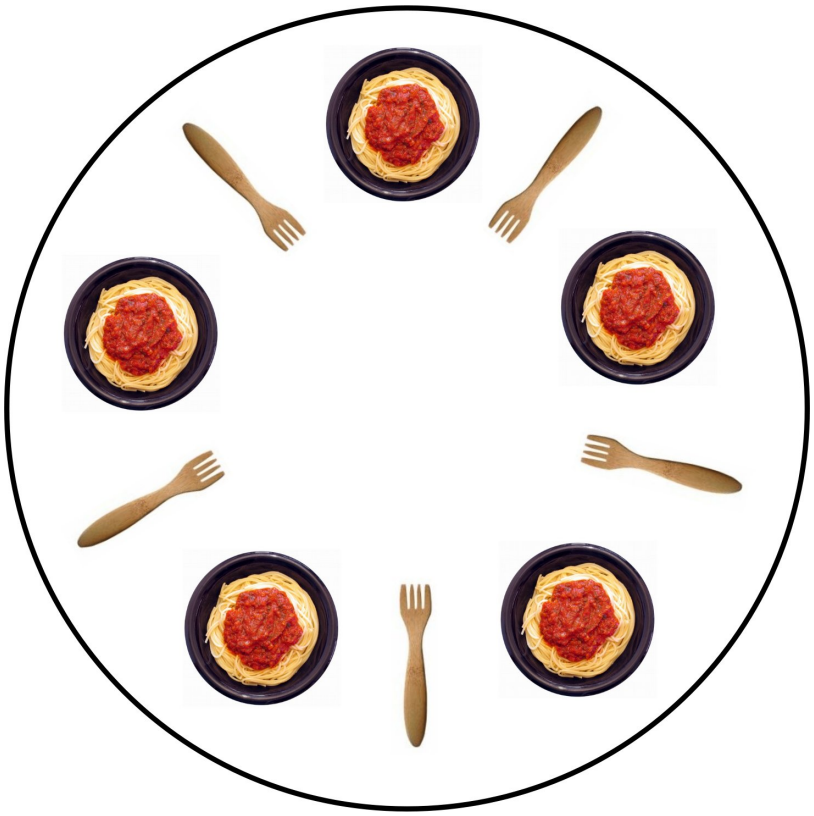




- Am Erreichbarkeitsbaum kann man sehen, dass eine Markierung erreicht wird.
  Er ist ungeeignet, um zu zeigen, dass eine Markierung nicht erreicht wird.
- Es gibt ein sehr kompliziertes Verfahren, mit dem man entscheiden kann, ob man im Netz $\mathcal{N}$ von einer Markierung M eine andere gegebene Markierung M' erreichen kann.
- Bei beschränkten Netzen liefert bereits der Erreichbarkeitsbaum (dann endlich!) diese Auskunft.

  Problem: Die nicht handhabbare Größe des Baumes

Beispiel 10.29: Dining Philosophers



Dining Philosophers (Forts.)

- Fünf Philosophen sitzen um den Tisch, sie denken und essen im Wechsel.
- Zum Essen benötigt ein Philosoph die linke und rechte Gabel.
- Problem: Finde ein Protokoll des Gabelzugriffs, das sicherstellt, dass jeder Philosoph immer wieder einmal zum Essen kommt.

