

## Vorlesung „Methoden des Software Engineering“

Block C „Formale Methoden“  
**Petrinetze**

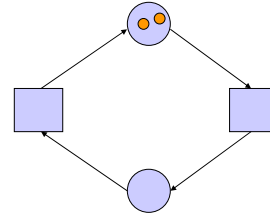
Martin Wirsing

Einheit C.3, 9.12.2004

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

**Ziele**

- Sprachen zur operationellen Beschreibung reaktiver Systeme kennen lernen:
  - Petri-Netze



Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

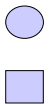
**Petrinetze**

- **Petri-Netz** (auch Stellen-/Transitions-Netz oder P/T-Net):  
Formaler Kalkül zur **Modellierung von Abläufen mit nebenläufigen Prozessen** und **kausalen Beziehungen**
- Basiert auf **bipartiten, gerichteten Graphen**:
  - **Knoten** repräsentieren **Bedingungen, Zustände** bzw. **Aktivitäten**.
  - **Kanten** verbinden **Aktivitäten** mit ihren **Vor- und Nachbedingungen**.
  - **Knotenmarkierung** repräsentiert den veränderlichen **Zustand des Systems**.
  - **graphische Notation**.
- **C. A. Petri** hat sie 1962 eingeführt.
- Es gibt zahlreiche **Varianten und Verfeinerungen von Petri-Netzen**. Hier nur die Grundform.
- **Anwendungen** von Petri-Netzen zur **Modellierung von**
  - realen oder abstrakten Automaten und Maschinen
  - kommunizierenden Prozessen in der Realität oder in Rechnern
  - Verhalten von Hardware-Komponenten/Geschäftsabläufen
  - **Semantik von UML 2.0 Aktivitätsdiagrammen**

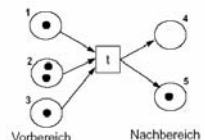
Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

**Petrinetze**

- Ein **Petri-Netz** ist ein Tripel  $P = (S, T, F)$  mit
  - S** Menge von **Stellen**, repräsentieren Bedingungen, Zustände; graphisch Kreise
  - T** Menge von **Transitionen** oder **Übergänge**, repräsentieren Aktivitäten; graphisch Rechtecke
  - F** **Relation** mit  $F \subseteq S \times T \cup T \times S$  repräsentieren kausale oder zeitliche Vor-, Nachbedingungen von Aktivitäten aus T



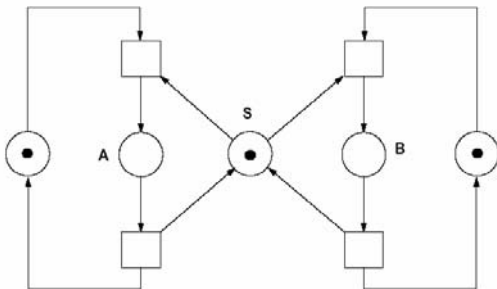
- P bildet einen **bipartiten, gerichteten Graph** und den Kanten F.
- Zu einer **Transition t** in einem Petri-Netz  $\tau$  sind folgende Stellenmengen definiert
  - $\cdot t = \{s \mid (s, t) \in F\}$  (Vorbereich)
  - $t \cdot = \{s \mid (t, s) \in F\}$  (Nachbereich)



Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

**Petrinetz: Beispiel gegenseitiger Ausschluss**

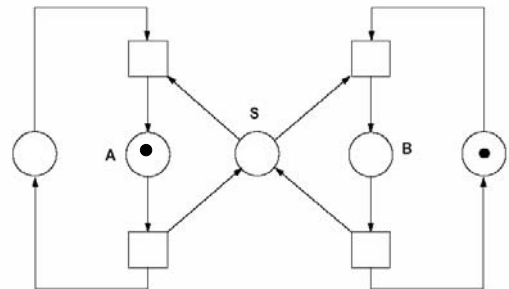
- Das Petri-Netz modelliert zwei **zyklisch ablaufende** Prozesse.
- Die mittlere Stelle synchronisiert die beiden Prozesse, so dass sie sich **nicht zugleich** in den Zuständen A und B befinden können.
- Prinzip: **gegenseitiger Ausschluss** durch **Semaphor**



Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

**Petrinetz: Beispiel gegenseitiger Ausschluss**

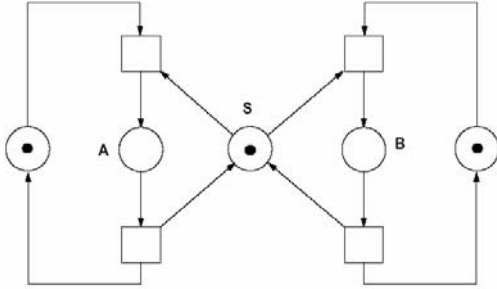
- Das Petri-Netz modelliert zwei **zyklisch ablaufende** Prozesse.
- Die mittlere Stelle synchronisiert die beiden Prozesse, so dass sie sich **nicht zugleich** in den Zuständen A und B befinden können.
- Prinzip: **gegenseitiger Ausschluss** durch **Semaphor**



Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

### Petrinetz: Beispiel gegenseitiger Ausschluss

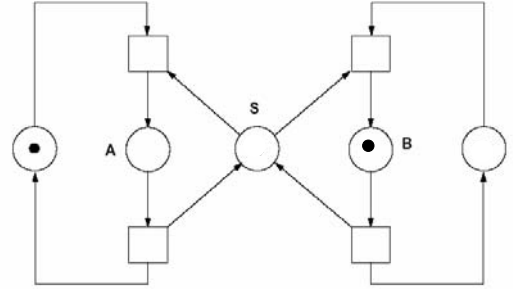
- Das Petri-Netz modelliert zwei **zyklisch ablaufende** Prozesse.
- Die mittlere Stelle synchronisiert die beiden Prozesse, so dass sie sich **nicht zugleich** in den Zuständen A und B befinden können.
- Prinzip: **gegenseitiger Ausschluss** durch **Semaphor**



Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

### Petrinetz: Beispiel gegenseitiger Ausschluss

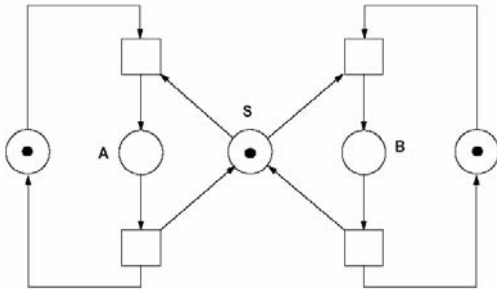
- Das Petri-Netz modelliert zwei **zyklisch ablaufende** Prozesse.
- Die mittlere Stelle synchronisiert die beiden Prozesse, so dass sie sich **nicht zugleich** in den Zuständen A und B befinden können.
- Prinzip: **gegenseitiger Ausschluss** durch **Semaphor**



Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

### Petrinetz: Beispiel gegenseitiger Ausschluss

- Das Petri-Netz modelliert zwei **zyklisch ablaufende** Prozesse.
- Die mittlere Stelle synchronisiert die beiden Prozesse, so dass sie sich **nicht zugleich** in den Zuständen A und B befinden können.
- Prinzip: **gegenseitiger Ausschluss** durch **Semaphor**



Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

### Schaltregel für Petrinetze

Das **Schalten einer Transition t** überführt eine Markierung M in eine Markierung M'. Eine **Transition t kann schalten**, wenn für alle Stellen  $s \in {}^*t$  gilt  $M(s) \geq 1$ .

Wenn eine Transition t **schaltet**, gilt für die **Nachfolgemarkierung M'**:

$$M'(v) = M(v) - 1 \quad \text{für alle } v \in {}^*t \setminus t$$

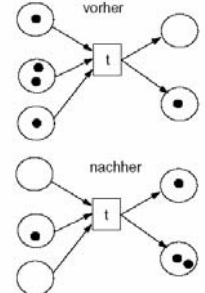
$$M'(n) = M(n) + 1 \quad \text{für alle } n \in t \setminus {}^*t$$

$$M'(s) = M(s) \quad \text{sonst}$$

Wenn in einem Schritt **mehrere Transitionen schalten können**, wird eine davon **nicht-deterministisch** ausgewählt.

In jedem Schritt **schaltet genau eine Transition** –auch wenn das Petri-Netz parallele Abläufe modelliert!

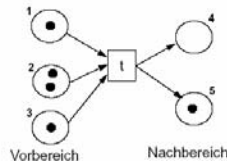
Zwei Transitionen mit gemeinsamen Stellen im Vorbereich können (bei passender Markierung) im **Konflikt** stehen: Jede kann schalten, aber nicht beide nacheinander.



Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

### Markierungen (I)

- Der **Zustand des Petri-Netzes** wird durch eine Markierungsfunktion angegeben, die jeder Stelle eine Anzahl von Marken zuordnet:  
 $M_p: S \rightarrow \mathbb{N}_0$
- Sind die Stellen von 1 bis n nummeriert, so kann man  $M_p$  als Folge angeben, z.B. (1,2,1,0,1)

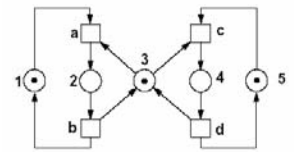


Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

### Markierungen (II)

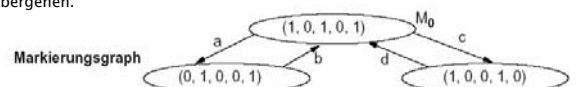
Zu jedem Petri-Netz wird eine **Anfangsmarkierung**  $M_0$  angegeben.  
z.B.  $M_0 = (1, 0, 1, 0, 1)$

Wir sagen, eine Markierung  $M_2$  ist von einer Markierung  $M_1$  **erreichbar**, wenn es ausgehend von  $M_1$  eine Folge von Transitionen gibt, die nacheinander schalten und  $M_1$  in  $M_2$  überführen können



Die Markierungen eines Petri-Netzes kann man als **gerichteten Markierungsgraphen** darstellen:

- **Knoten**: erreichbare Markierung
- **Kante  $M_1 \rightarrow M_2$** : Die Markierung  $M_1$  kann durch Schalten einer Transition in  $M_2$  übergehen.



Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

## Schaltfolgen

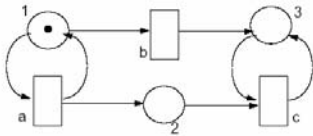
Schaltfolgen kann man angeben als

- Folge von Markierungen
- Folge der geschalteten Transitionen

Beispiel für eine Schaltfolgen zum Petri-Netz auf Folie 17:

- (1, 0, 1, 0, 1) a
- (0, 1, 0, 0, 1) b
- (1, 0, 1, 0, 1) c
- (1, 0, 0, 1, 0) d
- (1, 0, 1, 0, 1) d

Schaltfolgen können als Wörter einer Sprache aufgefasst werden.

