# Zusammenfassung Diskrete Ereignssysteme
# WS 05/06, D-ITET, ETH Zürich

Chris Walser
walserc@ee.ethz.ch

28. Februar 2006

Dozent: Prof. R. Wattenhofer

## Inhaltsverzeichnis

# 1 Automata and Languages

Definitionen:

- Alphabet $\Sigma$: Set von Symbolen welche der Automat versteht
- Wort: Sequenz von Symbolen.
  Leeres Wort: Wort das keine Symbole enthält, wird mit $\varepsilon$ bezeichnet.
  Länge eines Wortes: Anzahl Symbole des Wortes. z.B.$|\varepsilon| = 0$.
- Endlicher Automat/finite automaton/FA: ein FA ist ein 5-Tupel $(Q, \Sigma, \delta, q_0, F)$ mit:
  - $Q$: Zustände
  - $\Sigma$: Alphabet
  - $\delta$: Transformationsfunktion
  - $q_0$: Anfangszustand
  - $F$: Akzeptierende Zustände
- Akzeptiertes Wort: ein Wort ist akzeptiert von einem Automaten falls ein Pfad der bei $q_0$ beginnt, in einem akzeptierenden Zustand endet.
- Sprache: Eine Sprache ist ein Set von Worten die ein FA akzeptiert. Eine Sprache $L$ wird regulär genannt wenn ein FA existiert der die Sprache $L$ erkennt. Alle endlichen Sprachen sind regulär (Umkehrung gilt aber i.A. nicht!).
  Sprachen lassen sich verbinden mithilfe der regulären Operation $\bullet$. Dabei gilt: $L \bullet \varepsilon = L$ und $L \bullet \emptyset = \emptyset$ ($\emptyset$...Leere Sprache (nicht dasselbe wie leeres Wort $\varepsilon$)).

## 1.1 Regular Operations

Es gibt 4 reguläre Operationen:

| Operation | Symbol | Bedeutung |
|---|---|---|
| Union | $\bigcup$ | Match one of the patterns |
| Concatenation | $\bullet$ | Match patterns in sequence |
| Kleene-* | $*$ | Match pattern 0 or more times |
| Kleene-+ | $+$ | Match pattern 1 or more times |

In dieser Vorlesung muss ein Pattern immer auf das ganze Wort zutreffen (und nicht wie bei Unix auf einen Teil des Wortes).
Eigenschaften:

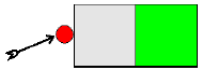
- $L \bullet \{\varepsilon\} = L$ mit $\varepsilon$... Sprache die aus dem leeren Wort besteht.
- $L \bullet \varnothing = \varnothing$ mit $\varnothing$... leere Sprache

## 1.2 Zusammenschaltung von FA's

Zusammenschaltung von FA's ist viel komplizierter als Zusammenschaltung von NFA's. Sie erfolgt über das Kreuzprodukt (Cartesian Product Construction). Siehe Slides 1/30 - 1/40.

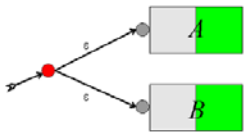
## 1.3 Nondeterministic Finite Automata (NFA)

Ein NFA ist gleich definiert wie ein FA, nämlich durch $M = (Q, \Sigma, \delta, q_0, F)$, ausser das $\delta$ anders definiert ist: $\delta : Q \times \Sigma_\varepsilon \to P(Q)$.
In Zustandsdiagrammen sind bei NFA auch Pfeile zulässig, die mit einem $\varepsilon$ beschriftet sind. Solche Pfeile können immer gewählt werden (auch ohne Zustandsänderung). Zudem dürfen bei NFA von einem Zustand aus mehrere Pfeile mit der gleichen Beschriftung wegführen. In diesen beiden Fällen werden beim Durchlaufen des Automaten für jede wählbare Verzweigung ein Token (Agent) erzeugt. Diese Tokens sterben wenn sie bei einer Zustandsänderung nicht "weitergehen"können. Ein Wort ist akzeptiert von einem NFA wenn sich am Schluss mindestens ein Token in einem akzeptierenden Zustand befindet.
Schematische Darstellung eines NFA:

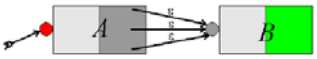


Dabei bezeichnen der rote Kreis den Anfangszustand $q_0$, der grüne (rechte) Bereich die akzeptierenden Zustände $F$ und der graue (linke) Bereich alle anderen Zustände.
Mithilfe der schematischen Darstellung lassen sich die Zusammenschaltungen von NFA's anschaulich erklären:

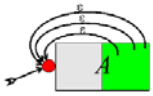
- Union (OR):



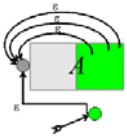

- Concatenation (AND):




- Kleene-+:




- Kleene-*:




## 1.4 Regular Expressions (REX), REX → NFA

Skript 1/16ff
Regular Expressions ermöglichen es, reguläre Operationen zu verknüpfen und dadurch neue Sprachen zu erzeugen.
Es gilt: eine Sprache die aus regulären Sprachen erzeugt wurde ist selbst wieder regulär.
Um die Notation zu verkürzen, können Klammern weggelassen werden. Damit Eindeutigkeit gewährleitstet bleibt, werden die Operatoren gewichtet:

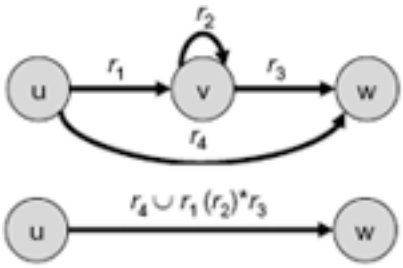
$$* > \bullet > \cup$$

**REX → NFA**
Aus jeder REX lässt sich in linearer Zeit ein NFA konstruieren der die REX implementiert (Bsp. siehe Slides 1/69 - 1/71). Aus diesem Grund heissen alle von NFA akzeptierten Sprachen regulär (da sie eine regular Expression implementieren).

## 1.5 NFA → REX

Skript 1/84ff
Vorgehen:

1. Konstruiere aus dem NFA einen GNFA:
   (a) Wenn von einem Zustand zu einem anderen mehr als 1 Pfeil führt, vereine diese mittels $\cup$.
   (b) Erstelle einen einzigen (unique) Startzustand.
   (c) Erstelle einen einzigen (unique) Endzustand.
2. Wenn der GNFA mehr als 2 Zustände hat: eliminere einen beliebigen Zustand nach folgendem Schema:




3. Iteriere 2.

### 1.6 Pumping Lemma

Skript 1/94ff
Um zu beweisen, dass eine Sprache regulär ist, kann ein Automat gezeichnet werden, welcher die Sprache akzeptiert. Um zu beweisen, dass eine Sprache nicht regulär ist, wird das Pumping Lemma verwendet:
Suche ein Wort $u$ in der Sprache $L$ und zerlege es so, dass folgendes gilt:

- $u = xyz$
- $|y| \geq 1$ (D.h. der Mittelteil darf nicht leer sein)
- $|xy| \leq p$ ($p$ ist die sog. pumping number)
- $xy^iz \in L \quad \forall i \geq 0$