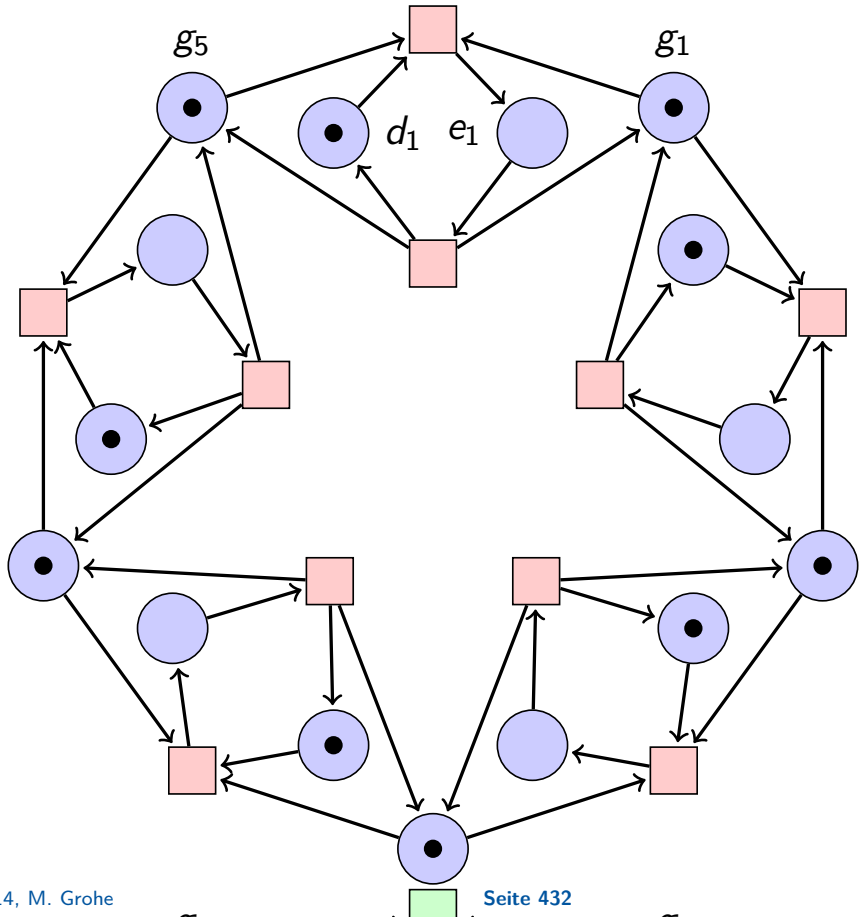
Bemerkung 10.30

Es handelt sich um ein klassisches Beispiel aus der Theorie nebenläufiger Prozesse (eingeführt von Dijkstra), das ein typisches Problem des Zugriffs nebenläufiger Prozesse auf gemeinsam verwendete Ressourcen illustriert.

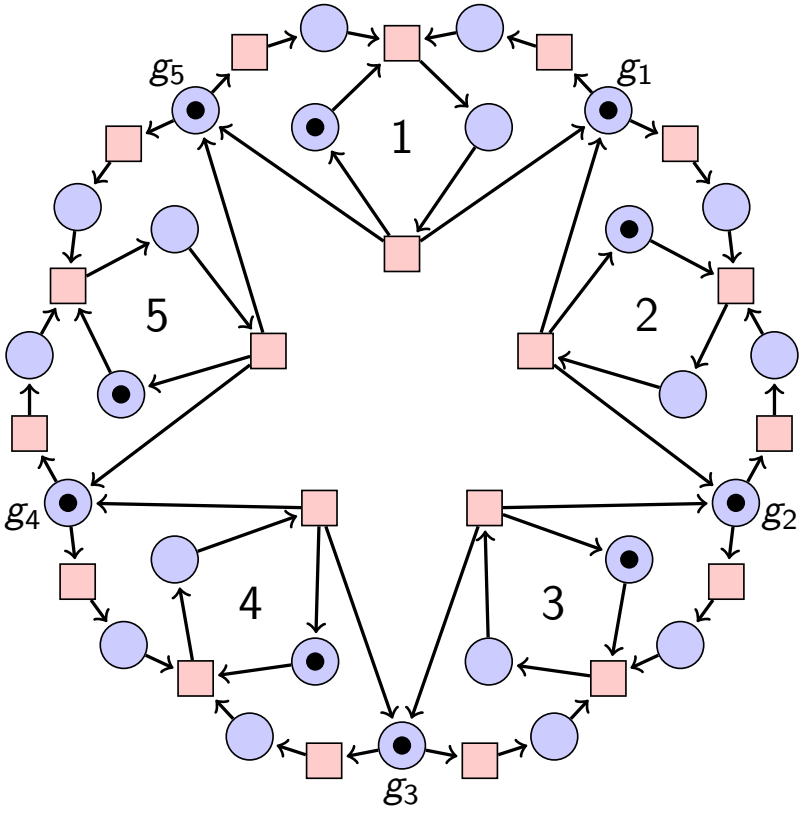
Edsger Dijkstra
(1930*)

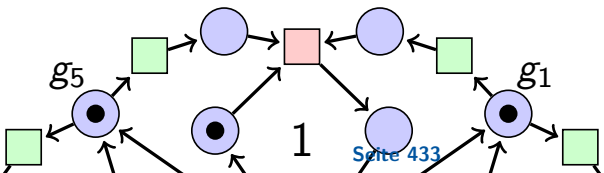
ACM Turing Award 1972

Dining Philosophers (Modellierung als Petrinetz)