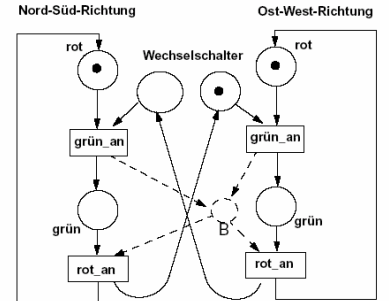


alle Schaltfolgen ohne Nachfolgemarkierung haben die Form  $a^n b c^n$

Petri-Netze können unbegrenzt zählen: Anzahl der Marken auf einer Stelle

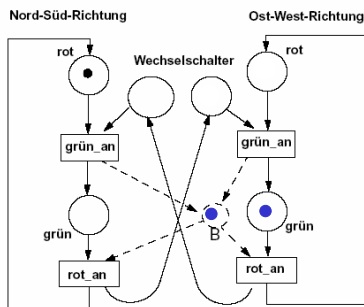
## Schaltfolgen: Beispiel alternierende Ampel

- 2 sich zyklisch wiederholende Prozesse
- Die beiden Stellen „Wechselschalter“ koppeln die Prozesse, sodass sie alternierend fortschreiten
- Alle Stellen repräsentieren Bedingungen: 1 oder 0 Marke
- „Beobachtungsstelle“ B modelliert, wieviele Richtungen „grün“ haben



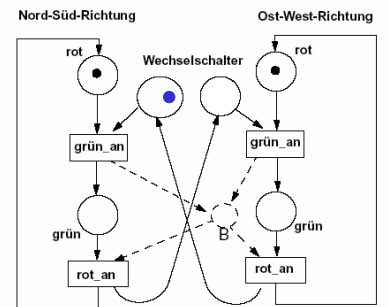
## Schaltfolgen: Beispiel alternierende Ampel

- 2 sich zyklisch wiederholende Prozesse
- Die beiden Stellen „Wechselschalter“ koppeln die Prozesse, sodass sie alternierend fortschreiten
- Alle Stellen repräsentieren Bedingungen: 1 oder 0 Marke
- „Beobachtungsstelle“ B modelliert, wieviele Richtungen „grün“ haben



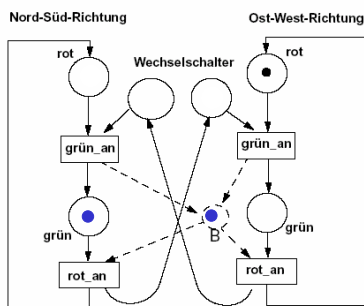
## Schaltfolgen: Beispiel alternierende Ampel

- 2 sich zyklisch wiederholende Prozesse
- Die beiden Stellen „Wechselschalter“ koppeln die Prozesse, sodass sie alternierend fortschreiten
- Alle Stellen repräsentieren Bedingungen: 1 oder 0 Marke
- „Beobachtungsstelle“ B modelliert, wieviele Richtungen „grün“ haben



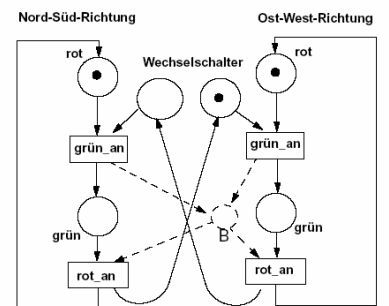
## Schaltfolgen: Beispiel alternierende Ampel

- 2 sich zyklisch wiederholende Prozesse
- Die beiden Stellen „Wechselschalter“ koppeln die Prozesse, sodass sie alternierend fortschreiten
- Alle Stellen repräsentieren Bedingungen: 1 oder 0 Marke
- „Beobachtungsstelle“ B modelliert, wieviele Richtungen „grün“ haben



## Schaltfolgen: Beispiel alternierende Ampel

- 2 sich zyklisch wiederholende Prozesse
- Die beiden Stellen „Wechselschalter“ koppeln die Prozesse, sodass sie alternierend fortschreiten
- Alle Stellen repräsentieren Bedingungen: 1 oder 0 Marke
- „Beobachtungsstelle“ B modelliert, wieviele Richtungen „grün“ haben

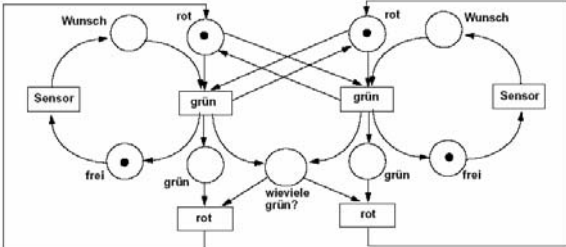


## Sichere Netze

Ein Petri-Netz heißt **binär (sicher)**, wenn für alle aus  $M_0$  erreichbaren Markierungen  $M$  und für alle Stellen  $s$  gilt  $M(s) \leq 1$ .

Petri-Netze, deren **Stellen Bedingungen repräsentieren**, müssen binär sein.

**Beispiel: Modellierung einer Sensor-gesteuerten Ampelkreuzung:**



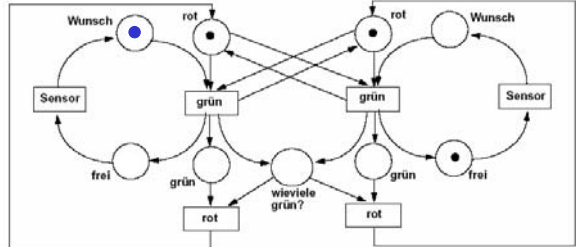
Aus B. Baumgarten: Petri-Netze, Bibliographisches Institut & F.A.Brockhaus AG, 1990

## Sichere Netze

Ein Petri-Netz heißt **binär (sicher)**, wenn für alle aus  $M_0$  erreichbaren Markierungen  $M$  und für alle Stellen  $s$  gilt  $M(s) \leq 1$ .