Wenn die Zerlegung und $p$ geschickt gewählt wird, kann mit der letzten Bedingung gezeigt werden, dass $xy^iz$ nicht in $L$ ist und die Sprache somit nicht regulär ist. Man muss zeigen, dass es unter keiner möglichen Aufteilung möglich ist, das gewählte Wort zu pumpen.
Beweise mit dem Pumping Lemma sind immer Widerspruchsbeweise: Nehme an, $L$ sei regulär und beweise, dass das Pumping Lemma nicht erfüllt ist. $\Rightarrow$ Widerspruch $\Rightarrow$ $L$ ist nicht regulär. Siehe dazu auch Nachbesprechung Übung 3.
Achtung: Wenn das Pumping Lemma erfüllt ist heisst das nicht, dass die Sprache regulär ist. Vielmehr gilt:

- Sprache lässt sich nicht pumpen $\Rightarrow$ Sprache ist nicht regulär.
- Sprache lässt sich pumpen $\Rightarrow$ Sprache kann regulär sein, muss aber nicht (es gibt auch nichtreguläre Sprachen die sich pumpen lassen).

Trick: jede reguläre Sprache lässt sich in einem endlichen Automaten realisieren. Wenn also das Pumping Lemma erfüllt ist kann gezeigt werden, dass es keinen endlichen Automaten gibt mit dem sich die Sprache darstellen lässt um zu beweisen, dass eine Sprache nicht regulär ist.

Reguläre Sprachen sind eine Untergruppe der kontextfreien Sprachen.

## 2 Smarter Automata

### 2.1 Context Free Grammars (CFG)/Context Free Languages (CFL)

Skript 2/5ff
Eine CFG besteht aus:

- einem endlichen Set von Symbolen/Variablen/non-Terminalen (Grossbuchstaben)
- einem endlichen Set von Terminalen/Alphabet (Kleinbuchstaben).
- einem endlichen Set von Produktionen der Form $v \rightarrow w$
- einem Startsymbol.

Eine CFL ist die Menge aller terminal strings die vom Startsymbol einer CFG aus erreicht werden können.

Jede CFG lässt sich in einem PDA realisieren.

**Derivation tree**: um eine CFL zu visualisieren, kann eine Ableitungsbaum verwendet werden. Dieser stellt den Ablauf der Ersetzungen dar (Bsp. siehe Skript 2/11).

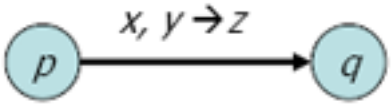
**Mehrdeutigkeit**: eine Grammatik ist mehrdeutig, wenn es einen String $x \in L(G)$ gibt, der auf mind. 2 verschiedene Arten erzeugt werden kann (d.h. es gibt mind. 2 verschiedene Derivation Trees für diesen String) (Bsp. siehe Skript 2/17).

## 2.2 Push-Down Automata (PDA)

Skript 2/21ff
Grammatiken können durch PDA's implementiert werden. Ein PDA ist ein endlicher Automat der durch einen Stack ergänzt wird. Ein PDA hat einen Stack der folgende Operationen zulässt:

- Push: ein neues Element auf den Stack legen
- Pop: Das oberste Element des Stacks entfernen
- Peek: Das oberste Element des Stacks anschauen ohne es zu entfernen.

Um zu testen, ob der Stack leer ist, wird als erstes Symbol eine Spezialzeichen (\$) auf den Stack gelegt. Wenn nun dieses Zeichen wieder vom Stack gelesen wird ist klar, dass der Stack leer sein muss.
Übergänge werden wie folgt bezeichnet:




D.h. wenn ein $x$ das nächste Inputysmbol ist und auf dem Stack ein $y$ liegt, wird vom Zustand $p$ nach $q$ gewechselt und $y$ wird auf dem Stack durch $z$ ersetzt.
Es gilt also folgende Notation:

$$\text{Input, Pop} \rightarrow \text{push}$$

Spezialfälle:

- $x, \varepsilon \rightarrow z$. Lege $z$ auf den Stack ohne von diesem zu lesen.
- $x, y \rightarrow \varepsilon$. Nimm oberstes Symbol vom Stack.
- $x, \varepsilon \rightarrow \varepsilon$. Lese nur $x$ ohne auf dem Stack etwas zu machen.

**Umwandlung CFG → PDA**: siehe Skript 2/40ff

## 2.3 Counter Automata (CA)

Übung 4
Ein Counter Automata ist ein Automat der wie ein PDA funktioniert, aber anstelle des Stacks einen Zähler (Counter) besitzt. Der Automat kann den Zähler inkrementieren, dekrementieren und auf 0 überprüfen. Weil theoretisch alle existierenden Daten in einen einzigen Integerwert codiert werden können, besitzt der CA einen unendlich grossen Speicher. PDA sind mächtiger als CA, da PDA's CA's simulieren können.

## 2.4 Rechtslineare Grammatiken

Skript 2/33ff
Eine rechstlineare Grammatik (right linear grammar) ist eine CFG, in der jede Produktion die Form $A \rightarrow uB$ oder $A \rightarrow u$ hat wenn $u$ ein Terminal ist und $A$, $B$ Variablen.

## 2.5 Chomksy Normal Form (CNF)

Skript 2/35ff
Jede Regel einer CFG in Chomsky Normal Form (CNF) muss folgende Form haben:

- $S \rightarrow \varepsilon$
- $A \rightarrow BC$
- $A \rightarrow a$

D.h. $\varepsilon$ darf nur auf der rechten Seite der Startvariablen auftauchen und auf allen anderen rechten Seiten von Produktionen dürfen nur 2 Variablen oder 1 Terminalsymbol stehen. OR-Verknüpfungen sind erlaubt (z.B. $S \to \varepsilon|AB|a$)

Umwandlung CFG $\to$ CNF: (siehe auch Skript 2/37ff und Übung 6, Aufgabe 1)

1. Die Startvariable darf nicht auf der rechten Seite von Regeln stehen. Falls doch muss eine neue Startvariable eingeführt werden die auf die alte Startvariable zeigt.
2. Entferne alle $\varepsilon$ ausser dasjenige bei der Startvariable. Dazu müssen bei den Produktionen u.U. neue Kombinationen hinzugefügt werden.
   Bsp.: Aus $B \to \varepsilon|a|aBa$ wird $B \to a|aBa|aa$
3. Entferne alle Produktionen der Form $A \to B$ (d.h. Produktionen auf eine einzige Variable).
4. Füge neue Variablen und Produktionen der Form $A \to BC$ ein um Produktionen die auf mehr als 2 Variablen oder mehr als 1 Terminalsymbol führen aufzulösen.
5. Am Schluss dürfen nur noch Produktionen der Formen $A \to BC$ oder $A \to a$ vorhanden sein (ausser bei Startvariable).

