

# DISKRETE EREIGNIS-SYSTEME

S. RYFFEL

## INHALTSVERZEICHNIS

1. Automaten und Sprachen	2
1.1. Alphabete und Wörter	2
1.2. Finite Automaten	2
1.3. Sprachen	2
1.4. Reguläre Ausdrücke	2
1.5. Nondeterministic FA	3
1.6. NFA $\rightarrow$ FA	3
1.7. Generalized Nondeterministic Finite Automaton (GNFA)	4
1.8. Pumping Lemma	4
2. Intelligentere Automaten	4
2.1. Context Free Grammars (CFG)	4
2.2. Push-Down Automata (PDA)	5
2.3. Right Linear Grammars	6
2.4. Chomsky Normal Form (CNF)	6
2.5. Finite State Transducer	7
2.6. Touring Machine	7
3. Specification Models	8
3.1. State Charts	8
3.2. Petri Nets	8
3.3. Analyse-Methoden	10
4. Stochastische Ereignissysteme	10
4.1. Stochastik	10
4.2. Markov Prozesse	12
4.3. Stochastische Prozesse in kontinuierlicher Zeit	13
4.4. Markov-Ketten in kontinuierlicher Zeit	14
4.5. Warteschlangen	15
5. Worst-Case Event Systems – Online Algorithms	19
5.1. Skis mieten	19
5.2. Skis mieten randomisiert	19
5.3. Zwei mögliche z	20
5.4. Untere Schranken	20
5.5. The TCP Acknowledgement Problem	20
5.6. Das TCP Congestion Control Problem	21
5.7. Das dynamische Modell	21
6. Network Calculus	22
6.1. Network Calculus	22
6.2. Adversarial Queuing Theory	25

## 1. AUTOMATEN UND SPRACHEN

## 1.1. Alphabete und Wörter.

**Alphabet  $\Sigma$ :** Sammlung von Symbolen

zB:  $\Sigma = \{D, Q\}$

**String:** (word, Wort) ist eine Sequenz aus  $\Sigma$

zB.  $DD, QD, DDD, \dots$

- $\varepsilon$  ist der leere String
- $|\text{wort}|$  bedeutet die Länge eines Wortes

## 1.2. Finite Automaten.

## 1.2.1. Definition.

**Finite Automaten:** Ein FA ist ein 5-Tupel  $(Q, \Sigma, \delta, q_0, F)$  mit

$Q$ : endliche Anzahl Zustände

$\Sigma$ : ein Alphabet

$\delta: Q \times \Sigma \rightarrow Q$  eine Transferfunktion

$q_0 \in Q$ : Startzustand

$F \subset Q$ : akzeptierende Zustände

- Ein Automat akzeptiert einen String  $u$ , falls der mit  $q_0$  beginnende Pfad von  $u$  in einem akzeptierende Zustand endet.
- Die *akzeptierende Sprache*  $L(M)$  ist die Menge aller Wörter, die  $M$  akzeptiert.

**Deterministic FA (DFA):** Jeder Zustand hat für jedes Symbol genau einen Folgezustand.

**Nondeterministic FA (NFA):** Siehe Kapitel 1.5.

## 1.3. Sprachen.

**regulär:** eine Sprache ist *regulär* wenn es einen FA gibt, welcher sie akzeptiert.

**finite Sprachen:** haben eine endliche Anzahl Wörter.

*Alle finiten Sprachen sind regulär!* (Man kann immer einen Baum aufspannend, dessen Blätter akzeptierende Zustände sind.)

**infinite Sprachen:**

## 1.4. Reguläre Ausdrücke.

Operation	Symbol	Unix	Bedeutung
Union	$\cup$		OR
Concatenation	$\bullet$		Sequenz
Kleene-star	*	*	0- oder mehrmals
Kleene-plus	+	+	1- oder mehrmals
Exponentation	$(r)^n$	$(r)\{n\}$	n-mal
		^	Zeilenanfang
		\$	Zeilenende
		egrep -i 'pattern'	

Auswertungsreihenfolge: \*,  $\bullet$ ,  $\cup$

Reguläre Sprachen sind *geschlossen*: durch die Anwendung von regulären Ausdrücken entstehen wieder reguläre Sprachen.

*Union Example*  $L_1 \cup L_2$  (Unioner):

- (1) mache Tabelle  
Zeilen: Wörter (Zustände) der Sprache  $L_1$   
Spalten: Wörter (Zustände) der Sprache  $L_2$
- (2) jede Zelle stellt einen Wort / Zustand dar
- (3) markiere akzeptierende Zustände

- (4) markiere Startzustand  
 (5) zeichne alle Transferfunktionen ein

$$\begin{aligned}
 M_1 &= (Q_1, \Sigma, \delta_1, q_1, F_1) \\
 M_2 &= (Q_2, \Sigma, \delta_2, q_2, F_2) \\
 M_U &= (Q_1 \times Q_2, \Sigma, \delta_1 \times \delta_2, (q_1, q_2), F_U) && \text{Unioner} \\
 &F_U = Q_1 \times F_2 \cup F_1 \times Q_2 \\
 M_\cap &= (Q_1 \times Q_2, \Sigma, \delta_1 \times \delta_2, (q_1, q_2), F_\cap) && \text{Intersector} \\
 &F_\cap = F_1 \times F_2 \\
 M_- &= (Q_1 \times Q_2, \Sigma, \delta_1 \times \delta_2, (q_1, q_2), F_-) && \text{Difference (nur } M_1 \text{ akzeptiert)} \\
 &F_- = Q_1 \times F_2 - F_1 \times Q_2 \\
 M_\otimes &= (Q_1 \times Q_2, \Sigma, \delta_1 \times \delta_2, (q_1, q_2), F_\otimes) && \text{Symmetric Difference} \\
 &F_\otimes = F_U - F_\cap \\
 \bar{M} &= (Q_1 \times Q_2, \Sigma, \delta_1 \times \delta_2, (q_1, q_2), \bar{F}) && \text{Complement} \\
 \bar{F} &= \Sigma^* - F_U
 \end{aligned}$$

#### 1.4.1. Concatenation.

- $L \bullet \varepsilon = L$
- $L \bullet \emptyset = \emptyset$

#### 1.5. Nondeterministic FA. Unterschiede zum DFA

- Mehrere Zustände gleichzeitig aktiv:
  - durch  $\varepsilon$  Übergänge
  - mehrere Übergänge mit gleichem Symbol
- Impliziten "fail-state": Zustände müssen nicht für jedes Symbol eine Transferfunktion definieren.

NFAs sind auch geschlossen unter regulären Ausdrücken.  
 Jeder reguläre Ausdruck kann mit einem NFA realisiert werden.

#### 1.6. NFA $\rightarrow$ FA.

##### 1.6.1. Nondeterminisms.

Type	Machine Analog	$\delta$ -Funktion	Easy to fix?	formaly
Under-determined	Crash	No output	yes, fail-state	$ \delta(q, a)  = 0$
Over-determined	Random choice	Multi-valued	no	$ \delta(q, a)  > 0$
$\varepsilon$	Pause	Redefine $\Sigma$	no	$ \delta(q, a)  > 0$

##### 1.6.2. Anleitung.

- (1) Mache  $2^n$ -power-states.
- (2) Mache die Transferfunktionen zwischen den power-states.
- (3) Vereine alle Zustände, von denen nur  $\varepsilon$ -Kanten zueinander führen.
- (4) Kennzeichne den Start-Zustand.
- (5) Accepting-States: alle Zustände welche akzeptierende enthalten.

**REX  $\rightarrow$  NFA  $\rightarrow$  FA**

### 1.7. Generalized Nondeterministic Finite Automaton (GNFA).

#### Definition

- Kanten sind mit regulären Ausdrücken bezeichnet.
- Nur ein Start-Zustand, welcher nur eine zuführende Transferfunktion hat.
- Nur einen akzeptierenden Zustand ohne Ausgang.

#### 1.7.1. NFA $\rightarrow$ REX.

- (1) Mache den GNFA
  - (a) Vereine alle doppelten Pfeile zwischen zwei Zuständen.
  - (b) Mache einen einzelnen Start-Zustand.
  - (c) Mache einen einzelnen End-Zustand.
- (2) Loop: fasse so lange Pfeile und Zustände zusammen, bis nur noch zwei Zustände sind!
- (3) REX ist der Ausdruck auf der einzigen Kante.

### 1.8. Pumping Lemma.

Wenn  $L$  regulär ist, gilt für alle  $u \in L$  mit  $|u| \geq p$ :

$u = xyz$	$x$ : Präfix, $z$ : Suffix
$ y  \geq 1$	$y$ nicht leer
$absxy \leq p$	pumping in den ersten $p$ Buchstaben
$xy^iz \in L$	$y$ pumpbar

- es gilt:
  - "regulär"  $\Rightarrow$  "pumpbar"
  - "nicht pumpbar für alle Aufteilungen von  $x, y, z$ "  $\Rightarrow$  "nicht regulär"
- *Achtung*: aber nicht:
  - "pumpbar"  $\Rightarrow$  "regulär"

## 2. INTELLIGENTERE AUTOMATEN

### 2.1. Context Free Grammars (CFG).

*Definition*: Eine CFG besteht aus  $(V, \Sigma, R, S)$  mit

$V$ : endliche Anzahl Variablen

$\Sigma$ : ein Alphabet

$R$ : endliche Anzahl von Regeln: " $y$  produziert  $w$ "

$v \rightarrow w$  mit  $v \in V$  und  $w \in (\Sigma \cup V)^*$

$S \in V$ : Startsymbol

*Korrektheitsbeweise*: Beweise, dass  $L$  die Sprache  $L(G)$  von  $G$  ist.