Petri-Netze, deren **Stellen Bedingungen repräsentieren**, müssen binär sein.

**Beispiel: Modellierung einer Sensor-gesteuerten Ampelkreuzung:**



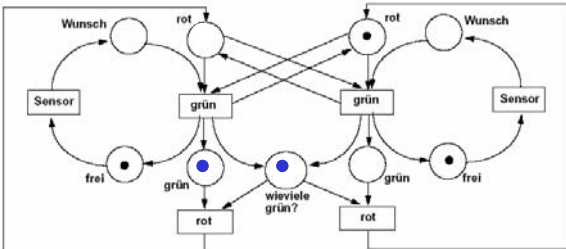
Aus B. Baumgarten: Petri-Netze, Bibliographisches Institut & F.A.Brockhaus AG, 1990

## Sichere Netze

Ein Petri-Netz heißt **binär (sicher)**, wenn für alle aus  $M_0$  erreichbaren Markierungen  $M$  und für alle Stellen  $s$  gilt  $M(s) \leq 1$ .

Petri-Netze, deren **Stellen Bedingungen repräsentieren**, müssen binär sein.

**Beispiel: Modellierung einer Sensor-gesteuerten Ampelkreuzung:**



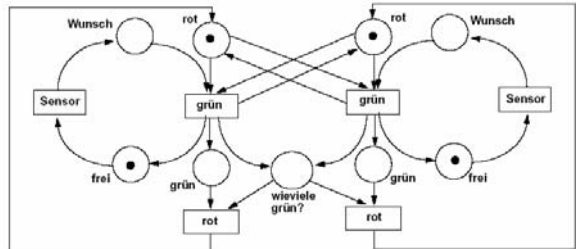
Aus B. Baumgarten: Petri-Netze, Bibliographisches Institut & F.A.Brockhaus AG, 1990

## Sichere Netze

Ein Petri-Netz heißt **binär (sicher)**, wenn für alle aus  $M_0$  erreichbaren Markierungen  $M$  und für alle Stellen  $s$  gilt  $M(s) \leq 1$ .

Petri-Netze, deren **Stellen Bedingungen repräsentieren**, müssen binär sein.

**Beispiel: Modellierung einer Sensor-gesteuerten Ampelkreuzung:**



Aus B. Baumgarten: Petri-Netze, Bibliographisches Institut & F.A.Brockhaus AG, 1990

## Lebendige Netze

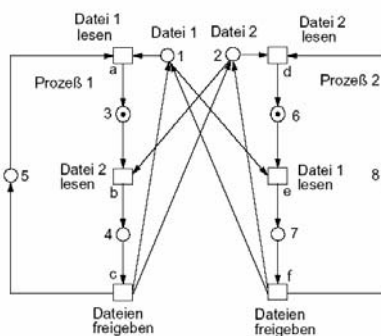
- Petri-Netze modellieren häufig **Systeme, die nicht anhalten** sollen.
- Ein Petri-Netz heißt **schwach lebendig**, wenn es zu jeder von  $M_0$  erreichbaren Markierung eine Nachfolgemarkierung gibt.
- Eine **Transition  $t$  heißt lebendig**, wenn es zu jeder von  $M_0$  erreichbaren Markierung  $M$  eine Markierung  $M'$  gibt, die von  $M$  erreichbar ist, und in der  $t$  schalten kann.
- Ein Petri-Netz heißt **lebendig**, wenn alle seine Transitionen lebendig sind.

weil **das Schalten einiger Transitionen zyklisch voneinander abhängt**.

Sei  $\sigma \subseteq S$  eine Teilmenge der Stellen eines Petri-Netzes und  
 Vorbereich ( $\sigma$ ) =  $\{t \mid \exists s \in \sigma : (t, s) \in F\}$ ,  
 d.h. die Transitionen, die auf Stellen in  $\sigma$  wirken  
 Nachbereich ( $\sigma$ ) =  $\{t \mid \exists s \in \sigma : (s, t) \in F\}$   
 d.h. die Transitionen, die auf Stellen in  $\sigma$  als Vorbedingung haben

Dann ist  $\sigma$  eine **Verklemmung**, wenn **Vorbereich ( $\sigma$ )  $\subseteq$  Nachbereich ( $\sigma$ )**.  
 Wenn **für alle  $s \in \sigma$  gilt  $M(s) = 0$** , dann kann es **keine Marken auf Stellen in  $\sigma$**  in einer Nachfolgemarkierung von  $M$  geben

## Verklemmung



$$\sigma = \{1, 2, 4, 5, 7, 8\}$$

$$\text{Vorbereich } (\sigma) = \{b, c, e, f\}$$

$$\text{Nachbereich } (\sigma) = \{a, b, c, d, e, f\}$$

$$M(\sigma) = 0$$

## Kapazitäten und Gewichte

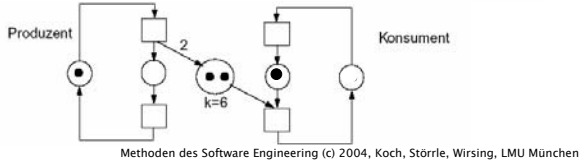
Man kann Stellen eine begrenzte Kapazität von  $k \in \mathbb{N}$  Marken zuordnen.