## 2.6 Tandem Pumping Lemma

Skript 2/47ff
Beispiel für eine nicht-kontextfreie Grammatik: $a^n b^n c^n$
Um zu zeigen, dass eine Sprache nicht-kontextfrei ist, kann das Tandem Pumping Lemma verwendet werden: Für eine Sprache $L$ existiert eine tandem-pumping Zahl $p$ mit $|L| \geq p$. Finde ein Wort $w \in L$ das sich folgendermassen aufteilen lässt:

- $w = uvxyz$
- $|vy| \geq 1$. Die beiden Teile die gepumpt werden, müssen mind. 1 Element enthalten.
- $|vxy| \leq p$
- $uv^i xy^i z \in L$ für alle $i \geq 0$ ($i$ kann beliebig gewählt werden).

Um zu zeigen, dass eine Sprache nicht-kontextfrei ist, wird ein Widerspruchsbeweis geführt: nehme an, $L$ sei kontextfrei und zeige, dass sie das Tandem-Pumping-Lemma nicht erfüllt. $\Rightarrow$ Sprache muss nicht-kontextfrei sein.
Achtung: wenn das Tandem-Pumping-Lemma erfüllt ist heisst das noch nicht, dass die Sprache kontextfrei ist (siehe auch Erklärungen bei Pumping-Lemma).

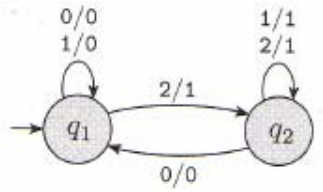
## 2.7 Context Sensitive Grammars (CSG)

Skript 2/44ff
Jeder CFG ist auch ein CSG. D.h. es gilt: Regular $\Rightarrow$ context-free $\Rightarrow$ context-sensitive.
Eine Grammatik ist context-sensitiv, wenn die Länge der linken Seite $\leq$ Länge der rechten Seite der Produktionen ist.
Bsp.: context-sensitiv: $S \to aa$
Nicht context-sensitiv: $ABC \to a$
$a^n b^n c^n$ ist eine context-sensitive Grammatik.

Jede Sprache die sich mit einem Petrinetz realisieren lässt ist context-sensitiv.

## 2.8 Transducers

Skript 2/50ff
Transducer sind Automaten, die einen Outputstring haben anstatt nur akzeptierende/zurückweisende Zustände. Ein Transducer kann mithilfe des Outputstrings etwas machen (z.B. einfache Roboter, siehe Übung 6, Aufg. 4)




Die Übergänge werden dabei wie folgt bezeichnet: Input/Output

### 2.9 Turing Maschinen (TM)

Skript 2/52ff
Eine Turing Maschine ist eine Maschine mit einem endlichen read-only-memory (Zustände) und einem unendlichen read/write-tape-memory. Es wird ein Schreib-/Lesekopf verwendet, der dem Tape entlang wandert.
Turing Maschinen sind gleich mächtig wie Computer (PC's).

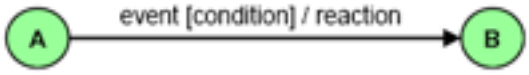
## 3 Specification Models

### 3.1 Zustandsdiagramme (State Charts)

Skript 3/3ff
In einem Statechart können Zustände zusammengefasst werden zu sog. Super-states. Ein Super-state hat einen default entrypoint, d.h. ein Übergang der defaultmässig genommen wird wenn der Super-state „betreten" wird. Der default entrypoint wird mit einem Punkt und einem Pfeil auf den entsprechenden Zustand markiert. Wird ein Pfeil von aussen auf den Rahmen des Superstates gezeichnet heisst das, dass dieser Übergang den default entrypoint verwendet.
Es gibt 2 Arten wie Superstates kombiniert werden können:

- AND-super-state: alle Unterzustände laufen parallel ab. Die Unterzustände werden durch gestrichelte Linien abgetrennt.
- OR-super-state: nur einer der Unterzustände läuft ab, d.h. man muss sich für einen entscheiden.

In Statecharts werden Kanten wie folgt beschriftet:

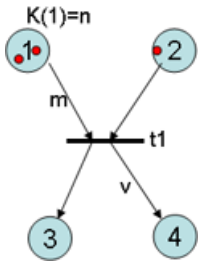



Jedes dieser Labels ist optional. Zudem können in einem Label mehrere Bedingungen untergebracht werden.
Ein Übergang von einem Zustand in den nächsten läuft folgendermassen ab:

1. Übergang wird evaluiert.
2. Alle reactions und sonstige Änderungen werden berechnet, aber noch nicht zugewiesen.
3. Der Übergang in den neuen Zustand wird vollzogen, dann werden die berechneten Änderungen zugewiesen/aktiv.

### 3.2 Petri Nets

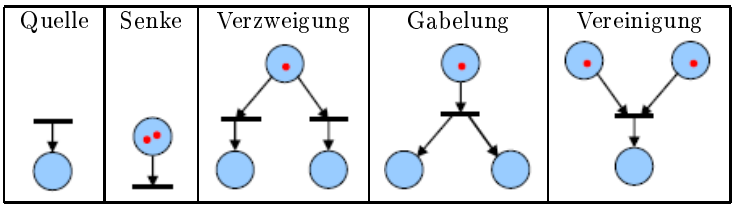
Skript 3/23ff
Petri Nets sind bipartite, gerichtete Graphen die asynchrone Zustandsübergänge darstellen können. Petrinetze sind nicht-deterministisch! Ein allgemeiner Zustandsübergang sieht folgendermassen aus:



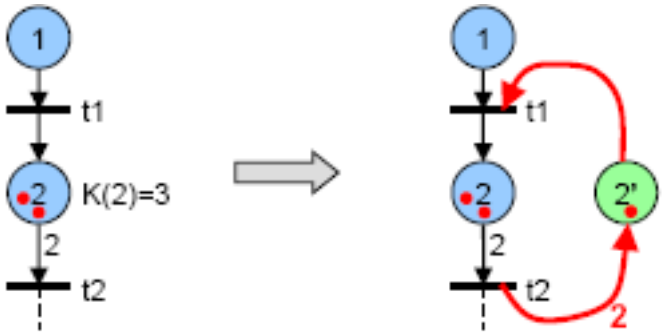
Transitionen entsprechen Events/Ereignissen.

Der Übergang/Transition $t1$ kann feuern, wenn im Zustand 1 $m$ Tokens liegen und im Zustand 2 1 Token. Wenn der Zustand feuert, entfernt er die „benötigte“ Anzahl Tokens aus den Zuständen 1 und 2 und fügt dafür in den Zuständen 3 und 4 1 bzw. $v$ neue Tokens ein. Der Zustand 1 hat zudem eine Kapazität, d.h. er kann maximal $n$ Tokens aufnehmen.
Petrinetze sind nicht deterministisch: Wenn mehrere Übergänge feuern können, wird ein zufälliger ausgewählt.

Folgende Spezialfälle existieren:




Kapazitäten können wie folgt entfernt werden:




Dabei muss darauf geachtet werden, dass die Zustände 2 und 2' zusammen gleichviele Tokens haben wie die Kapazität beträgt und dass die Übergänge des neuen Zustands gleiche Gewichtungen aufweisen wie die entsprechenden Übergänge der alten Zustände.