- (1)  $L \subseteq L(G)$ : Jedes Wort von  $L$  kann mit  $G$  erzeugt werden.
- (2)  $L \supseteq L(G)$ :  $G$  generiert nur Strings aus  $L$ .

#### 2.1.1. Ableitungen und Sprachen.

**derivation symbol**  $\Rightarrow$ : "1-step produces", "Herleitungssymbol"

$x \Rightarrow y$  bedeutet, dass  $y$  in einem Schritt aus  $x$  gemacht werden kann.

**derivation symbol**  $\Rightarrow^*$ : "produces"

$x \Rightarrow^* y$  bedeutet, dass  $y$  aus  $x$  gemacht werden kann.

**Context Free Language**: Ist die von einer CFG  $G$  generierte Sprache  $L(G)$ .

$L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$

**left-most derivation**: Die Variable ganz links wird zuerst ersetzt.

**right-most derivation**: Die Variable ganz rechts wird ersetzt.

**derivation tree**: "Ableitungs-Baum"

- *Definition*: Die Wurzel ist das Startsymbol. Knoten sind Variablen. Blätter sind Symbole.
- Ableitungen von Strings sind nicht eindeutig.

- Ableitungsbäume stellen Ableitungen eindeutig dar.

**Ambiguity:** "Mehrdeutigkeit":

- Wort ist mehrdeutig bezüglich einer Grammatik, wenn mehrere Ableitungsbäume existieren.
- Grammatik mit mehrdeutigem Wort ist mehrdeutig.

### 2.1.2. Designing CFG.

Einige Möglichkeiten:

- kreativer Prozess
- Aufteilen in einfachere Grammatiken und vereinen der Startzustände ( $S \rightarrow S_1 | S_2$ )
- wenn CFG regulär:
  - (1) Mache den FA.
  - (2) Definiere eine Variable oder Produktion für jeden Zustand.
  - (3) Wenn ein Pfeil mit  $a$  vom Zustand  $x$  nach  $y$  zeigt, mache die Regel  $x \rightarrow ay$ .
  - (4) Wenn  $x$  akzeptierend ist, mache  $x \rightarrow \varepsilon$ .
  - (5) Der Startzustand des CFGs und des FAs sind gleich.
- viele weitere Möglichkeiten ...

## 2.2. Push-Down Automata (PDA).

### 2.2.1. Probleme von FAs.

- Automat für CFGs.
- können nicht zählen, wir brauchen unbegrenzten Speicher.

### 2.2.2. Definition. Ein Push-Down Automaton ist ein 6-Tupel

$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$  mit

$Q, \Sigma, q_0, F$ : wie beim FA

$\Gamma$ : Tape Alphabet

$\delta(p, x, y) = (q, z)$ : gibt alle Folgezustände und Push-Operationen.

$\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow P(Q \times \Gamma_\varepsilon)$

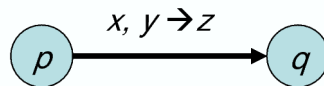


ABBILDUNG 1. wechsle nach  $q$ , wenn ein  $x$  gelesen wird und  $y$  auf dem Stack liegt. Ersetze  $y$  durch  $z$ .

### 2.2.3. PDA's à la Sipser.

- Notation
  - $x, y \rightarrow z$ : input  $x$ , pop  $y$ , push  $z$
  - $x = \varepsilon$ : ignoriere Input, lese nichts
  - $y = \varepsilon$ : lese nichts vom Stack, push  $z$
  - $z = \varepsilon$ : kein push
- Pushe zuerst  $\$$  um leeren Stack zu erkennen.

### 2.3. Right Linear Grammars.

*Definition:* CFG, deren Produktionen die Form  $A \rightarrow uB$  oder  $A \rightarrow u$  haben. ( $A, B$ : Variablen,  $u \in \Sigma$ )

*Theorem:* Zu jedem NFA ( $M = (Q, \Sigma, \delta, q_0, F)$ ) gibt es eine Right Linear Grammar ( $G = (V, \Sigma, R, S)$ ) mit der gleichen Sprache.

*Beweis:*

- Die Variablen sind die Zustände:  $V = Q$
- Startsymbol ist der Startzustand:  $S = q_0$
- das Alphabet ist gleich
- der Transfer  $q_1 \rightarrow a \rightarrow q_2$  wird die Produktion  $q_1 \rightarrow aq_2$
- akzeptierende Zustände ( $q \in F$ ) werden  $\varepsilon$ -Produktionen:  $q \rightarrow \varepsilon$

*Achtung:* Nicht jede CFG kann in eine Right Linear Grammar umgewandelt werden!

### 2.4. Chomsky Normal Form (CNF).

*Definition:* Eine CFG heisst CNF, falls jede Produktion folgende Form hat:

- $S \rightarrow \varepsilon$       $\varepsilon$  nur rechts vom Startsymbol,  $S$  nur auf der linken Seite
- $A \rightarrow BC$      genau zwei Variablen
- $A \rightarrow a$      nur einzelne Terminalsymbole

#### 2.4.1. CFG $\rightarrow$ CNF.

- (1) Entferne  $S$  von der linken Seite  
( $'S \rightarrow \varepsilon|a|Sa' \Rightarrow 'S \rightarrow \varepsilon|A, A \rightarrow a|Aa'$ )
- (2) Entferne alle  $\varepsilon$ -Produktionen, ausser der Startvariable.
- (3) Fasse  $A$  und  $B$  zusammen, wenn  $A \rightarrow B$
- (4) Teile Produktionen mit mehr als drei Terminalsymbolen auf  
( $'A \rightarrow bCd' \Rightarrow 'A \rightarrow BE, E \rightarrow CD'$ )
- (5) Produziere Terminalsymbole einzeln:  
( $'A \rightarrow aB' \Rightarrow 'A \rightarrow AB, A \rightarrow a'$ )

#### 2.4.2. CFG $\rightarrow$ GPDA.

*Definition GPDA:* Ein Generalized PDA ermöglicht das Pushen eines Strings als oberstes Stacksymbol.

- (1) Pushe die Markierung  $\$$  und das Startsymbol  $S$  auf den Stack.
- (2) Repeat ewig:  
nächstes Symbol auf dem Stack ist ein:
  - (a) Variable  $A$ : wähle eine Produktion und ersetze  $A$  durch rechte Seite.  
Nichtdeterministisch, da mehrere Regeln möglich!!
  - (b) Terminalsymbol  $b$ : Wenn das nächste Symbol ein  $b$  ist mache weiter, sonst verwerfe Ausführungszweig.
  - (c)  $\$$ : gehe in akzeptierenden Status.

#### 2.4.3. PDA $\rightarrow$ CFG.

*Theorem:* Für jeden PDA gibt es eine CFG mit der gleichen Sprache.

#### 2.4.4. Context Sensitive Grammars.

Allgemeine Grammatik.

Beispiel, mit  $\Sigma = \{a, b, c\}$ :

$$\begin{array}{ll} S \rightarrow \varepsilon | ASBC & bB \rightarrow bb \\ A \rightarrow a & bC \rightarrow bc \\ CB \rightarrow BC & cC \rightarrow cc \\ aB \rightarrow ab & \end{array}$$

Auf beiden Seiten sind beliebige Variablen/Terminal-Strings erlaubt.

Aus technischen Gründen muss die linke Seite kürzer als die rechte sein.

#### 2.4.5. Tandem Pumping.

Für die Sprache  $L$  eines CFGs gilt:  $\forall w \in L$  mit  $|w| > p$

$$w = uvxyz$$

$$|vy| \geq 1 \quad \text{pumpable Regionen sind nicht leer}$$

$$|vxy| \leq p$$

$$uv^i xy^i z \in L \text{ für alle } i \quad \text{tandem-pump } v, y$$

Alle Wörter pumpbar  $\Rightarrow$  CFG

Ein Wort nicht pumpbar für alle möglichen Aufteilungen  $\Rightarrow$  keine CFG

#### 2.5. Finite State Transducer.

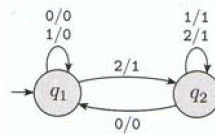


ABBILDUNG 2. Transducer

*Definition:* Ein FA, dessen Output ein Wort ist.

Kantenbeschriftung: Input / Output

#### 2.6. Turing Machine.

*Definition:* Eine Maschine mit

- endlichem read-only Speicher
- unendliches read-write Band
- Input vom Band

*Formale Definition:* Eine Turing Maschine ist ein 7-Tupel  $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$

$Q, \Sigma, q_0$ : wie beim FA

$q_{acc}, q_{rej}$ : akzeptierende, verwerfende Zustände

$\Gamma$ : Alphabet des Bandes, mindestens  $\square$  und  $\Sigma$

$\delta$ :  $\delta : (Q - q_{acc}, q_{rej}) \times \Gamma \rightarrow Q \times \Gamma \times L, R$

Also bedeutet  $\delta(p, x) = (q, y, D)$  mit einem Nicht-Halt-Zustand  $p$  und einem Band-Symbol  $x$ : TM geht in den Zustand  $q$ , ersetzt  $x$  durch  $y$  und geht auf dem Band nach  $D$  (links oder rechts)

*Eigenschaften der TM:*

- (1) TM ist so mächtig wie jeder "richtige" Computer.
- (2) TM ist simpel genug, um interessante computertheoretische Resultate zu beweisen.

### 2.6.1. Komplexitätsklassen P und NP.

**P:** Problem kann in polinomialer Zeit gelöst werden.

**NP:** Entscheidungsprobleme können von einem nicht-deterministischer TM gelöst werden. TM akzeptiert, wenn ein Pfad akzeptiert.

- Eine TM kann Resultat in polinomialer Zeit prüfen.