Die Bedingung, dass **eine Transition t schalten kann**, wird erweitert um:

Die **Kapazität keiner der Stellen im Nachbereich von t darf überschritten** werden.

**Kanten** kann ein Gewicht  $n \in \mathbb{N}$  zugeordnet werden: sie bewegen **beim Schalten n Marken**.

Beispiel: **Beschränkter Puffer**



Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

## Kapazitäten und Gewichte

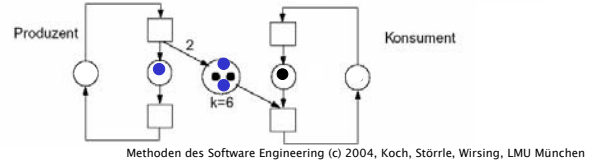
Man kann Stellen eine begrenzte Kapazität von  $k \in \mathbb{N}$  Marken zuordnen.

Die Bedingung, dass **eine Transition t schalten kann**, wird erweitert um:

Die **Kapazität keiner der Stellen im Nachbereich von t darf überschritten** werden.

**Kanten** kann ein Gewicht  $n \in \mathbb{N}$  zugeordnet werden: sie bewegen **beim Schalten n Marken**.

Beispiel: **Beschränkter Puffer**



Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

## Kapazitäten und Gewichte

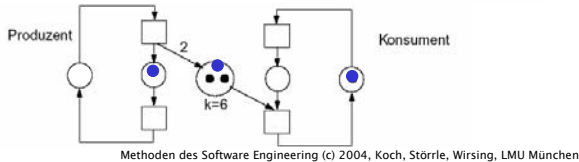
Man kann Stellen eine begrenzte Kapazität von  $k \in \mathbb{N}$  Marken zuordnen.

Die Bedingung, dass **eine Transition t schalten kann**, wird erweitert um:

Die **Kapazität keiner der Stellen im Nachbereich von t darf überschritten** werden.

**Kanten** kann ein Gewicht  $n \in \mathbb{N}$  zugeordnet werden: sie bewegen **beim Schalten n Marken**.

Beispiel: **Beschränkter Puffer**



Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

## Kapazitäten und Gewichte

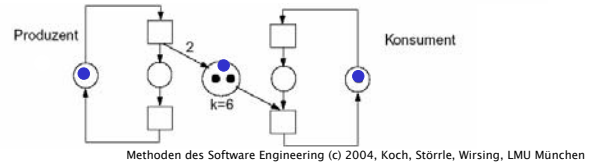
Man kann Stellen eine begrenzte Kapazität von  $k \in \mathbb{N}$  Marken zuordnen.

Die Bedingung, dass **eine Transition t schalten kann**, wird erweitert um:

Die **Kapazität keiner der Stellen im Nachbereich von t darf überschritten** werden.

**Kanten** kann ein Gewicht  $n \in \mathbb{N}$  zugeordnet werden: sie bewegen **beim Schalten n Marken**.

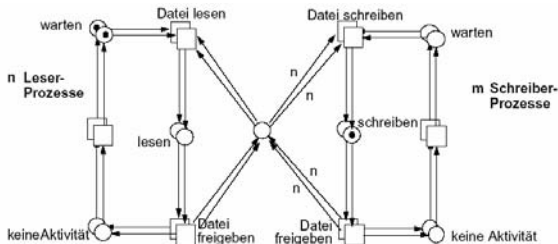
Beispiel: **Beschränkter Puffer**



Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

## Leser- /Schreiber-System

- $n$  **Leser-Prozesse** und  $m$  **Schreiber-Prozesse** operieren auf derselben Datei.
- **Mehrere Leser** können **zugleich** lesen.
- Ein Schreiber darf **nur dann schreiben, wenn kein anderer** Leser oder Schreiber **aktiv** ist.
- **Modellierung:** ein Schreiber entzieht der Synchronisationsstelle alle  $n$  Marken.

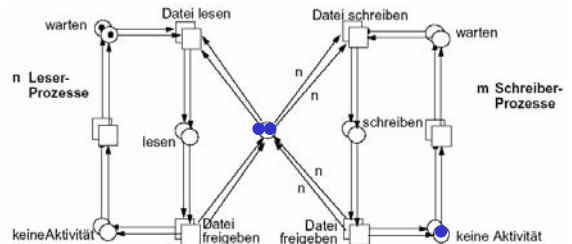


Aus B. Baumgarten: Petri-Netze, Bibliographisches Institut & F.A.Brockhaus AG, 1990

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

## Leser- /Schreiber-System

- $n$  **Leser-Prozesse** und  $m$  **Schreiber-Prozesse** operieren auf derselben Datei.
- **Mehrere Leser** können **zugleich** lesen.
- Ein Schreiber darf **nur dann schreiben, wenn kein anderer** Leser oder Schreiber **aktiv** ist.
- **Modellierung:** ein Schreiber entzieht der Synchronisationsstelle alle  $n$  Marken.