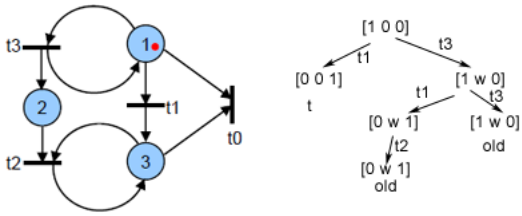
**Eigenschaften von Petri-Netzen:**

- Jeder FA kann mit einem Petrinetz modelliert werden.
- Reachability/Erreichbarkeit: Ein Zustand ist erreichbar wenn es einen Transitionenfolge gibt, die in diesem Zustand endet.
- K-Boundedness/K-Beschränktheit: Ein Petrinetz ist $k$-beschränkt, wenn die Anzahl Tokens in jedem Platz nie grösser als $k$ ist.
- Safety: Ein Petrinetz ist sicher, wenn jeder Knoten zu jeder Zeit maximal ein Token enthält.
- Liveness: Die Liveness ist ein Mass für die Anzahl Male die eine Transition feuern kann.
  - dead: Eine Transition wird als tot bezeichnet, wenn sie für keine mögliche Transitionenabfolge ab einem bestimmten Zustand gefeuert werden kann. Tote Transitionen können ohne Verlust aus dem Netz entfernt werden.
  - L1-Live: Eine Transition ist L1-live wenn sie mit min. einer Transitionenfolge aus dem Startzustand abgefeuert werden kann.
  - L2-Live: Die Transition kann für jede beliebige Zahl $k$ min. $k$ mal gefeuert werden in einer Transitionenabfolge aus dem Startzustand.
  - L3-Live: Die Transition erscheint unendlich viele Male in Transitionenabfolgen ab dem Startzustand.
  - L4-Live/live: Eine Transition ist L4-live/live wenn sie L1-live ist für jeden von einem best. Zustand aus erreichbaren neuen Zustand.

- die Liveness eines Netzes entspricht der Liveness des „schlechtesten“ Knotens. Trick: Netz reduzieren um Liveness einfacher bestimmen zu können.

L4-live ⇒ L3-live ⇒ L2-live ⇒ L1-live
Beispiele: siehe Skript 3/40

- Coverability Tree/Abdeckungsbaum: Wird benutzt um bei kleinen Petrinetzen zu analysieren, welche Tokenverteilungen erreichbar sind. Es wird ein neues Symbol $\omega$ eingeführt, das in einem Zustand anzeigt, dass dieser Zustand eine beliebige Anzahl Tokens enthält.
Bsp.:



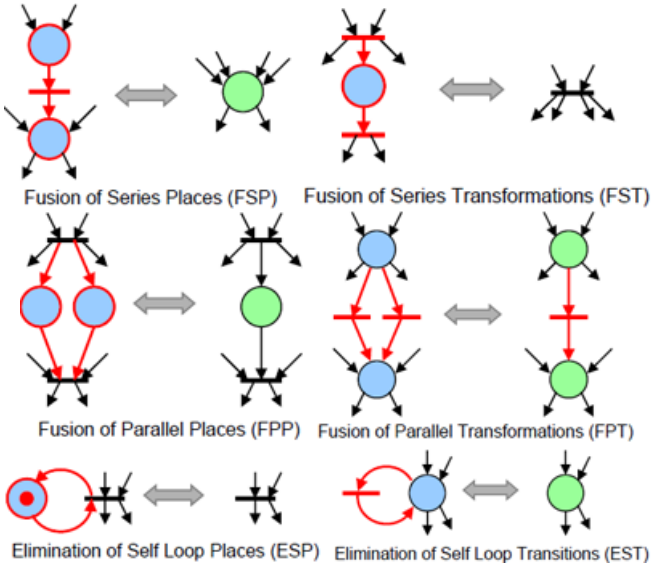

Dabei gelten folgende Bezeichnungen:

- $[abc]$ ... Anzahl Tokens in den Zuständen $a$, $b$, $c$ (im Bsp. sind die Zustände mit 1,2,3 beschriftet).
- $t$ ... „dead end“.
- old... eine bereits bekannte Tokenverteilung (Rekursion). Wenn diese Verteilung immer wieder erreicht wird, kann für die entsprechende Stelle ein $\omega$ gesetzt werden.

Aus dem Abdeckungsbaum lassen sich einige Dinge ablesen:

- Boundedness: Wenn $\omega$ nicht im Abdeckungsbaum auftritt, ist das Netz beschränkt.
- Safety: Wenn jeder Knoten im Abdeckungsbaum nur immer 0 oder 1 Token enthält, ist das Netz safe. Wenn jeder Knoten höchstens $k$ Token enthält, ist das Netz $k$-safe.
- dead: Eine Transition ist tot, wenn sie im Abdeckungsbaum nicht auftritt.
- Reachability Tree/Erreichbarkeitsbaum: Für bounded Petrinetze sind der Abdeckungsbaum und der Erreichbarkeitsbaum identisch.

- Incidence Matrix: Die Inzidenzmatrik stellt den Tokenfluss in einer Matrix dar. Jede Spalte steht für eine Transition, jede Zeile für einen Zustand. Zusätzlich muss noch eine Initialverteilung der Token angegeben werden.
- Reduction Rules: Ein Petrinetz lässt sich häufig stark vereinfachen ohne dass gewisse Eigenschaften (Liveness, Safeness, Boundedness) beinträchtigt werden. Dazu gibt es die folgenden 6 Regeln:



# 4 Stochastische diskrete Ereignisysteme

## 4.1 Wahrscheinlichkeitsrechnung

Skript 4/4ff

- Unabhängigkeit: Zwei Ereignisse $A$ und $B$ heissen unabhängig falls gilt: $P[A \cap B] = P[A] \cdot P[B]$ bzw. zeitkontinuierlich: $\Pr[X \leq x, Y \leq y] = \Pr[X \leq x] \cdot \Pr[Y \leq y]$
  Für unabhängige Zufallsvariablen gilt: $E[XY] = E[X]E[Y]$
  $Var[X+Y] = Var[X] + Var[Y]$
- Bedingte Wahrscheinlichkeit: $P[A|B] = \frac{P[A \cap B]}{P[B]} = \frac{P[A]P[B|A]}{P[B]}$
- Totale Wahrscheinlichkeit: $P[B] = \sum_{i=1}^{n} P[B|A_i]P[A_i]$
- Erwartungswert: $E[X] = \sum_{x \in W_X} x \cdot P[X = x]$ bzw. $E[X] = \int_{-\infty}^{\infty} x \cdot f_X(x)dx$ falls $\int_{-\infty}^{\infty} |x| f_X(x)dx$ endlich.
- Varianz: $Var[X] = E[X^2] - E[X]^2$
- Standardabweichung: $\sigma(X) = \sqrt{Var[X]}$
- Linearität des Erwartungswertes: Falls für die Zufallsvariablen gilt $X = a_1X_1 + \ldots + a_nX_n$, dann gilt für den Erwartungswert: $E[X] = aE[X_1] + \ldots + a_nE[X_n]$
- Dichte: $\int_{-\infty}^{\infty} f_X(x)dx = 1$
- Verteilungen:
  Jeder Dichte $f_X$ kann eine Verteilungsfunktion zugeordnet werden: $F_X(x) = \Pr[X \leq x] = \int_{-\infty}^{x} f_X(t)dt$
  Eigenschaft der Verteilung: $\Pr[a < X \leq b] = F_X(b) - F_X(a)$
  - Bernoulli-Verteilung (zeitdiskret):
    $P[X = 1] = p, \quad P[X = 0] = 1 - p$
    $E[X] = p, \quad Var[X] = p(1-p)$
  - Binomial-Verteilung (zeitdiskret):
    $P[X = k] = \binom{n}{k} p^k (1-p)^{n-k}$ für $0 \leq k \leq n$
    $E[X] = np, \quad Var[X] = np(1-p)$
  - Poisson-Verteilung (zeitdiskret):
    $P[X = k] = \frac{e^{-\lambda}\lambda^k}{k!}$ für $k \in \mathbb{N}_0$
    $E[X] = \lambda, \quad Var[X] = \lambda$
  - Geometrische Verteilung (zeitdiskret):
    $P[X = k] = (1-p)^{k-1}p$ für $k \in \mathbb{N}$
    $E[X] = \frac{1}{p}, \quad Var[X] = \frac{1-p}{p^2}$
  - Gleichverteilung auf $[a, b]$ (zeitkontinuierlich):
    $$f_X(x) = \begin{cases} \frac{1}{b-a}, & a \leq x \leq b \\ 0, & \text{sonst} \end{cases}$$
    $E[X] = \frac{a+b}{2}, \quad Var[X] = \frac{(b-a)^2}{12}$
  - Normalverteilung (zeitkontinuierlich):
    Parameter: $\mu$, $\sigma$
    $f_X(x) = \frac{1}{\sqrt{2\pi}\cdot\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$
    $E[X] = \mu, \quad Var[X] = \sigma^2$
  - Exponentialverteilung (zeitkontinuierlich):
    Parameter: $\lambda > 0$
    $$f_X(x) = \begin{cases} \lambda \cdot e^{-\lambda x}, & x \geq 0 \\ 0, & \text{sonst} \end{cases}$$
    $E[X] = \frac{1}{\lambda}, \quad Var[X] = \frac{1}{\lambda^2}$
    $$F_X(x) = \begin{cases} 1 - e^{-\lambda x}, & x \geq 0 \\ 0, & \text{sonst} \end{cases}$$
    Exponentialverteilung ist ein gutes Modell für: Dauer von Telefongesprächen, Zwischenankunftszeiten von Anfragen (d.h. Zeit zwischen 2 Anfragen), Ausführungszeit von Tasks.
    Eigenschaften der Exponentialverteilung:

* Gedächtnislosigkeit: Für alle $x, y > 0$ gilt: $\Pr[X > x + y | X > y] = \Pr[X > x]$
* Skalierung: Falls $X$ exponentialverteilt ist mit Parameter $\lambda$ so ist für $a > 0$ die Zufallsvariable $Y = aX$ ebenfalls exponentialverteilt mit Paramater $\frac{\lambda}{a}$.
* Warteproblem: Falls $X_1, \cdots, X_n$ unabhängig und exponentialverteilt mit Parametern $\lambda_1, \ldots, \lambda_n$ so ist $X = \min\{X_1, \ldots, X_n\}$ ebenfalls exponentialverteilt mit Parameter $\lambda_1 + \cdots + \lambda_n$

## 4.2 Markov-Ketten in diskreter Zeit

Skript 4/21ff
Markov-Prozess: Prozess, dessen weiterer Ablauf nur vom aktuellen Zustand abhängig ist, nicht aber von der Vergangenheit.

Markov-Kette: eine endliche, zeitdiskrete Markov-Kette ist eine Folge von Zufallsvariablen $X_t$ und einer Startverteilung $q_0 = (q_{00}, q_{01}, \ldots, q_{0,n-1})$ mit $q_0 \geq 0$ und $\sum_{i=0}^{n-1} q_{0,i} = 1$. $X_{t+1}$ darf dabei nur von $X_t$ abhängen für alle $t > 0$.

Übergangsmatrix:

$$q_t = q_0 P^t$$
$$q_{t+k} = q_t P^k \qquad \forall k \geq 0$$

Übergangszeit (Hitting time): Gibt die W'keit an, dass ein Zustand $j$ von einem anderen Zustand $i$ aus erreicht werden kann. Wird mit $T_{ij}$ bezeichnet. Aus der hitting time lassen sich folgende beiden Werte herleiten:

- erwartete Übergangszeit:
$$h_{ij} = E[T_{ij}] = 1 + \sum_{k:k \neq j} p_{ik} h_{kj}$$
Diese Formel ist rekursiv.
- Ankunftswahrscheinlichkeit:
$$f_{ij} = Pr[T_{ij} < \infty] = p_{ij} + \sum_{k:k \neq j} p_{ik} f_{kj}$$
Diese Formel ist rekursiv.

### 4.2.1 Stationäre Analyse von Markov-Ketten

Skript 4/38ff
Hierbei geht es um das Verhalten einer Markov-Kette für $t \to \infty$, d.h. nach sehr langer Zeit wenn sich das System eingeschwungen hat.
Dazu werden einige Definitionen gemacht:

- Stationäre Verteilung: Existiert nur für irreduzible Markov-Ketten. Beschreibt die W'keit mit welcher das System nach langer Zeit in einem bestimmten Zustand ist. Ein Zustandsvektor $\pi$ heisst stationäre Verteilung einer Markovkette wenn folgendes gilt:
  - $\pi = \pi P$ mit $P$ ...Übergangsmatrix
  - $\pi_i = \sum_j \pi_j p_{ij}$
  - $\sum_{j \in S} \pi_j = 1$. D.h. die Summe über alle Zustände muss Eins ergeben.

  Der Zustandsvektor lässt sich auf verschiedene Arten berechnen:
  - $\pi$ ist Eigenvektor von $P$ zum Eigenwert 1 falls $P$ doppeltstochastisch ist, d.h. falls alle Einträge von $P$ nichtnegativ und alle Zeilen- und Spaltensummen gleich Eins sind.
  - $\pi_j = \frac{1}{h_{jj}} \quad \forall j \in S$ sofern die Markov-Kette irreduzibel und endlich ist. Der Zustandsvektor ist in diesem Fall eindeutig.
  - $\lim_{t \to \infty} q_t = \pi$ für ergodische, endliche Markov-Ketten. Der Zustandsvektor ist in diesem Fall eindeutig und die Zustandsverteilung konvergiert unabhängig vom Startzustand gegen diese Verteilung.