**NP-complete:** Subset der schwierigsten Probleme in NP, wenn eines gelöst werden kann, können alle gelöst werden.

## 3. SPECIFICATION MODELS

### 3.1. State Charts.

3.1.1. *Hierarchie.* Zustände können zu Superzuständen zusammengefasst werden.

**OR super-state:** Nur ein Subzustand ist aktiv, wenn Superzustand aktiv ist.

**AND super-state:** Alle Subzustände sind aktiv, wenn Superzustand aktiv ist.

**ancestor states:** die Eltern-Zustände eines Subzustandes.

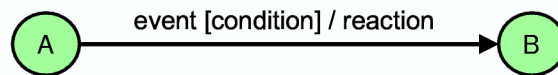


ABBILDUNG 3. Kantenbeschriftung

3.1.2. *Kantenbeschriftung.*

**Event:** Kante wird bei diesem Event durchlaufen, Events sind global sichtbar

$e1$  and  $e2$

$e1$  or  $e2$

not  $e1, \overline{e1}$

**Condition:** Bedingung, um Kante zu durchlaufen.

$a = b$

**Reaction / Action:** Zuweisung von neuen Werten

$a = 10$

$(a = b = 0; c = -1)$

3.1.3. *Simulations-Prozess.*

*Simulations-Schritt:*

- (1) Mögliche Zustandswechsel suchen.
- (2) Alle Aktionen rechnen, zwischenspeichern.
- (3) Werte der Variablen aktualisieren.

3.1.4. *Vor- und Nachteile.*

**Vorteile:**

- einfach in Hardware und Software implementierbar
- effiziente Simulation

**Nachteile:**

- schwierig für grosse Systeme
- keine formale Repräsentation
- Systemzustand in Variablen versteckt

3.2. **Petri Nets.**

3.2.1. *Definition.*

Ein Petri Net  $(S, T, F, M_0)$  ist ein gerichteter, bipartiter Graph mit

- S: Set von Plätzen  $p_i$
- T: Set von Transitionen  $t_i$
- F: Set von Kanten  $f_i$
- $M_0: S \rightarrow \mathbb{N}$  Startbesetzung mit Tokens
- W: Gewichtung der Kanten  $f_i$
- K: Kapazität der Plätzen  $p_i$

3.2.2. *Token-Besetzung.*

- Jeder Platz  $p_i$  hat eine Anzahl Tokens.
- $M(p_i)$  bezeichnet die Besetzung eines Platzes.
- Verteilung der Tokens ist der Zustand des Petri Netzes.
- Die Dynamik des Netzes heist *token game*.

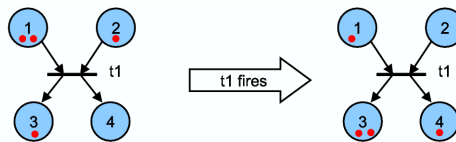


ABBILDUNG 4. Transition von  $t_1$

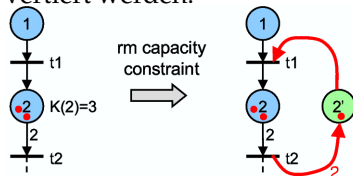
3.2.3. *Transitionen.*

- Eine Transition ist *aktiviert*, wenn alle Input-Plätze  $p_i$  mindestens ein Token enthalten.  
Oder mit Gewichtung  $W$ , mindestens  $W(f_i)$  Tokens.
- Nur eine *einzig*e Transition feuert gleichzeitig.
- Keine Reihenfolge festgelegt!
- Wenn eine Transition *feuert*:
  - (1) Löscht ein Token von allen Input-Plätzen  $p_i$  (oder  $W(f_i)$  Tokens).
  - (2) Kreiert ein Token in allen Output-Plätzen  $p_o$  (oder  $W(f_i)$  Tokens).

Dynamik des Petri Netzes ist *nicht deterministisch!*

3.2.4. *Petri Net mit endlicher Kapazität.*

- Platz  $p_i$  kann nur  $K(p_i)$  Tokens haben.
- $t$  kann nur feuern, wenn alle Outputs noch genügend Kapazität haben.
- Petri Netze mit endlicher Kapazität können in solche mit unendlicher konvertiert werden.



- Füge für jeden Platz  $p$  einen komplementären Platz  $p'$  mit  $M_0(p') = K(p) - M_0(p)$  ein.
- Für jede ausgehende Kante  $e = (p, t)$  füge ein  $e' = (t, p')$  mit Gewicht  $W(e)$  ein.
- Für jede zuführende Kante  $e = (t, p)$  füge ein  $e' = (p', t)$  mit Gewicht  $W(e)$  ein.

### 3.2.5. Sprache von Petri Netzen.

- Jede Transition wird mit einem Symbol bezeichnet.
- Die Sequenz der feuern den Transitionen generiert ein Wort.

Jede reguläre Sprache ist die Sprache eines Petri Netzes.

### 3.2.6. Verhaltenseigenschaften.

**Reachability:**  $M_n$  ist erreichbar, wenn eine Sequenz von feuern den Transitionen existiert, so dass  $M_n = M_0 \cdot t_1 \cdot t_2 \dots t_n$

**K-Boundedness:** Ein Petri-Netz ist k-bounded, wenn die Anzahl Tokens in jedem Platz höchstens  $k$  ist.

**Safety:** 1-Boundedness

**Liveness:** Wenn wir  $M_n$  von  $M_0$  erreicht haben, können wir eine Transition feuern?

Transition  $t$  des Petri-Netzes  $(N, M_0)$

**dead:** wenn  $t$  nie in  $L(M_0)$  gefeuert werden kann.

**L1-live:** wenn  $t$  mindestens einmal gefeuert werden kann.

**L2-live:**  $\exists k \in \mathbb{N}^+$ , so dass  $t$  in  $L(M_0)$  mindestens  $k$  mal gefeuert werden kann.

**L3-live:**  $t$  kommt in einer Sequenz von  $L(M_0)$  unendlich mal vor.

**L4-live:**  $t$  ist L1-live für alle erreichbaren Zustände von  $M_0$ .

L4-Liveness  $\Rightarrow$  L3-Liveness  $\Rightarrow$  L2-Liveness  $\Rightarrow$  L1-Liveness

## 3.3. Analyse-Methoden.

### 3.3.1. Erreichbarkeits Baum. Erzeugung:

- (1) Initialzustand, tag: "new"  
Bsp:  $(1, 0, 0, 1)$  "neu"
- (2) Wiederhole für jeden Zustand  $M$ 
  - (a) wenn  $M$  alt, tag: "old", break
  - (b) für jede mögliche Transaktion  $t$ 
    - (i)  $M' = M \times t$
    - (ii) Wenn  $t$  unendlich viel gefeuert werden kann, schreibe  $\omega: (1, \omega, 0)$ .
    - (iii) Füge  $M'$  als neuen Knoten, mit Kante  $t$  ein.

Resultate:

**bounded:**  $\omega$  kommt nirgends vor.

**safe:** Nur 0 und 1 kommen vor.

**dead:**  $t$  kommt nicht vor.

**reachable:**  $M$  kommt vor.

### 3.3.2. Inzidenz-Matrix $A$ . Beschreibt den Token-Fluss der verschiedenen Transaktionen.

$A_{ij} = A_{\text{Zeile}} \text{ Spalte}$  Tokengewinn im Platz  $i$ , wenn Transition  $j$  feuert.

$M_k = M_0 + A \times t_1 \times t_2 \times \dots \times t_k$

$$\text{Bsp: } \begin{pmatrix} 3 \\ 0 \\ 0 \\ 2 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} -2 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & -2 & 2 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