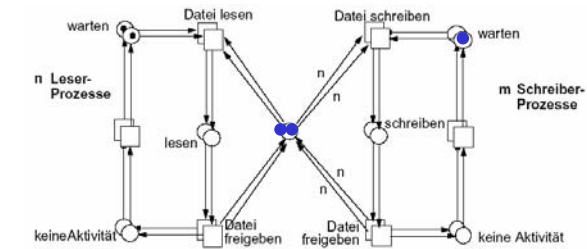


Aus B. Baumgarten: Petri-Netze, Bibliographisches Institut & F.A.Brockhaus AG, 1990

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

## Leser- /Schreiber-System

- $n$  Leser-Prozesse und  $m$  Schreiber-Prozesse operieren auf derselben Datei.
- Mehrere Leser können zugleich lesen.
- Ein Schreiber darf **nur dann schreiben, wenn kein anderer** Leser oder Schreiber aktiv ist.
- **Modellierung:** ein Schreiber entzieht der Synchronisationsstelle alle  $n$  Marken.

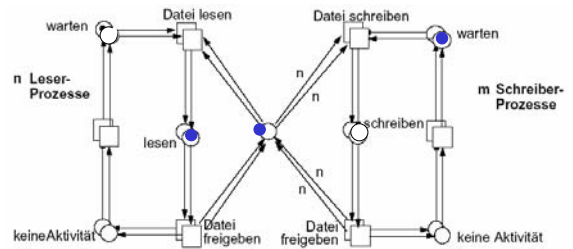


Aus B. Baumgarten: Petri-Netze, Bibliographisches Institut & F.A.Brockhaus AG, 1990

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

## Leser- /Schreiber-System

- $n$  Leser-Prozesse und  $m$  Schreiber-Prozesse operieren auf derselben Datei.
- Mehrere Leser können zugleich lesen.
- Ein Schreiber darf **nur dann schreiben, wenn kein anderer** Leser oder Schreiber aktiv ist.
- **Modellierung:** ein Schreiber entzieht der Synchronisationsstelle alle  $n$  Marken.

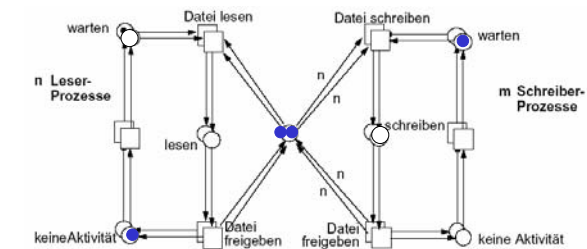


Aus B. Baumgarten: Petri-Netze, Bibliographisches Institut & F.A.Brockhaus AG, 1990

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

## Leser- /Schreiber-System

- $n$  Leser-Prozesse und  $m$  Schreiber-Prozesse operieren auf derselben Datei.
- Mehrere Leser können zugleich lesen.
- Ein Schreiber darf **nur dann schreiben, wenn kein anderer** Leser oder Schreiber aktiv ist.
- **Modellierung:** ein Schreiber entzieht der Synchronisationsstelle alle  $n$  Marken.

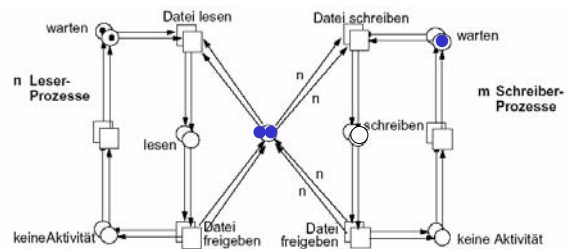


Aus B. Baumgarten: Petri-Netze, Bibliographisches Institut & F.A.Brockhaus AG, 1990

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

## Leser- /Schreiber-System

- $n$  Leser-Prozesse und  $m$  Schreiber-Prozesse operieren auf derselben Datei.
- Mehrere Leser können zugleich lesen.
- Ein Schreiber darf **nur dann schreiben, wenn kein anderer** Leser oder Schreiber aktiv ist.
- **Modellierung:** ein Schreiber entzieht der Synchronisationsstelle alle  $n$  Marken.

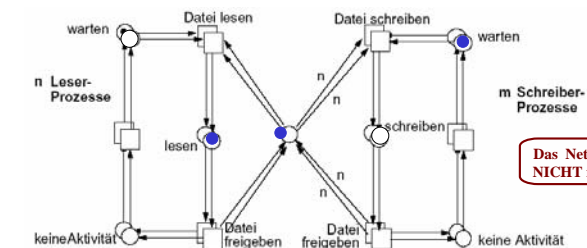


Aus B. Baumgarten: Petri-Netze, Bibliographisches Institut & F.A.Brockhaus AG, 1990

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

## Leser- /Schreiber-System

- $n$  Leser-Prozesse und  $m$  Schreiber-Prozesse operieren auf derselben Datei.
- Mehrere Leser können zugleich lesen.
- Ein Schreiber darf **nur dann schreiben, wenn kein anderer** Leser oder Schreiber aktiv ist.
- **Modellierung:** ein Schreiber entzieht der Synchronisationsstelle alle  $n$  Marken.



Aus B. Baumgarten: Petri-Netze, Bibliographisches Institut & F.A.Brockhaus AG, 1990

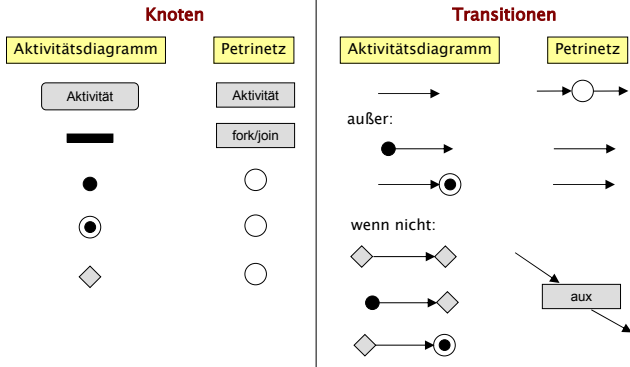
Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