- irreduzibel: Eine Markov-Kette ist irreduzibel wenn es für alle Zustände $i, j \in S$ eine Zahl $n \in \mathbb{N}$ gibt s.d. gilt: $p_{ij}^{(n)} > 0$. D.h. man muss von jedem Zustand in jeden anderen Zustand gelangen können.
- Periode: die Periode eines Zustandes $j \in S$ ist die grösste Zahl $\xi \in \mathbb{N}$ für die gilt:

$$\{n \in \mathbb{N}_0 | p_{jj}^{(n)} > 0\} \subseteq \{i \cdot \xi | i \in \mathbb{N}_0\}$$

  D.h. die Periode eines Zustandes ist die kleinste Zahl, mit der man periodisch einen Zustand in sich selber überführen kann (Bsp: man kann immer nach 2,4,6,... Übergängen von Zustand $A$ wieder zurück zu Zustand $A$ gelangen. Dann ist die Periode von Zustand $A$ $\xi = 2$)
- Aperiodisch: ein Zustand heisst aperiodisch wenn er die Periode $\xi = 1$ hat. Eine Markov-Kette heisst aperiodisch wenn alle ihre Zustände aperiodisch sind.
  Testbedingungen zur Überprüfung von Aperiodizität bei Zuständen: Wenn eine der folgenden Bedingungen gilt, so ist der Zustand aperiodisch:
  - $p_{jj} > 0$ (d.h. Der Zustand besitzt im Übergangsdiagramm eine Schleife).
  - $\exists n, m \in \mathbb{N} : p_{jj}^{(m)}, p_{jj}^{(n)} > 0$ und $\text{ggT}(m, n) = 1$ (d.h. Der Zustand liegt auf mindestens zwei geschlossenen Wegen, deren Längen teilerfremd sind).

  Die Aperiodizität einer irreduziblem Markov-Kette kann auf einfache Weise sichergestellt werden: Füge an allen Zuständen Schleifen ein, d.h. Übergänge vom Zustand auf sich selbst. Füge zu diesen Schleifen Übergangsw'keiten von $p = \frac{1}{2}$ ein und halbiere die W'keiten an allen anderen Kanten.
- ergodisch: Irreduzible, aperiodische Markov-Ketten heissen ergodisch.

## 4.3 Markov-Ketten in kontinuierlicher Zeit

Skript 4/57ff
Stochastische Prozesse in kontinuierlicher Zeit können als Markov-Ketten modelliert werden.

- Zeithomogen: Eine Markov-Kette heisst zeithomogen falls gilt: $\Pr[X(t+u) = j | X(t) = i] = \Pr[X(u) = j | X(0) = i]$
- Übergänge der zeitkontinuierlichen Markov-Kette: Die Übergänge sind keine W'keiten mehr wie bei der zeitdiskreten Markov-Kette sondern Übergangsraten.
- Übergangsrate: vom Zustand $i$ nach $j$: $\nu_{i,j} = \nu_i \cdot p_{i,j}$
  $p_{ij} = \frac{\text{Ausgangsverteilung der Kante } ij}{\sum \text{Ausgangsverteilungen von Knoten } i}$
- Aufenthaltswahrscheinlichkeiten: können mit folgender Differentialgleichung bestimmt werden:

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

  Die Lösung dieser Differentialgleichung ist meist aufwendig. Falls die Aufenthaltsw'keiten gegen eine stationäre Verteilung konvergieren, muss $\frac{d}{dt} q_i(t) = 0$ gelten.
  Für $t \to \infty$ erhält man ein lineares Gleichungssystem, das von einer stationären Verteilung $\pi$ erfüllt werden muss:

$$0 = \underbrace{\sum_{j:j \neq i} \pi_j \cdot \nu_{j,i}}_{\text{Zufluss}} - \underbrace{\pi_i \cdot \nu_i}_{\text{Abfluss}} \qquad (\text{„Knotenregel“})$$

$$\sum_j \pi_j = 1$$

  $\nu_i$ ...W'keit, aus Knoten $i$ „rauszukommen“ (Summe aller Ausgänge von $i$)
  $\nu_{i,j} = \nu_i p_{i,j}$
- Irreduzibel: Eine Markov-Kette heisst irreduzibel, falls jeder Zustand von jedem anderen aus erreichbar ist. Für irreduzible Markov-Ketten gilt:

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

- Stationäre Verteilung: Existiert nur für irreduzible Markov-Ketten. Berechnung der stationären Verteilung: siehe Skript Beispiel p.4/65

## 4.4 Warteschlangentheorie

Skript 4/66ff
Es werden Systeme von Servern untersucht, die Jobs abarbeiten. Die Ankunftszeiten der Jobs und die Bearbeitungsdauern auf den Servern werden als Zufallsvariablen modelliert.

### 4.4.1 Kendall-Notation $X/Y/m/\ldots$

Skript 4/68ff

$$X/Y/m/\ldots$$

Dabei bedeuten:

- $X$: Verteilung der Zwischenankunftszeiten (Zeiten zwischen zwei ankommenden Jobs, d.h. „Wartezeiten“).
- $Y$: Verteilung der reinen Bearbeitungszeiten (d.h. ohne Wartezeit in Queue) auf dem Server.
- $m$: Anzahl Server.
- $\ldots$: weitere Angaben wie z.B. Grösse der Warteschlange

$X$ und $Y$ sind unabhängige Zufallsvariablen. Die Verteilungen für $X$ und $Y$ werden angegeben als:

- D: deterministic (d.h. Gleichverteilung)
- M: memoryless (d.h. Exponentialverteilung)
- G: general (d.h. beliebige andere Verteilung)

### 4.4.2 M/M/1 Warteschlangen

Skript 4/70ff
Die Zwischenankunftszeiten und die Bearbeitungszeiten sind exponentialverteilt mit Parameter $\lambda$ (Ankunftsrate) und $\mu$ (Bedienrate) sowie Verkehrsdichte $\rho = \frac{\lambda}{\mu}$ (die Verkehrsdichte entspricht der mittleren Auslastung des Servers).

Stationäre Verteilung bei $M/M/1$:

- Für $\rho \geq 1$ konvergiert das System nicht. D.h. die Warteschlange wächst ins Unendliche. Die einzige Lösung ist dann: $\pi = (0, 0, \ldots)$
- Für $\rho < 1$ konvergiert das System gegen die stationäre Verteilung $\pi$ mit $\pi_k = (1-\rho)\rho^k$.

Weitere Formeln:

- Erwartungswert der Anzahl Jobs im System (Warteschlange + Server): $N = \frac{\rho}{1-\rho} = \frac{\lambda}{\mu-\lambda}$
- Varianz der Anzahl Jobs im System (Warteschlange + Server): $Var(N) = \frac{\rho}{(1-\rho)^2}$
- Mittlere Antwortzeit eines Jobs im Gleichgewichtszustand des Systems: $T = N\frac{1}{\lambda} = \frac{\rho}{\lambda(1-\rho)} = \frac{1}{\mu-\lambda}$
- Mittlere Wartezeit (ohne Bearbeitungszeit) eines Jobs im Gleichgewichtszustand des Systems: $W = T - \frac{1}{\mu} = \frac{\rho}{\mu(1-\rho)} = \frac{\rho}{\mu-\lambda}$
- Mittlere Anzahl Jobs in der Warteschlange (Variante von Little's Law): $N_Q = \lambda W = \frac{\rho^2}{1-\rho}$

### 4.4.3 Little's Law

Skript 4/75ff
Definitionen:

- $N(t)$: Anzahl Jobs im System (Warteschlange + Server) zur Zeit $t$.
- $N_t$: Durchschnittliche Anzahl Jobs im System zur Zeit $t$: $N_t = \frac{1}{t}\int_0^t N(\tau)d\tau$
- $N$: Anzahl Jobs im stationären Zustand: $N = \lim_{t\to\infty} N_t$
- $\alpha(t)$: Anzahl Jobs, die in $[0, t]$ angekommen sind.
- $\lambda_t$: Durchschnittliche Ankunftsrate: $\lambda_t = \frac{\alpha(t)}{t}$
- $\lambda$: Ankunftsrate. $\lambda = \lim_{t\to\infty} \lambda_t$

- $T_i$: Antwortzeit des $i$-ten Jobs (Wartezeit + Bearbeitungszeit).
- $T_t$: Durchschnittliche Antwortzeit. $T_t = \frac{\sum_{i=1}^{\alpha(t)} T_i}{\alpha(t)}$
- $T$: Durchschnittliche Antwortzeit der Jobs im stat. Zustand. $T = \lim_{t\to\infty} T_t$

Little's Law: Falls die obigen Grenzwerte für $N$, $\lambda$ und $T$ existieren, so gilt:

$$N = \lambda \cdot T$$

Die Formel von Little gilt auch für andere Serverstrategien als FCFS/FIFO.

Stochastische Variante von Little's Law:

$$\overline{N} = \lambda \cdot \overline{T} \quad \text{bzw.} \quad E[N] = \lambda \cdot E[T]$$

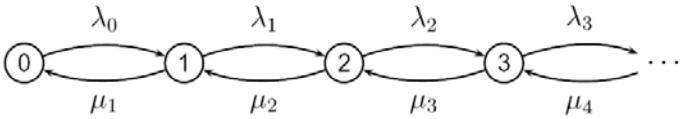
mit

- $E[N] = \sum_{i=1}^{\infty} i \cdot \pi_i$
- $\overline{N} = \lim_{t\to\infty} E[N(t)]$
- $\overline{T} = \lim_{i\to\infty} E[T_i]$
- $\lambda = \lim_{t\to\infty} \frac{E[\alpha(t)]}{t}$

Anwendungen von Little's Law: Siehe Skript 4/80ff

### 4.4.4 Birth-and-Death Prozesse

Skript 4/87ff
Ein Birth-and-Death Prozess ist eine Verallgemeinerung der Markov-Kette einer $M/M/1$-Warteschlange. Jeder Übergang kann nun eine eigene Ankunfts-/Bedienrate haben:




Der stationäre Zustand berechnet sich i.A. als:

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

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

Die Berechnung auf diese Weise ist aber sehr mühsam und sollte vermieden werden. Es gibt meistens einen Trick der die Berechnung vereinfacht.

Die W'keit, sich in einem Zustand $i$ zu befinden, ist:

$$p_i = p_0 \frac{\lambda_1 \cdot \lambda_2 \cdot \ldots \cdot \lambda_i}{\mu_1 \cdot \mu_2 \cdot \ldots \cdot \mu_i}$$

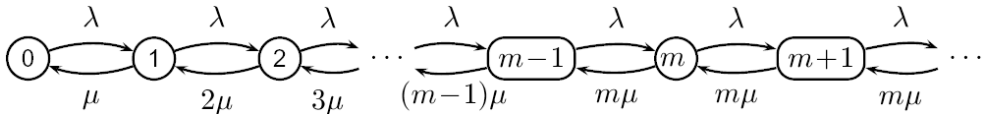
Obige Formel erhält man per Induktion aus: $p_0\lambda_1 = p_1\mu_1$.
Weiter gilt:

$$p_i(\mu_i + \lambda_i) = p_{i-1}\lambda_{i-1} \cdot p_{i+1}\mu_{i+1} \quad \text{(Knotenregel)}$$

Beispiele: siehe Skript 4/88ff

### 4.4.5 $M/M/m(/m)$-Systeme

Skript 4/90ff
$M/M/m$-Warteschlange ($m$...Anzahl Server)



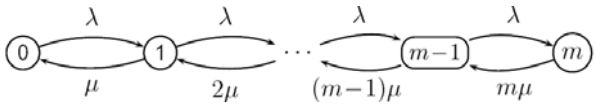

Berechnung des stationären Zustands: Siehe Skript 4/90.
$\pi_k, \pi_0$: Formeln siehe Skript 4/90.
$P_Q$: W'keit, dass ein ankommender Job in der Warteschlange warten muss:

$$P_Q = \frac{\pi_0(\rho m)^m}{m!(1-\rho)} = \frac{(\rho m)^m/(m!(1-\rho))}{\sum_{k=0}^{m-1}\frac{(\rho m)^k}{k!} + \frac{(\rho m)^m}{m!(1-\rho)}} \quad (\text{für } \rho < 1)$$

Obige Formel wird Erlang C-Formel genannt.

- Erwartete Anzahl Jobs in der Warteschlange: $N_Q = P_Q \frac{\rho}{1-\rho}$
- Mittlere Wartezeit in der Queue: $W = \frac{N_q}{\lambda} = P_Q \frac{\rho}{\lambda(1-\rho)}$
- Mittlere Antwortzeit: $T = W + \frac{1}{\mu} = \frac{\rho P_Q}{\lambda(1-\rho)} + \frac{1}{\mu} = \frac{P_Q}{m\mu-\lambda} + \frac{1}{\mu}$
- Mittlere Anzahl von Jobs im System: $N = \lambda T = \frac{\lambda P_Q}{m\mu-\lambda} + \frac{\lambda}{\mu} = \frac{\rho P_Q}{1-\rho} + m\rho$
- Bedingung für Gleichgewicht im System: $\rho = \frac{\lambda}{m\cdot\mu} < 1$

$M/M/m/m$-Warteschlange: System mit $m$ Servern und maximal $m$ Jobs im System:




Berechnung des stationären Zustands: Siehe Skript 4/95.
$\pi_k, \pi_0$: Formeln siehe Skript 4/95.
Blockierungswahrscheinlichkeit (W'keit, dass ein neu ankommender Job abgewiesen wird):

$$\pi_m = \frac{\left(\frac{\lambda}{\mu}\right)^m \frac{1}{m!}}{\sum_{k=0}^{m}\left(\frac{\lambda}{\mu}\right)^k \frac{1}{k!}}$$

Obige Formel wird Erlang B-Formel genannt. Die Erlang B-Formel gilt auch für $M/G/m/m$-Systeme.

### 4.4.6 Burke's Theorem

Skript 4/98
Gegeben ein $M/M/m$ System mit Ankunftsrate $\lambda$. Wir nehmen an, dass das System im stat. Zustand gestartet wird. Dann ist der Ausgangsprozess (d.h. der Prozess, der das System verlässt) wie der Eingangsprozess auch ein Poisson-Prozess mit Rate $\lambda$.

Dank Burke's Theorem können Warteschlangen-Netzwerke direkt analysiert werden.

## 5 Worst-Case Event Systems/Online-Theory

Bei der Worst Case Analyse wird immer davon ausgegangen, dass es einen „Gegner" gibt, der Events so eintreten lässt, dass der schlimmstmögliche Fall auftritt. Wir wissen nie, was der Gegner machen wird, aber er weiss alles. Online-Theory-Aufgaben sind meist Knobelaufgaben.