### 3.3.3. Vereinfachungsregeln. Vereinfachung des Petri-Netzes unter Erhaltung von: Liveness, Safeness, Boundedness.

## 4. STOCHASTISCHE EREINGISSYSTEME

### 4.1. Stochastik.

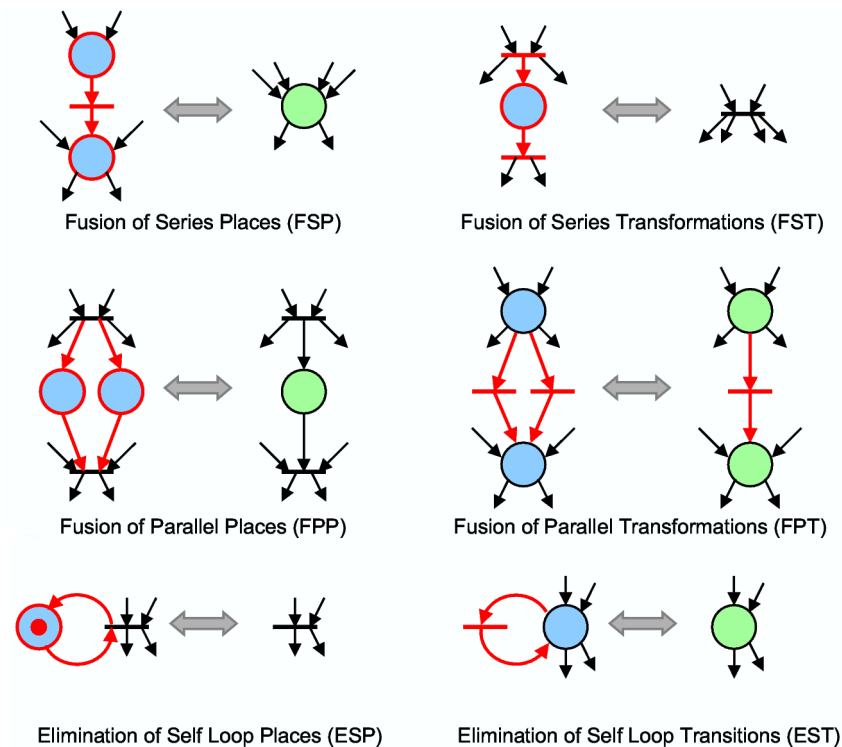


ABBILDUNG 5. Vereinfachungsregeln

## 4.1.1. Grundbegriffe.

$\Omega$ : Menge aller Elementarereignisse

$P[\omega]$ : Whkeit des Elementarereignisses  $\omega$

**Ereignis**: Teilmenge von  $\Omega$

**unabhängig**:  $P[A \cap B] = P[A] \cdot P[B]$

- $\Rightarrow E[X \cdot Y] = E[X] \cdot E[Y]$

- $\Rightarrow Var[X + Y] = Var[X] + Var[Y]$

**bedingte Whkeit**:  $P[A|B] = \frac{P[A \cap B]}{P[B]}$

**totale Whkeit**:  $P[B] = \sum P[B|A_i]P[A_i]$       wenn  $A_i$  paarweise diskunkt

**Zufallsvariable**: Abbildung  $X : \Omega \rightarrow \mathbb{R}$

**Dichtefunktion**:  $f_X : \mathbb{R} \rightarrow [0, 1]$       mit  $f_X(x) = P[X = x]$

**Verteilungsfunktion**:  $F_X : \mathbb{R} \rightarrow [0, 1]$       mit  $F_X(x) = P[X \leq x] = \int_0^x f_X(r) dr$

**Erwartungswert**:  $E[X] = \int x f_X(x) dx$

- $E[a \cdot X + b] = a \cdot E[X] + b$

- *Linearität des Erwartungswertes*

$$E[X] = a_1 E[X_1] + a_2 E[X_2] + \dots + a_n E[X_n] \quad \text{mit } X = a_1 X_1 + a_2 X_2 + \dots + a_n X_n$$

müssen nicht unabhängig sein!

**Varianz**:  $Var[X] = E[(X - E[X])^2] = E[X^2] - E[X]^2$

- $Var[a \cdot X + b] = a^2 \cdot Var[X]$

**Standardabweichung**:  $\sigma(X) = \sqrt{Var[X]}$

## 4.1.2. Verteilungen.

**Bernulli:** Erfolgswahrscheinlichkeit  $p$

$$P[X = 1] = p \quad P[X = 0] = 1 - p$$

$$E[X] = p \quad \text{Var}[X] = p(1 - p)$$

**Binomial:** Parameter  $n, p$

$$P[X = k] = \binom{n}{k} p^k (1 - p)^{n-k}$$

$$E[X] = np \quad \text{Var}[X] = np(1 - p)$$

**Poisson:** Parameter  $\lambda$

$$P[X = k] = \frac{e^{-\lambda} \lambda^k}{k!}$$

$$E[X] = \lambda \quad \text{Var}[X] = \lambda$$

**Geometrische:** Parameter  $p$

$$P[X = k] = (1 - p)^{k-1} p$$

$$E[X] = \frac{1}{p} \quad \text{Var}[X] = \frac{1-p}{p^2}$$

#### 4.2. Markov Prozesse.

Zeitdiskrete Folge von Zufallsexperimenten, welche nur vom aktuellen Zustand abhängen.

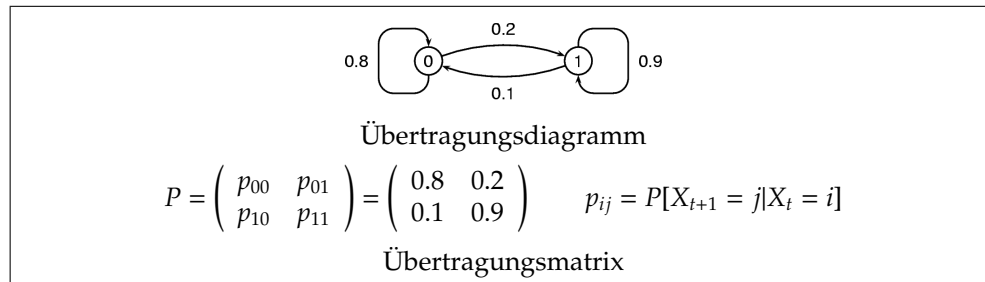


ABBILDUNG 6. Markov-Prozesse

*Eigenschaften:*

**unendlich:** wenn unendliche fortsetzbar

**(zeit)homogen:** wenn Matrix  $P$  nicht zeitabhängig ist.

$$q_{t+1} = q_t \cdot P = q_0 \cdot P^{t+1}$$

##### 4.2.1. Übergangszeit.

**Übergangszeit:** Zufallsvariable: minimale Anzahl Schritte von  $i$  nach  $j$

$$T_{ij} = \min\{n \geq 1 | X_n = j, \text{ wenn } X_0 = i\}$$

**erwartete Übergangszeit:**

$$h_{ij} = E[T_{ij}] = 1 + \sum_{k:k \neq j} p_{ik} h_{kj} \text{ (direkter Weg plus Wege von allen Nachbarn)}$$

**Ankunftswahrscheinlichkeit:** von  $i$  nach  $j$

$$f_{ij} = P[T_{ij} < \infty] = p_{ij} + \sum_{k:k \neq j} p_{ik} f_{kj} \text{ (direkte Whkeit plus Whkeit zum Nachbarn plus Ankunftswahkeit von dort ans Ziel)}$$

##### 4.2.2. Stationäre Analyse. Verhalten für $t \rightarrow \infty$

**stationäre Verteilung  $\pi$ :** Zustandsvektor mit  $\pi_i \geq 0$  und  $\sum \pi_i = 1$

$$\pi = \pi \cdot P \quad \text{Eigenvektor von } P \text{ zum EW } 1$$

muss nicht eindeutig sein

**irreduzible Markov-Kette:** Man kann von jedem Zustand in jeden anderen kommen.

- Für alle Zustände  $i, j$  gibt es ein  $n$  mit  $p_{ij}^{(n)} > 0$ .

$p_{ij}^{(n)}$  ist Whkeit in  $n$  Schritten von  $i$  nach  $j$  zu kommen.

- **Satz:** irreduzibel  $\Rightarrow$  eindeutige stationäre Verteilung mit  $\pi_i = 1/h_{ii}$
- Irreduzible endliche Markov-Kette muss *nicht* gegen stationäre Verteilung konvergieren.

**Periode:** eines Zustandes  $j$  ist die grösste Zahl  $\xi$ :

$$\{n \in \mathbb{N} \mid p_{jj}^{(n)} > 0\} \subseteq \{i \cdot \xi\} \quad \text{mit } n, i \in \mathbb{N}$$

$\xi$  ist die kleinste obere Schranke für die Anzahl Schritte, um vom Zustand zu sich selber zu gelangen.

**aperiodischer Zustand:** hat  $\xi = 1$

Testbedingungen:

- $p_{jj} > 0$
- $\exists m, n \in \mathbb{N}$  mit  $p_{jj}^{(n)}, p_{jj}^{(m)} > 0$  und  $\text{ggT}(m, n) = 1$

**aperiodische Markov-Kette:** Hat nur aperiodische Zustände.

**ergodisch:** irreduzibel, aperiodisch

$$\lim_{t \rightarrow \infty} q_t = \pi \quad \text{für jeden Startzustand}$$

Ergodische Markov-Ketten besitzen eine eindeutige stationäre Verteilung.

4.2.3. *Vektor-Ketten.* Von Prozessen, welche nicht nur von  $X_{t-1}$  abhängen, sondern auch von  $X_{t-2}, X_{t-3}, \dots$

Beispiel:  $X_t = X_{t-1} - X_{t-2} \Rightarrow$

$$Z_t = \begin{pmatrix} X_t \\ X_{t-1} \end{pmatrix} = \begin{pmatrix} 1 & -1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} X_{t-1} \\ X_{t-2} \end{pmatrix} = \begin{pmatrix} 1 & -1 \\ 1 & 0 \end{pmatrix} \cdot Z_{t-1}$$

#### 4.3. Stochastische Prozesse in kontinuierlicher Zeit.

4.3.1. *Kontinuierliche Zufallsvariablen.*

**Zufallsvariable:**  $X : \Omega = \mathbb{R} \rightarrow \mathbb{R}$

**Dichte:**  $f_X(x) : \mathbb{R} \rightarrow \mathbb{R}_0^+$

$$\int_{-\infty}^{\infty} f_X(x) dx = 1$$

**Verteilung:**

$$F_X(x) : P[X \leq x] = \int_{-\infty}^x f_X(x) dx$$

$$P[a \leq x \leq b] = F_X(a) - F_X(b)$$

**Erwartungswert:**

$$\mathbb{E}(X) = \int_{-\infty}^{\infty} t \cdot f_X(t) dt$$

**Varianz:**

$$\text{Var}[X] = \mathbb{E}[(X - \mathbb{E}[X])^2] = \int_{-\infty}^{\infty} (t - \mathbb{E}[X])^2 \cdot f_X(t) dt$$

**unabhängig:**

$$P[X \leq x, Y \leq y] = P[X \leq x] \cdot P[Y \leq y]$$

4.3.2. *Kontinuierliche Verteilungen.*

**Gleichverteilung:** auf  $[a, b]$  ist alles gleich wahrscheinlich

$$f_X = \begin{cases} \frac{1}{a+b} & , \text{ falls } a \leq x \leq b \\ 0 & , \text{ sonst} \end{cases}$$

$$\mathbb{E}[X] = \frac{a+b}{2}$$

$$\text{Var}[X] = \frac{(b-a)^2}{12}$$

**Normalverteilung:** mit Parameter  $\mu, \sigma$

$$\begin{aligned} f_X &= \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \\ \mathbb{E}[X] &= \mu \\ \text{Var}[X] &= \sigma^2 \end{aligned}$$

4.3.3. *Exponentialverteilung.* mit Parameter  $\lambda > 0$

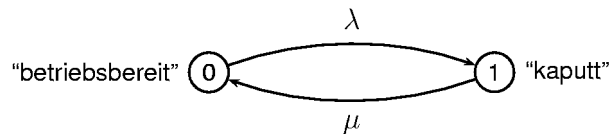
$$\begin{aligned} f_X &= \begin{cases} \lambda \cdot e^{-\lambda x} & , \text{ falls } x \geq 0 \\ 0 & , \text{ sonst} \end{cases} \\ \mathbb{E}[X] &= \frac{1}{\lambda} \\ \text{Var}[X] &= \frac{1}{\lambda^2} \\ F_X(x) &= \begin{cases} 1 - e^{-\lambda x} & , \text{ falls } x \geq 0 \\ 0 & , \text{ sonst} \end{cases} \end{aligned}$$

**Gedächtnislosigkeit:**

$$P[X > x + y \mid X > y] = P[X > x]$$

**Skalierung:** Falls  $X$  exponentialverteilt mit Parameter  $\lambda$  ist, ist  $Y := aX$  exponentialverteilt mit Parameter  $\lambda/a$ .