## Petrinetze als Semantik für UML 2.0- Aktivitätsdiagramme (I)

- Anwendung von Petrinetzen als Semantik für Aktivitätsdiagramme (nach H. Störrle)
- Idee: Übersetzung der Elemente von Aktivitätsdiagrammen in Petrinetzkonstrukte
- Semantik berücksichtigt
  - Kontrollfluß
  - Prozeduraufruf
- Semantik berücksichtigt nicht
  - Datentypannotationen

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

### Petrinetze als Semantik für UML 2.0- Aktivitätsdiagramme (II): Übersetzung



### Petrinetze als Semantik für UML 2.0- Aktivitätsdiagramme (II): formal (nur einfach Akt.diag.)

Die Menge der Knoten sei aufgeteilt in

$$\text{Knoten} = \langle EN, iN, fN, BN, CN \rangle$$

- EN die Menge der ausführbaren Knoten
- iN, fN Start- bzw. Endknoten
- BN Verzweigungsknoten (Decision- und Merge)
- CN Nebenläufigkeitsknoten (Fork, Join, ForkJoin)

### Petrinetze als Semantik für UML 2.0- Aktivitätsdiagramme (III): formal (nur einfach Akt.diag.)

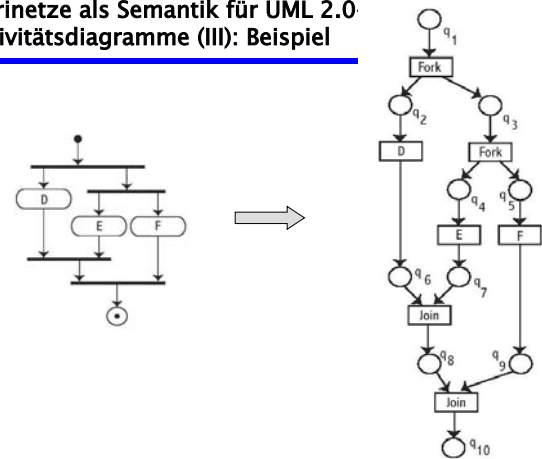
Die Übersetzung lautet damit: (mit  $(P, T, A)$  entspricht  $(S, T, F)$  und  $\bar{m}$ ,  $\underline{m}$  seien Startmarkierung und Endmarkierung)

$$[[\langle Nodes, Edges \rangle]] = \langle P, T, A, \bar{m}, \underline{m} \rangle$$

wobei

$$\begin{aligned}
 P &= \{iN, fN\} \cup BN \cup \{p_a \mid a \in Edges, \{a_1, a_2\} \cap (EN \cup CN) \neq \emptyset\}, \\
 T &= EN \cup CN \cup \{t_a \mid a \in Edges, \{a_1, a_2\} \subseteq BN \cup \{iN, fN\}\}, \\
 A &= \left\{ \langle x_{\langle from, to \rangle}, to \rangle, \langle from, x_{\langle from, to \rangle} \rangle \mid \langle from, to \rangle \in Edges \right\}, \\
 \bar{m} &= iN, \\
 \underline{m} &= fN.
 \end{aligned}$$

### Petrinetze als Semantik für UML 2.0- Aktivitätsdiagramme (III): Beispiel



### Zusammenfassung

- Petri-Netze wurden 1962 von C. A. Petri eingeführt. Sie dienen zur Modellierung von Abläufen mit nebenläufigen Prozessen und kausalen Beziehungen.
- Petri-Netze basieren auf bipartiten gerichteten Graphen:
  - Knoten repräsentieren Bedingungen, Zustände bzw. Aktivitäten.
  - Kanten verbinden Aktivitäten mit ihren Vor- und Nachbedingungen.
  - Die Knotenmarkierung repräsentiert den veränderlichen Zustand des Systems.

### Literatur

- G. Goos: Vorlesungen über Informatik, Band 1, Springer-Verlag, 1995
- H. Balzert: Lehrbuch der Software-Technik, Spektrum, 1996
- H. Störrle: Semantics of UML 2.0 Activities

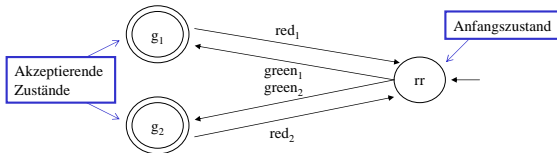
## ω-Automaten

- Alphabet** endliche Menge  $\Sigma$
- ω-Wörter**  $\alpha = a_0 a_1 \dots$  ( $a_i \in \Sigma$ )
- ω-Sprachen** Menge  $L \subseteq \Sigma^\omega$  von ω-Wörtern
- Speziell**  $\Sigma = 2^V$  interpretiert als auss.log. Belegungen über  $V$  (endlich)  
ω-Wörter über  $\Sigma$  entsprechen LTL-Strukturen