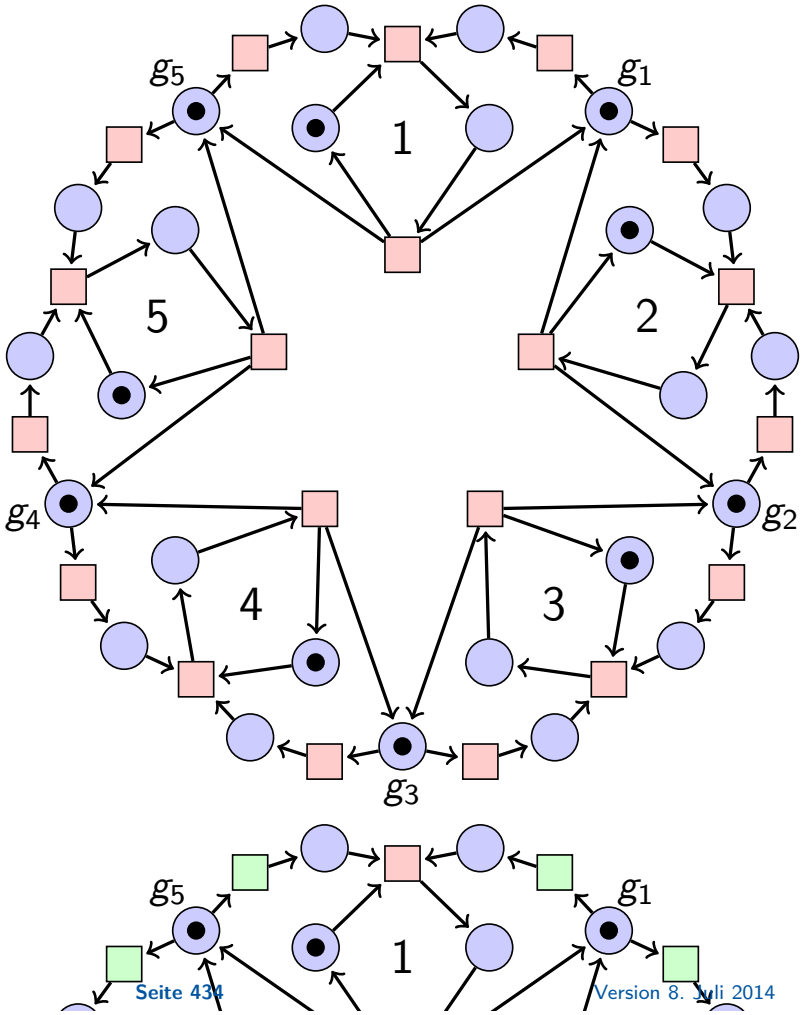


Dining Philosophers (Verfeinertes Modell)

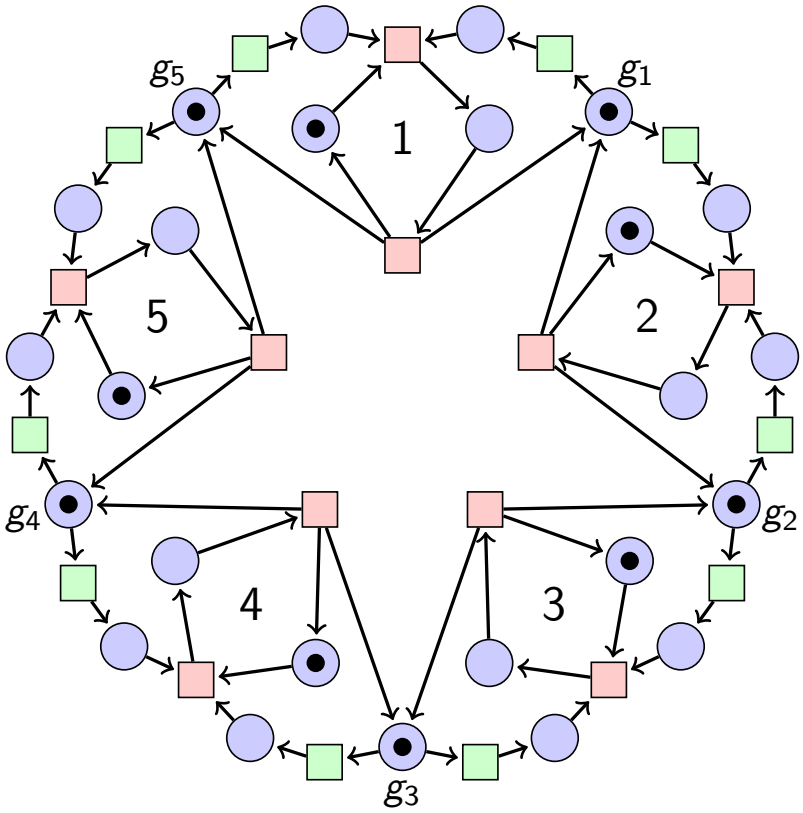


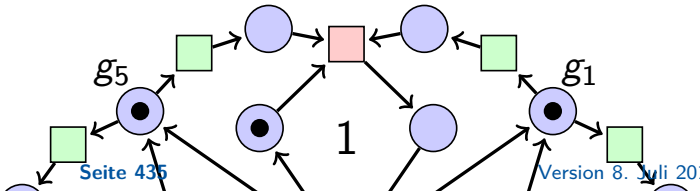


Dining Philosophers (Naives Protokoll)



Dining Philosophers (Dijkstras Protokoll)





Zusammenfassung

- Petrinetze sind ein einfaches und anschauliches, aber dennoch sehr mächtiges Werkzeug zur Modellierung nebenläufiger Systeme.
- Sie können, im Gegensatz etwa zu synchronisierten Produkten endlicher Automaten, unendliche Zustandsräume haben.
- In einer Erweiterung der elementaren Petrinetzversion, die wir betrachtet haben, kann man unterschiedliche Arten von Marken verwenden, um so verschiedene Ressourcen zu modellieren.
- Ein Nachteil von Petrinetzen ist, dass sie sich nicht ohne Weiteres in ihre Komponenten zerlegen oder modular zusammensetzen lassen.