**Warteproblem:** Falls  $X_1, \dots, X_n$  exponentialverteilt mit Parameter  $\lambda_1, \dots, \lambda_n$ , dann ist  $X = \min(X_1, \dots, X_n)$  exponentialverteilt mit Parameter  $\lambda = \lambda_1 + \dots + \lambda_n$ .



- Aufenthaltswahrscheinlichkeit in 0 exponentialverteilt mit  $\lambda$
- Aufenthaltswahrscheinlichkeit in 1 exponentialverteilt mit  $\mu$

ABBILDUNG 7. Markov-Kette in kontinuierlicher Zeit

#### 4.4. Markov-Ketten in kontinuierlicher Zeit.

**Zustandsmenge:**  $S = \{0, \dots, n-1\}$

**Folge von Zustandsvariablen:**  $X(t)$  mit Wertemenge  $S$

**Startverteilung:**  $q(0) = (q_0(0), \dots, q_{n-1}(0))$  mit

$$q_i(0) > 0 \text{ und} \\ \sum q_i(0) = 1$$

**Markov-Bedingung:** "Markov-Kette ist gedächtnislos":

für alle  $k \in \mathbb{N}$  und beliebige

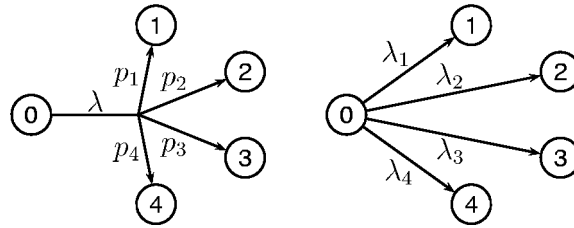
$0 \leq t_0 < t_1 < \dots < t_k < t$  und

$s, s_1, \dots, s_k \in S$  gilt:

$$P[X(t) = s \mid X(t_k) = s_k, X(t_{k-1}) = s_{k-1}, \dots, X(t_0) = s_0] = P[X(t) = s \mid X(t_k) = s_k]$$

**zeithomogen:** "Markov-Kette hängt nicht von der Zeit ab, es ist egal, wie lange der Prozess schon läuft.":

$$P[X(t+u) = j \mid X(t) = i] = P[X(u) = j \mid X(0) = i]$$



$$\lambda_i = \lambda \cdot p_i$$

ABBILDUNG 8. Mehrere Nachfolger

4.4.1. Zustände mit mehreren Nachfolgern. Siehe Abbildung 8.

- Jeder Zustand hat exponentialverteilte Aufenthaltsdauer  $v_i$ .
- Wenn Zustand  $i \in S$  verlassen wird, wird mit Whkeit  $p_{i,j}$  der Zustand  $j \in S$  angenommen.

$$p_{i,i} = 0$$

$$\sum_{j \in S} p_{i,j} = 1$$

- es gilt für alle  $i \in S$ :

$$\sum_{j \in S} v_{i,j} = v_i$$

4.4.2. Aufenthaltswahrscheinlichkeiten.

$$q_i(0) : = P[X(0) = i] \text{ für } i \in S \quad \text{Startverteilung } q(0)$$

$$q_i(t) : = P[X(t) = i] \text{ für } i \in S \quad \text{Verteilung zur Zeit } t$$

$$\underbrace{\frac{d}{dt} q_i(t)}_{\text{Änderung}} = \underbrace{\sum_{j:j \neq i} q_j(t) \cdot v_{i,j}}_{\text{Zufluss}} - \underbrace{q_i(t) \cdot v_i}_{\text{Abfluss}}$$

**Übergangsrate:** von  $i$  nach  $j$  ist:  $v_{i,j} = v_i \cdot p_{i,j}$

**stationäre Verteilung:** für  $t \rightarrow \infty$

$$0 = \sum_{j:j \neq i} \pi_j \cdot v_{i,j} - \pi_i \cdot v_i \quad \text{für alle } i \in S$$

**erreichbar:**  $j$  ist von  $i$  erreichbar, wenn es ein  $t > 0$  gibt mit

$$P[X(t) = j \mid X(0) = i] > 0$$

**irreduzibel:** Jeder Zustand ist von jedem anderen erreichbar.

*Satz:* für irreduzible Markov-Ketten existieren die Grenzwerte für alle  $i \in S$ .

$$\pi_i = \lim_{t \rightarrow \infty} q_i(t)$$

zusätzliche Bedingung:  $\sum \pi_i = 1$

#### 4.5. Warteschlangen.

4.5.1. Kendall-Notation  $X/Y/m/..$

**X:** Verteilung der Zwischenankunftszeiten.  
Zeiten zwischen zwei ankommenden Jobs.

**Y:** Verteilung der Bearbeitungszeiten  
der Jobs auf den Servern.

**m:** Anzahl Server

**Verteilungen:** von X und Y

**D:** deterministic: feste Dauer  
**M:** memoryless: exponentialverteilt  
**G:** general: beliebige Verteilung

#### 4.5.2. Der Poisson-Prozess.

$$\begin{aligned} P[\alpha(t + \tau) - \alpha(t) = n] &= e^{-\lambda\tau} \cdot \frac{(\lambda\tau)^n}{n!} && \text{für } n = 0, 1, \dots \\ \mathbb{E}[\alpha(t + \tau) - \alpha(t)] &= \lambda \cdot \tau \end{aligned}$$

- Ankunftsprozess mit exponentialverteilten Zwischenankunftszeiten ist ein Poisson-Prozess.
- Modelliert viele unabhängige, gleiche Nutzer.

#### 4.5.3. M/M/1 Warteschlangen.

**Ankunftsrate:** exponentialverteilt mit Parameter  $\lambda$

**Bedienrate:** exponentialverteilt mit Parameter  $\mu$

**Verkehrsdichte:**  $\rho := \frac{\lambda}{\mu}$

**stationäre Verteilung:** in der Warteschlange

$$\pi_k = \rho^k \cdot \pi_0$$

- falls  $\rho \geq 1 \implies$  Warteschlange wächst ins Unendliche.
- falls  $\rho < 1 \implies \pi_0 = 1 - \rho$ , System konvergiert.

Anzahl Jobs im System (Warteschlange + Server)

$$\begin{aligned} N &= \frac{\lambda}{\mu - \lambda} \\ \text{Var} &= \frac{\rho}{(1 - \rho)^2} \end{aligned}$$

**Auslastung:** des Servers:  $1 - \pi_0 = \rho$

#### 4.5.4. Little's Law.

$N(t)$ : Anahl Jobs in System zur Zeit  $t$ . (Warteschlange + Server)

$\alpha(t)$ : Anzahl angekommene Jobs in  $[0, t]$ .

$\beta(t)$ : Anzahl der beendeten Jobs in  $[0, t]$ .

$T_i$ : Antwortzeit des  $i$ -ten Jobs (Wartezeit + Bearbeitungszeit)

**Durchschnittswerte:** bis  $t$

$$\begin{aligned} N_t &= \frac{1}{t} \int_0^t N(\tau) d\tau \\ \lambda_t &= \frac{\alpha(t)}{t} \\ T_t &= \frac{\sum_{i=1}^{\alpha(t)} T_i}{\alpha(t)} \end{aligned}$$

**Grenzwerte:**

$$N = \lim_{t \rightarrow \infty} N_t, \quad \lambda = \lim_{t \rightarrow \infty} \lambda_t = \lim_{t \rightarrow \infty} \frac{\beta(t)}{t}, \quad T = \lim_{t \rightarrow \infty} T_t$$

**Formel von Little:** falls alle Grenzwerte existieren, gilt:

“Anzahl Jobs im System ist gleich Ankunftsrate mal Antwortzeit”

$$N = \lambda \cdot T$$

Diese Formel gilt auch für andere Server-Strategien als FCFS.