- Kosten eines Algorithmus: man berechnet die Kosten eines Algorithmus aus dem Worst-Case-Zug des Gegners. Bezeichnung: $cost_z(u)$ mit $z$...Laufvariable des Algorithmus und $u$...Spielzug des Gegners.

- Kompetitivität: Ein Online-Algorithmus A ist $c$-kompetitiv wenn für alle Eingabesequenzen $I$ gilt:

  $$cost_A(I) \leq c \cdot cost_{OPT}(I) + k$$

  bzw.

  $$c \cdot gain_A(I) \geq gain_{OPT}(I) + k$$

  Wenn $k = 0$ gilt, nennt man den Algorithmus strikt $c$-kompetitiv.
- Randomisierung: Der Gegner kann ausgetrickst werden, indem man im Algorithmus einen Zufallswert einfügt (Randomisierung). Ein randomisierter Algorithmus kann aber nie besser werden als ein deterministischer Algorithmus.

## 5.1 Lower Bounds

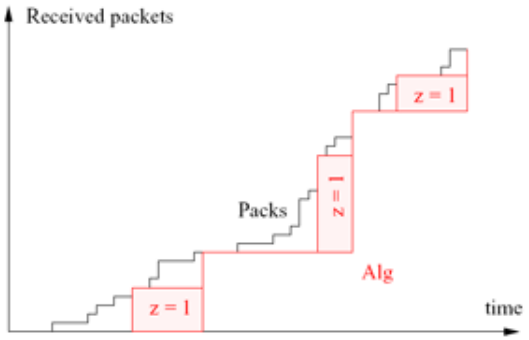
Skript 5/6f
Von Neumann/Yao Principle: Wenn für eine gegebene Verteilung eines Problems die minimalen Kosten $c$ betragen, so ist $c$ eine untere Schranke für den besten möglichen randomisierten Algorithmus.

## 5.2 Ausgewählte Probleme

Siehe Skript 5/7ff

### 5.2.1 TCP Acknowledgement Problem

Skript 5/7ff
Bei TCP können ACK's zurückgehalten werden bis zu einer maximalen Zeit und dann gleich mehrere Pakete gleichzeitig bestätigt werden. Das Problem ist, dass man nicht weiss ob in naher Zukunft noch Pakete kommen welche bestätigt werden müsssen oder ob man das ACK besser abschickt.
Der beste Algorithmus ist 2-kompetitiv und heisst z=1-Algorithmus: Immer wenn sich zwischen die beiden Kurven ein Rechteck mit Fläche 1 zeichnen lässt, sendet der Empfänger ein ACK welches alle vorangegangenen Pakete bestätigt:




### 5.2.2 TCP Congestion Control Problem

Skript 5/10ff
Congestion=Netzbelastung
Die Frage ist, nach welchem Algorithmus die Senderate erhöht werden muss um das Netz optimal auslasten zu können ohne es zu überladen. Die Netzkapazität ist hierbei Schwankungen unterworfen. Sofern angenommen wird, dass die Kanalkapazität konstant bleibt (Static Model), können folgende Algorithmen angewendet werden:

- AIMD: Standardmässig wird bei TCP AIMD (Additive Increase, Multiplicative Decrease) eingesetzt. Dabei wird die Rate schrittweise erhöht. Sobald die Kapazität überschritten wurde, wird die Rate um einen Faktor gesenkt und dann wieder schrittweise erhöht. Diese Methode ist sehr ineffizient. Benötigt $n$ Schritte und hat Kosten $\Theta(n^2)$.
- Binary Search: Benötigt $\log n$ Schritte bis die Senderate der Kapazität angepasst ist und hat Kosten $\Theta(n \log n)$.
- Shrink: Komplexer Algorithmus (siehe Skript 5/11). Hat Kosten $O(n \log \log n)$.

Wird eine sich ändernde Kanalkapazität angenommen (Dynamic Model), wird die Behandlung des Problems anspruchsvoller. Siehe dazu Skript 5/12ff.

# 6 Network Calculus / Adversarial Queuing Theory

Ziel: Worst-Case-Analyse (mit eingeschränktem Gegner) von Warteschlangen in Kommunikationsnetzwerken.
Bisher wurde immer angenommen, dass die Ankunftsrate in Wartschlangen-Systemen poissonverteilt sei. In der Realität stimmt das aber nicht. Deshalb wird Network calculus benutzt um solche Systeme besser beschreiben zu können.

- Arrival Curve: Normalerweise wird die leaky bucket-Version verwendet: $\alpha(u) = ru+b$ mit $r$...Senderate, $b$...Burst. Für Arrival Curves kann sub-additivity angenommen werden: $\alpha(s+t) \leq \alpha(s) + \alpha(t)$.
- Service Curve: Kurve mit welcher der Server die ankommenden Jobs abarbeitet. Folgende Bedingung muss erfüllt sein:
$$R^*(t) - R(t) \geq \beta(t-s)$$
mit $R^*$...Abgang und $R$...Ankunft
Es kommen verschiedene Varianten von $\beta$ zum Einsatz:
  - Constant rate server: $\beta(t) = ct$
  - Guaranteed-delay node: $\beta = \delta_T$
Der Server hält eine bestimmte garantierte Abarbeitungszeit ein (d.h. kein Job muss länger als diese Zeit in der Queue warten).
  - Internet Router
  - Tight Bounds on delay and backlog
- Min-Plus algebra: Algebra kennt Operationen Plus ($+$) und min ($\wedge$). Zudem gilt: $a + (b \wedge c) = (a+b) \wedge (a+c)$
Min-Plus-Faltung: $f \otimes g(t) = \inf_u \{f(t-u) + g(u)\}$
- Adversarial Queuing Theory: Das Ziel ist, Stabilität unter vorgegebenem Gegner zu studieren. Als Netzwerke werden gerichtete Graphen mit endlichen Warteschlangen angenommen. Zudem wird das Netz wie folgt eingeschränkt: alle Pakete haben die gleiche Länge, alle Links haben die gleiche Bandbreite, jeder Link kann pro Zeiteinheit maximal ein Paket befördern.
Der Gegner wird ebenfalls eingeschränkt, damit eine Analyse überhaupt möglich wird: Der Gegner ist durch zwei Parameter $(r, b)$ gebunden, wobei $r \leq 1$ und $b \geq 1$ den Parametern des leaky bucket Modells entsprechen.

  Stabilität: Eine scheduling policy ist stabil bei einer Rate $(r, b)$ in einem Netzwerk $G$ falls es eine Grenze $C(G, r, b)$ gibt sodass ein beliebiger $(r, b)$-Gegner es nicht schafft, mehr als $C(G, r, b)$ Pakete gleichzeitig ins Netzwerk zu speisen. Eine scheduling policy ist universell stabil falls sie stabil ist für jede Rate $r < 1$ in jedem beliebigen Netzwerk. Ein Netzwerk ist universell stabil wenn es stabil ist für jede Rate $r < 1$ mit jeder beliebigen greedy scheduling policy. Verschiedene scheduling policies: siehe Skript p. 6/35.

# Index