## Endliche Automaten über ω-Wörtern

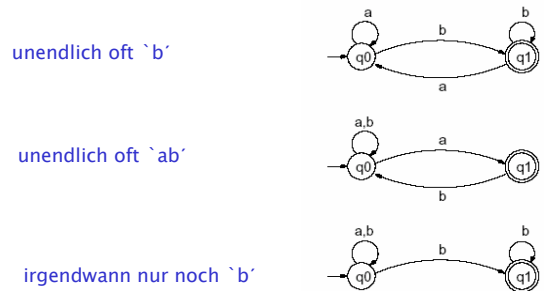
- Büchi-Automat** (über Alphabet  $\Sigma$ )  $B = (Q, I, \delta, F)$ 
  - $Q$  endliche Menge von (Automaten-) Zuständen
  - $I \subseteq Q$  Anfangszustände
  - $\delta \subseteq Q \times \Sigma \times Q$  Übergangsrelation
  - $F \subseteq Q$  akzeptierende Zustände
- Ablauf von B über ω-Wörtern**
  - Ablauf  $\rho = q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} q_2 \dots$
  - Initialisierung  $q_0 \in I$
  - Übergänge  $(q_i, a_i, q_{i+1}) \in \delta$  für alle  $i \in \mathbb{N}$
  - Akzeptanz  $q_i \in F$  für unendlich viele  $i \in \mathbb{N}$
- Sprache**  
 $L(B) = \{\alpha \in \Sigma^\omega : \text{es gibt einen akzeptierenden Ablauf von B über } \alpha = a_0 a_1 \dots\}$   
 ω-reguläre Sprachen:  
 Klasse der durch Büchi-Automaten definierbaren ω-Sprachen

## Büchi-Automaten: Beispiel Ampel



- Der Büchi-Automat akzeptiert alle ω-Wörter
- über dem Alphabet  $\{\text{red}_1, \text{red}_2, \text{green}_1, \text{green}_2\}$ ,
  - die unendlich viele  $\text{green}_1$  und unendlich viele  $\text{green}_2$  enthalten.

## Büchi-Automaten: Beispiele



## Von LTL zu Büchi-Automaten

### Prinzip

- $L(\varphi)$  bezeichne Menge der Zustandsfolgen, die  $\varphi$  erfüllen
- Konstruiere Automaten  $B_\varphi$  mit  $L(B_\varphi) = L(\varphi)$  (über Alphabet  $2^V$ )

### Grundidee der Konstruktion

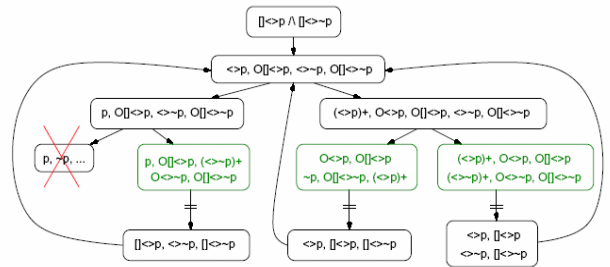
- Zustände** Mengen von „Teilformeln“ von  $\varphi$ , die „jetzt“ erfüllt werden müssen
- Anfangszustände** Zustände, die  $\varphi$  enthalten
- Übergangsrelation** sichert Erfülltheit nicht-temporaler Formeln im Ausgangszustand  
ersetzt temporale Formeln im Ausgangs- durch andere im Zielzustand  
benutzt Rekursionsgesetze zur Zerlegung temporaler Formeln

$$\begin{aligned} \square \varphi &\equiv \varphi \wedge \square \varphi \\ \diamond \varphi &\equiv \varphi \vee \diamond \varphi \\ \varphi \text{ until } \psi &\equiv \psi \vee (\varphi \wedge \square (\varphi \text{ until } \psi)) \end{aligned}$$

akzeptierende Zustände definiert aus „eventualities“  $\diamond \varphi$  oder  $\varphi$  until  $\psi$

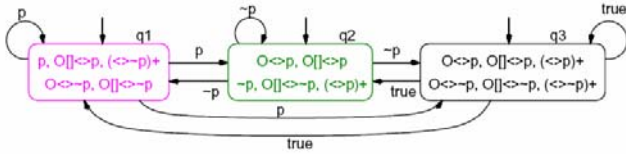
## Von LTL zu Büchi-Automaten: Beispiel

Tableau zur Formel  $\square \diamond p \wedge \diamond \neg p$



## Von LTL zu Büchi-Automaten: Beispiel

### Verallgemeinerter Büchi-Automat $\mathcal{B}_\varphi$



Akzeptanzmengen  $F = \{q1\}, \{q2\}$

## Beispiel Eisenbahnsteuerung

**Transitionssystem** aufgefasst als Büchi-Automat, bei dem alle Zustände akzeptieren

### Automatentheoretisches Modelchecking

$$\begin{aligned}
 &M \models \varphi \\
 &\text{gdw.} \\
 &\mathcal{L}(M) \subseteq \mathcal{L}(\varphi) \\
 &\text{gdw.} \\
 &\mathcal{L}(M) \cap \mathcal{L}(\neg\varphi) = \emptyset \\
 &\text{gdw.} \\
 &\mathcal{L}(M \times \mathcal{B}_{\neg\varphi}) = \emptyset
 \end{aligned}$$

**Komplexität:**  $O(|M| \cdot |\mathcal{B}_{\neg\varphi}|) = O(|M| \cdot 2^{|\varphi|})$   $|M|$  i.a. kritischer als  $2^{|\varphi|}$ !

## Zustandsexplosion

- Produktautomat  $M \times \mathcal{B}_{\neg\varphi}$  ist zu groß

problematisch ab ca.  $10^6$  Zuständen (des Produktautomaten)

- **Lösungen**
  - Reduktion            ignoriere irrelevante Teile des Zustandsraums
  - Kompression        kompakte Darstellung von  $M \times \mathcal{B}_{\neg\varphi}$  berechnen
  - Abstraktion         Projektion auf kleineren Zustandsraum (von  $M$ )