**stochastische Variante:** der Formel von Little

$$\bar{N} = \lambda \cdot \bar{T}$$

$$\text{mit } \bar{N} = \mathbb{E}[N], \quad \lambda = \lim_{t \rightarrow \infty} \frac{\mathbb{E}[\alpha(t)]}{t}, \quad \bar{T} = \mathbb{E}[T_i]$$

4.5.5. *Little's Law und Warteschlangen.* gilt im Gleichgewichtszustand

$$\begin{aligned}
 N &= \frac{\rho^2}{1-\rho} \\
 T &= \frac{1}{\lambda} N = \frac{\rho}{\lambda(1-\rho)} = \frac{1}{\mu-\lambda} && \text{Antwortzeit} \\
 W &= T - \frac{1}{\mu} = \frac{\rho}{\mu(1-\rho)} = \frac{\rho}{\mu-\lambda} && \text{Wartezeit} \\
 N_Q &= \lambda W = \frac{\rho^2}{1-\rho} && \text{mittlere Anzahl Jobs in Warteschlange}
 \end{aligned}$$

4.5.6. *Anwendungen der Formel von Little.*

- *Ein geschlossenes Warteschlangensystem*  
**K:** Anz Server, sind immer beschäftigt  
**N:** Anz Jobs, konstant  
 $\bar{X}$ : mittlere Bearbeitungszeit eines Jobs  
 $\bar{T}$ : mittlere Antwortzeit

$$T = \frac{N}{\lambda} = \frac{N\bar{X}}{K}$$

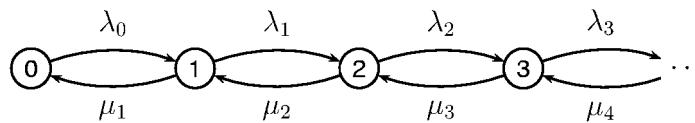
- *Ein geschlossenes Wartesystem mit N Plätzen.*  
 $\lambda$ : Rate neuer Jobs, Jobs werden abgewiesen, wenn schon N im System sind.  
 $\bar{K}$ : mittlere Zahl aktiver Server  
 $\beta$ : Anteil abgewiesener Jobs  
**Formel von Little:**

$$\bar{K} = (1 - \beta)\lambda\bar{X}$$

$$\beta = 1 - \frac{\bar{K}}{\lambda\bar{X}} \geq 1 - \frac{K}{\lambda\bar{X}}$$

4.5.7. *Durchsatzanalyse für Time-Sharing.*

- N:** Anzahl Benutzer des gleichen Servers
- R:** Mittlere Zeit zwischen zwei Anfragen des gleichen Benutzers.
- P:** Mittlere Rechenzeit des Servers.



Zustände: von 0 (alle tot) bis  $\infty$  unendlich viele

ABBILDUNG 9. Birth-and-Death Prozess

4.5.8. *Birth-and-Death Prozesse.*

$$\pi_k = \pi_0 \prod_{i=0}^{k-1} \frac{\lambda_i}{\mu_{i+1}} \quad \text{für } k \geq 1$$

$$\sum \pi_i = 1$$

$$\pi_0 = \frac{1}{1 + \sum_{k \geq 1} \prod_{i=0}^{k-1} \frac{\lambda_i}{\mu_{i+1}}}$$

- *Beschränkter Warteraum:* es werden maximal  $n$  Jobs angenommen  
 $\pi_k = \rho^k \pi_0$  für  $1 \leq k \leq n$

$$\pi_0 = \begin{cases} \frac{1}{n+1} & \text{für } \rho = 1 \\ \frac{1-\rho}{1-\rho^{n+1}} & \text{sonst} \end{cases}$$

- *Beschränkte Benutzerzahl:*  $M$  Terminals und ein Server  
 $\pi_k = \pi_0 \cdot \prod_{i=0}^{k-1} \frac{\lambda(M-i)}{\mu}$  für  $1 \leq k \leq M$   $\pi_0 = \frac{1}{\sum_{k=0}^M (\lambda/\mu)^k \cdot M_k}$   
 mit  $M_k = M(M-1)(M-2) \dots (M-k+1)$

4.5.9. Das M/M/m System. System mit  $m$  Servern.

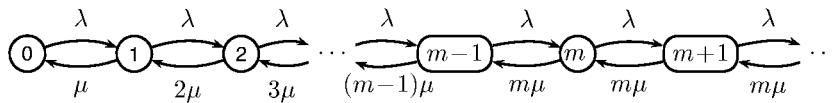


ABBILDUNG 10. Das M/M/m System

$$\begin{aligned} \rho &:= \frac{\lambda}{m\mu} < 1 \\ \pi_k &= \begin{cases} \pi_0 \cdot \frac{\lambda^k}{\mu^k \cdot k!} = \pi_0 \cdot \frac{(\rho m)^k}{k!} & \text{für } 1 \leq k \leq m \\ \pi_0 \cdot \frac{\lambda^k}{\mu^k \cdot m! \cdot m^{k-m}} = \pi_0 \cdot \frac{\rho^k m^m}{m!} & \text{für } k \geq m \end{cases} \\ \pi_0 &= \frac{1}{\sum_{k=0}^{m-1} \frac{(\rho m)^k}{k!} + \frac{(\rho m)^m}{m!(1-\rho)}} \\ P_Q &= \frac{\pi_0 (\rho m)^m}{m!(1-\rho)} = \frac{(\rho m)^m / (m!(1-\rho))}{\sum_{k=0}^{m-1} \frac{(\rho m)^k}{k!} + \frac{(\rho m)^m}{m!(1-\rho)}} \\ N_Q &= P_Q \cdot \frac{\rho}{1-\rho} \\ W &= \frac{\rho P_Q}{\lambda(1-\rho)} \\ T &= W + \frac{1}{\mu} \\ N &= \lambda T = \frac{\rho P_Q}{1-\rho} + m\rho \end{aligned}$$

Wahkeit, dass Job in Queue warten muss.  
 Erlang C-Formel  
 erwartete Anzahl Jobs in Queue  
 mittlere Wartezeit in Queue  
 mittlere Antwortzeit  
 mittlere Anzahl Jobs im System

4.5.10. Das M/M/m/m System. System mit  $m$  Servern und maximal  $m$  Jobs im System. Jobs, die ankommen, wenn alle Server belegt sind, werden abgewiesen.

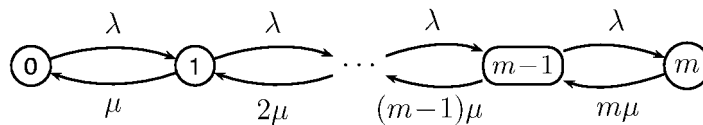


ABBILDUNG 11. Das M/M/m/m System

$$\begin{aligned} \pi_k &= \pi_0 \left(\frac{\lambda}{\mu}\right)^k \frac{1}{k!} \text{ für } 1 \leq k \leq m \\ \pi_0 &= \frac{1}{\sum_{k=0}^m \left(\frac{\lambda}{\mu}\right)^k \frac{1}{k!}} \\ \pi_m &= \frac{\left(\frac{\lambda}{\mu}\right)^m \frac{1}{m!}}{\sum_{k=0}^m \left(\frac{\lambda}{\mu}\right)^k \frac{1}{k!}} \end{aligned} \quad \text{Erlang B-Formel}$$

**Erlang B-Formel:** Gibt die Whkeit an, dass ein ankommender Job abgewiesen wird.

Gilt auch für M/G/m/m, wenn Erwartungswert der Bearbeitungszeit  $1/\mu$  ist.

4.5.11. *Warteschlangen Netzwerke.* Graphen, deren Knoten Warteschlangen-Systeme sind.

**offen:** Jobs können von aussen zum System hinzukommen, oder dieses verlassen.

**geschlossen:** Jobs sind im System gefangen.

**Burke's Theorem:** Die Ausgangsrate eines M/M/m-Systemes ist im stationären Zustand gleich der Ankunftsrate.

*Annahme* beim analysieren von Warteschlangen-Netzwerken:

- Servicezeit eines Jobs in jedem Warteschlangen-System wieder unabhängig.

**Jacksons Theorem:** für offene Netzwerke

$r_j$ : Rate der ankommenden Jobs im Knoten  $j$ .

$p_{ij}$ : Whkeit, dass Job Knoten  $i$  in Richtung Knoten  $j$  verlässt.

$p_{i,\text{exit}}$ : Whkeit, dass Job Knoten  $i$  und Netzwerk verlässt.

$$p_{i,\text{exit}} + \sum_{\forall j} p_{ij} = 1$$

$\lambda_j$ : Ankunftsrate beim Knoten  $j$

$$\lambda_j = r_j + \sum_{\forall i} \lambda_i p_{ij}$$

## 5. WORST-CASE EVENT SYSTEMS – ONLINE ALGORITHMS

### 5.1. Skis mieten.

**Definition:** Das Problem besteht aus folgenden Werten:

**Input  $u$ :** vom Gegner ausgesuchte Zahl, nach der man aufhört Ski zu fahren.

**Algorithm  $z$ :** Zahl, bei welcher Algorithmus Skis zum Preis 1 kauft.

Bemerkung: der Algorithmus kennt  $u$  nicht!

**Competitive Analysis:** Ein online Algorithmus  $A$  ist  $c$ -kompetitiv, wenn für alle Inputsequenzen  $I$  gilt:

$$\text{cost}_A(I) \leq c \cdot \text{cost}_{\text{opt}}(I) + k$$

$$c \cdot \text{gain}_{\text{Alg}}(I) \geq \text{gain}_{\text{opt}}(I)$$

oder

$$c = \frac{\text{cost}_A(I)}{\text{cost}_{\text{opt}}(I)} = \frac{\text{gain}_{\text{opt}}(I)}{\text{gain}_{\text{Alg}}(I)}$$

**strikt  $c$ -kompetitiv:**  $\iff k = 0$

$$\text{cost}_z(u) = \begin{cases} u & \text{wenn } u \leq z \\ z + 1 & \text{wenn } u > z \end{cases}$$

optimaler "offline" Algorithmus:

$$\text{cost}_{\text{opt}}(u) = \begin{cases} u & \text{wenn } u \leq z \\ 1 & \text{wenn } u > z \end{cases}$$

### 5.2. Skis mieten randomisiert.

5.3. **Zwei mögliche z.** Unser Algorithmus kauft mit Whkeit  $p_1$  nach  $z_1$  und mit  $p_2 = (1 - p_1)$  nach  $z_2$ .

$$\text{cost}_A(u) = \begin{cases} u & \text{wenn } u \leq z_1 \\ p_1 \cdot (z_1 + 1) + p_2 \cdot u & \text{wenn } z_1 < u \leq z_2 \\ p_1 \cdot (z_1 + 1) + p_2 \cdot (z_2 + 1) & \text{wenn } z_2 < u \end{cases}$$

Gegner wählt mit Whkeit  $q_1$   $z_1$  und mit  $q_2$   $z_2$ :

$$\begin{aligned} \text{cost}_A &= p_1 \cdot (z_1 + 1) + p_2 \cdot (q_1 z_1 + q_2 (z_2 + 1)) \\ \text{cost}_{\text{opt}} &= q_1 z_1 + q_2 z_2 \end{aligned}$$

5.3.1. *Verteilungen.*

$p(z)$ : Wahrscheinlichkeitsverteilung des Algorithmus  $z$

$$\int p(z) dz = 1$$

$d(u)$ : Wahrscheinlichkeitsverteilung des Gegners  $u$

$$\int d(u) du = 1$$

Trick:

- Wir wollen im Erwartungswert immer kompetitiv sein.
- $\implies \text{cost}_A(I) = c \cdot \text{cost}_{\text{opt}}(I)$  oder  $c \cdot \text{gain}_{\text{Alg}}(I) = \text{gain}_{\text{opt}}(I)$
- Gleichungen aufstellen, ableiten und nach  $p(z)$  auflösen.

*Theorem:*  $p(z) = \frac{e^{-z}}{e-1}$  liefert uns einen Alogrithmus, welcher im Erwartungswert  $\frac{1}{e-1}$ -kompetitiv ist.

5.4. **Untere Schranken.** *Theorem "Vom Neumann/Yao Principle":* Wähle eine Problem-Verteilung (zB  $d(u)$ ). Wenn alle deterministischen Algorithmen für diese Verteilung mindestens  $c$  kosten, ist dies die unterste Schranke für randomisierte Algorithmen.

Wähle Verteilung geschickt, um untere Schranke zu erhöhen.

5.4.1. *Ski rental.*

Die schlechteste Verteilung mit der höchsten unteren Schranke ist:

$$d(u) = \begin{cases} \frac{1}{e^u} & \text{für } 0 < u < 1 \\ \frac{1}{e} & \text{für } u = \infty \end{cases}$$

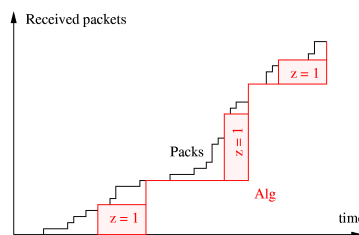


ABBILDUNG 12. Der  $z = 1$  Algorithmus

5.5. **The TCP Acknowledgement Problem.**

**Das Problem:** Der Empfänger versucht die Summe von ACKs  $k$  und die Latenz zu minimieren:

$$\min k + \sum_{i=1}^n \text{latency}(i)$$

**Der  $z = 1$  Algorithmus:** Immer wenn ein Rechteck mit der Fläche 1 eingeschrieben werden kann, schickt der Algorithmus ein ACK. Siehe Abbildung 12.

- Dies ist der optimale Algorithmus.
- $z = 1$  Algo ist 2-kompetitiv.
- Das TCP ACK ist wie Ski rental. Wenn man auf ein Rechteck der Grösse  $z$  mit der Wahrscheinlichkeit  $p(z) = e^{-z}/(e-1)$  wartet, ist der Erwartungswert  $e/(e-1)$ .
- *Halbtax-Problem*: Ein Halbtax, welches den Preis um  $\beta$  reduziert, ist  $e/(e-1+\beta)$  kompetitiv.

### 5.6. Das TCP Congestion Control Problem.

$u_t$ : Unbekannte obere Schranke der Bandbreite in der Periode  $t$ .

$x_t$ : Gesendete Pakete in der Periode  $t$ .

**Kosten:**

$$\text{cost}_t = \begin{cases} u_t - x_t & \text{wenn } x_t \leq u_t \text{ (zu wenig gesendet)} \\ \alpha(x_t - u_t) & \text{sonst (zu viel gesendet, Pakete gehen verloren)} \end{cases}$$

#### 5.6.1. Das statische Modell. Bandbreite bleibt konstant bei $u$

*Algorithmen*

**binary search:** braucht  $\log n$  Suchschritte mit Kosten  $u = \Theta(n)$

$$\text{cost} = \Theta(n \log n)$$

**AIMD:** Additive Increase, Multiplicative Decrease

$$\text{cost} = \Theta(n^2)$$

**Shrink:** operiert im Intervall  $[i, j]$

- Algorithmus:

(1) Finde die richtige Obergrenze  $j$  der Form:  $2^k < j \leq 2^{k+1}$

(2) Intervall  $[i, j]$ :  $2^{t-1} + 1 \leq i < j \leq 2^t$ . Jetzt testen wir

$$i + \max\left(1, \frac{2^t}{2^{m+1}}\right)$$

$m$  ist die grösste Ganzzahl, dass  $j - i < 2^t / 2^{2^m}$

passe  $[i, j]$  an.

- $\text{cost} = O(n \log \log n)$ ,

Shrink ist asymptotisch also fast optimal.

**theoretische untere Grenze:** ist  $O(u \log \log u / \log \log \log u)$

### 5.7. Das dynamische Modell. Der Gegner wählt eine Sequenz $\{u_t\}$ .

$$\text{gain}_{x_t} = \begin{cases} x_t & \text{wenn } x_t \leq u_t \\ 0 & \text{sonst} \end{cases}$$

Gegner setzt immer  $x_t = u_t$  und hat Kosten 0  $\implies$  wir können nicht kompetitiv sein und müssen den Gegner irgendwie einschränken!

#### 5.7.1. Bandbreite in einem fixen Intervall. Gegner sucht $u_t \in [a, b]$ .

*Deterministischer Algorithmus*

- Wenn der Algo  $x_t > a$  wählt, macht der Gegner  $u_t = a \implies \text{gain} = 0$
- $\implies$  wir müssen also  $x_t = a$  wählen, um  $\text{gain} = a$  zu haben
- $\implies$  der Gegner weiss dies und wählt  $u_t = b$
- $\implies$  wir sind also  $b/a$ -kompetitiv

*Randomisierter Algorithmus*

Wir wählen

$$x_t = \begin{cases} a & \text{mit Whkeit } p_a \\ (a, b] & \text{mit Whkeitsdichte } p(a), \end{cases} \quad p_a + \int_a^b p(x) dx = 1$$

wir sind  $c = 1 + \ln \frac{b}{a}$ -kompetitiv  
 untere Schranke:  $c = 1 + \ln \frac{b}{a}$ ,  $p_b = a/b, p(u) = a/u^2$

5.7.2. *Multiplikative Änderung der Bandbreite.* Der Gegner wählt  $u_t$ , dass  $u_t/\mu \leq u_{t+1} \leq \mu \cdot u_t$ . Also kennt  $u_1$  und  $\mu$ .

- Wenn Gegner möglichst schnell erhöht:  $u_{t+1} = \mu u_t$
- $\Rightarrow$  wir machen das gleiche:  $x_{t+1} = (1 - \epsilon)\mu x_t$
- $\Rightarrow \lim_{t \rightarrow \infty} \frac{u_t}{x_t} = \frac{\mu^t}{(1-\epsilon)^t \cdot \mu^t} = \infty$
- *Algorithmus 1:*

$$x_{t+1} = \begin{cases} \mu \cdot x_t & \text{wenn } x_t \text{ erfolgreich} \\ 1/\mu^3 \cdot x_t & \text{sonst} \end{cases}$$

- Nach jeder erfolglosen Transaktion kommt eine erfolgreiche.
- Eine erfolgreiche Runde ist  $\mu^4$ -kompetitiv.
- Der Algorithmus ist  $(\mu^4 + \mu)$ -kompetitiv.

- *Algorithmus 2:*

$$x_{t+1} = \begin{cases} \mu \cdot x_t & \text{wenn } x_t \text{ erfolgreich} \\ 1/2 \cdot x_t & \text{sonst} \end{cases}$$

- Der Algorithmus ist  $4\mu$ -kompetitiv.

## 6. NETWORK CALCULUS

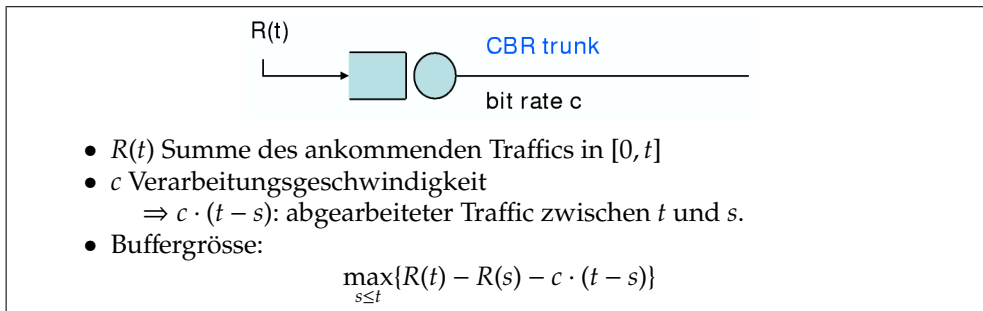


ABBILDUNG 13. Beispiel Network Calculus

### 6.1. Network Calculus.

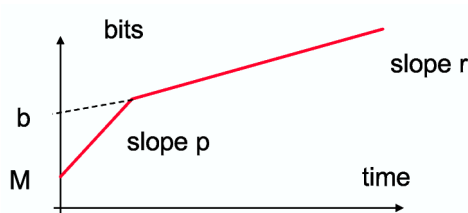


ABBILDUNG 14. Ankommens-Kurve  $\alpha(u) = \min(p \cdot u + M, r \cdot u + b)$

#### 6.1.1. Ankommens-Kurven.

**Ankommens-Kurve:**

$$R(t) - R(s) \leq \alpha(t - s)$$

Beispiele:

- "leaky bucket":

$$\alpha(u) = r \cdot u + b$$

- vernünftige Ankommens-Kurve im Internet:

$$\alpha(u) = \min(p \cdot u + M, r \cdot u + b)$$

**Theorem:**  $\alpha$  kann mit einer *sub-additiven* Funktion ersetzt werden:

$$\Rightarrow \alpha(s + t) \leq \alpha(s) + \alpha(t)$$

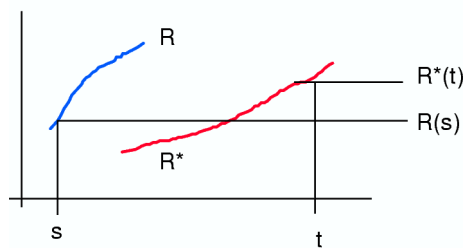


ABBILDUNG 15. Service-Kurve

6.1.2. Service-Kurven.

$R^*$ : Abgearbeiteter Traffic

**Service-Kurve:**  $\beta$

$$R^*(t) - R(s) \leq \beta(t - s)$$

Beispiele: Abbildung 16

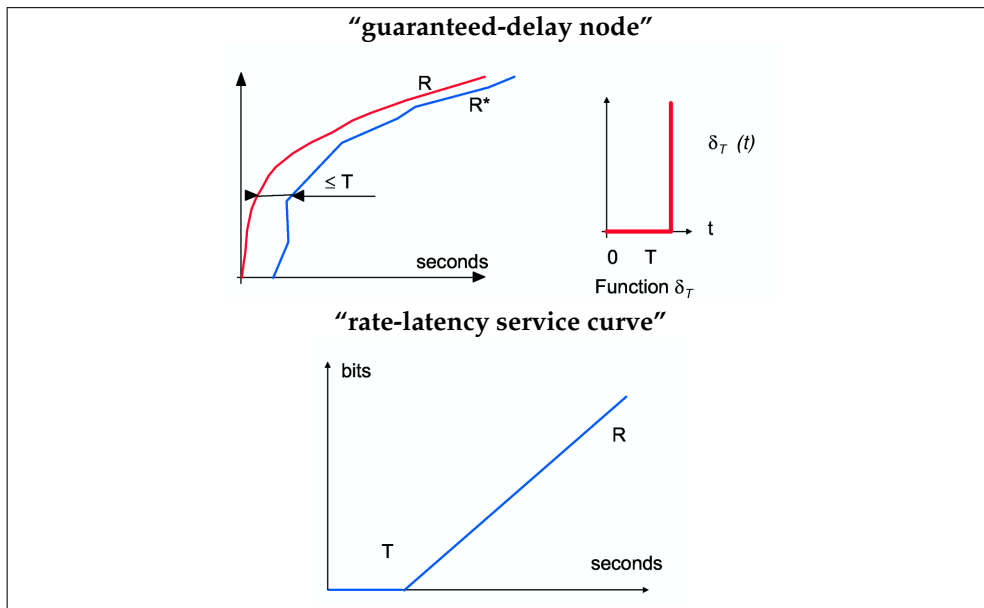


ABBILDUNG 16. Beispiele für Service Kurven

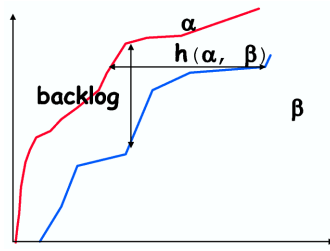


ABBILDUNG 17. Schranken für delay und backlog

6.1.3. *Schranken für delay und backlog.* Siehe Abbildung 17.

**backlog:** minimale Größe des Buffers

$$\text{backlog} \leq \max(\alpha(s) - \beta(s))$$

**delay:** maximaler Delay

$$\text{delay} \leq h(\alpha, \beta)$$

**Beispiel:** Abbildung 18

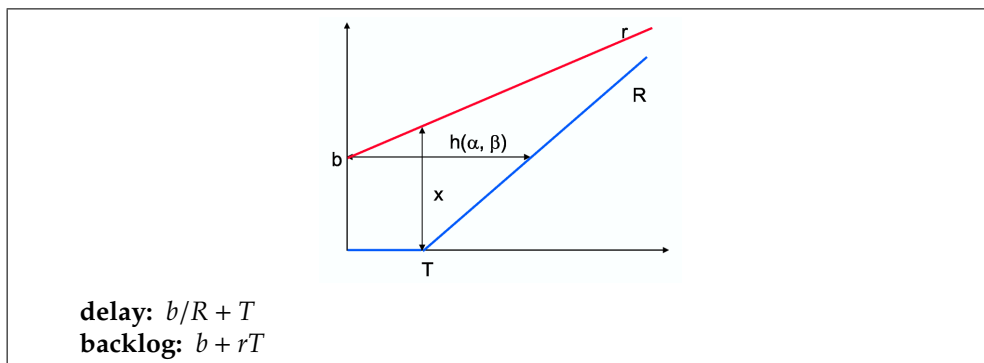


ABBILDUNG 18. Für vernünftige Ankommens- und Service-Kurven

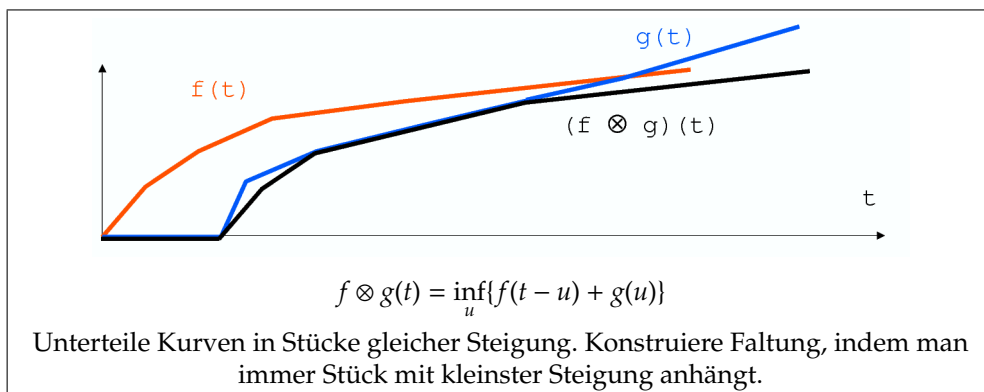


ABBILDUNG 19. Min-plus Faltung

6.1.4. *Min-plus Faltung.* Siehe Abbildung 19.

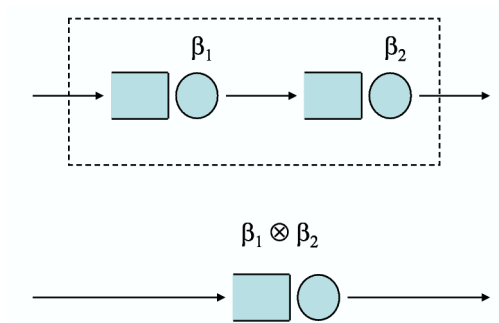


ABBILDUNG 20. Systeme in Serie

6.1.5. *Systeme in Serie.* Siehe Abbildung 20.

## 6.2. Adversarial Queuing Theory.

### 6.2.1. Model des Netzwerkes.

- *Annahmen*
  - Das Netzwerk ist ein gerichteter Graph.
  - Queues sind unbegrenzt.
  - Packete müssen durch Netzwerk geroutet werden.
  - Bei jedem Knoten müssen Packete gescheduled werden.
- *Vereinfachungen*
  - Alle Packete sind gleich gross.
  - Alle Links haben die gleiche Bandbreite.
  - Das Netzwerk entwickelt sich synchron, in Schritten.

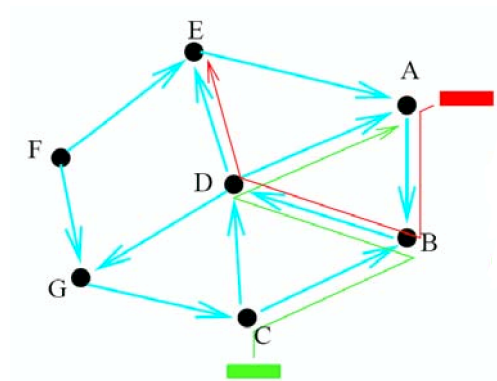


ABBILDUNG 21. Beispiel Adversarial Queuing Theory

### 6.2.2. Adversarial Queuing Theory Model.

- Der Gegner wählt die Ankunftszeiten und Routen der Packete.
- Der Gegner ist mit  $(r, b)$  begrenzt:
  - r:**  $r \leq 1$
  - b:**  $b \geq 1$

Im Schritt  $s$  können höchstens  $r \cdot s + b$  Packete ankommen, welche die Kante  $e$  kreuzen müssen.

### 6.2.3. *Stabilität.*

**Stabilität:** Eine scheduling policy  $P$  im Netzwerk  $G$  ist stabil bei der Rate  $(r, b)$ , wenn eine Grenze  $C(G, r, b)$  existiert, so dass kein  $(r, b)$ -Gegner mehr als  $C(G, r, b)$  Pakete ins Netzwerk bringt.

**universell stabil:**  $P$  ist für alle  $r < 1$  in jedem Netzwerk  $G$  stabil.

**universell stabil:**  $G$  ist für alle  $r < 1$  mit jeder "greedy scheduling policy"  $P$  stabil.

### 6.2.4. *Einige Resultate. . . .*