# THEORETISCHE INFORMATIK

## Sprachen und Automaten (~36%)

### Sprachen

| | | |
|---|---|---|
| **Alphabet Σ** (Sigma) | nicht leere endliche **Menge** (von Zeichen/Symbole)<br>Beispiel: $\{a,b,c,d\} = \Sigma_1$ $\{O\} =: \Sigma_2$ Nicht Beispiel: $\{0,1,2,3\ldots\}$ | Zeichen: $'a'$<br>Anzahl: $\lvert\Sigma\rvert$ |
| **Wort *w* /** String / Zeichenkette | über Alphatbet Σ, ist eine endliche **Folge** von Zeichen/Elementen aus Σ, evt. leer<br>Beispiel: $acdcd$ Wort über $\Sigma_1$ $0000$ Wort über $\Sigma_2$ | String: "$a$" oder "$ab$"<br>$'a' \neq$ "a" |
| **leeres Wort $\epsilon$** (epsilon) | über jedem Alphabet<br>$\epsilon$ nicht Element von Alphabet, sondern ein Meta-Symbol -> stehen lassen | Leerer String: ""<br>Length: $\lvert w\rvert$ |
| **$\Sigma^*$** | Menge aller Wörter über Σ | |
| **Konkatenation / Zusammenfügen** $'.'$ | von Wörter x und y über Σ:<br>Konkat (=Punkt) kann weggelassen werden<br>In Java +, In Haskell ++ | $x = x_1x_2..x_n, \quad x_i \in \Sigma_1, \quad n \geq 0$<br>$y = y_1y_2..y_n, \quad y_i \in \Sigma_2, \quad n \geq 0$<br>$xy = x \cdot y = x_1x_2 \ldots x_ny_1y_2 \ldots y_n$ |
| **Monoid** (Re-use) | Sei M eine Menge von $\circ$ $(Kringel)$: $M \times M \to M$<br>Operation: Das Paar $(M,\circ)$ heisst Monoid, falls gilt:<br>1. **Assoziativgesetz** $(a \circ b) \circ c = a \circ (b \circ c)$<br>$\forall(= Quantor)a,b,c \in M$<br>2. Es gibt ein **$\epsilon$(neutrales Element)** $\in M$ mit<br>$\epsilon \circ a = a = a \circ \epsilon \to \forall_a \in M$ | Beispiel: Natürliche Zahlen<br>$\mathbb{N} = \{0,1,2,3,\ldots\}$ $\mathbb{N}^+ = \{1,2,3\}$<br>1. $(\mathbb{N},+)$ (Addition) ist **ein** Monoid mit 0 neutral<br>2. $(\mathbb{N}_+,+)$ (Addition) ist **kein** Monoid, 0 fehlt<br>3. $(\mathbb{N}_+,*)$(Multiplikation) ist **ein** Monoid mit 1 neutral<br>4. $(\Sigma^*,\cdot)$( Konkatenation) ist **ein** Monoid mit $\epsilon$ neutral |
| **Potenzen** | Sei $(M,\circ)$ ein Monoid mit neutral $\epsilon$. Sei $a \in M$<br>$a^0 = \epsilon, \quad \emptyset^0 = \{\epsilon\}$ | $a^n = a \circ a^{n-1} = a \circ a \circ \ldots \circ a \circ \epsilon, \quad n \geq 1$<br>$0^31^5 = 00011111, \quad (01)^4 = 01010101$ |
| **Sprache über Σ** | Menge von Wörtern über Σ, also $\subseteq \Sigma^*$<br><br>Wort hat hier(bei uns) keinerlei Bedeutung.<br>Wir interessieren uns aber, ob ein Wort in einer Sprache enthalten ist.<br>$C = \{w \in \Sigma^*(ASCII) \lvert w\ von\ gcc\ akzeptiert\}$ | $\{\} = \emptyset$ // leere Sprache (kleinste)<br>$\{\epsilon\}$ // Sprache aus leerem Wort<br>$\Sigma^*$ // Menge aller Wörter (grösste)<br>$\{a, aa, ab, ac, ddd\}$ über $\Sigma_1$<br>$\{0,00,000,\ldots\}$ // codiert $\mathbb{N}^+$<br>$\{\epsilon,0,00,000,\ldots\}$ // codiert $\mathbb{N}$ |
| **Assoziativgesetz** | $(a \circ b) \circ c = a \circ (b \circ c)$ | |
| **Distributivgesetz** | $a * (b + c) = a * b + a * c$ und $(a + b) * c = a * c + b * c$ wobei $\forall a,b,c \in M$ | |
| **Kommutativges.** | $a \circ b = b \circ a$ | |
| **Operationen** | Sei $L_1, L_2$ Sprachen über Σ. | Beispiel: $\Sigma = \{a,b,..,z\},\ L_1 = \{a,b\},\ L_2 = \{c,d\}$ |
| Vereinigung | $L_1 \cup L_2$ | $L_1 \cup L_2 = \{a,b,c,d\}$<br>$L_1 \cup \epsilon = \{a,b,\epsilon\}$<br>$L_1 \cup \emptyset = L_1\ (neutrales\ Element)$ |
| Schnittmenge | $L_1 \cap L_2$ | $L_1 \cap L_2 = \emptyset$<br>$L_1 \cap \emptyset = \emptyset\ (vernichtendes\ Element)$ |
| Konkatenation | $L_1 \cdot L_2 = \{w_1 \cdot w_2 \lvert w_1 \in L_1, w_2 \in L_2\}$<br><br>Konkat auf Sprachen ist ein Monoid mit $\{\epsilon\}$ neutral -> $L^n$ also definiert | $L_1L_2 = \{ac, ad, bc, bd\}$<br>$L_2L_1 = \{ca, cb, da, db\}$<br>$L_1 \cdot \epsilon = L_1\ (neutrales\ Element)$<br>$L_1 \cdot \emptyset = \emptyset\ (vernichtendes\ Element)$ |
| Kleene'sche Stern | $L^* = \bigcup_{k=0}^{\infty} \Sigma^k = L^0 \cup L^1 \cup L^2 \cup \ldots \cup \ldots L^n$ | $L_1^* = \{\underbrace{\epsilon}_{L_1^0}, \underbrace{a,b}_{L_1^1}, \underbrace{aa,ab,ba,bb}_{L_1^2}, \ldots\}$<br>$\emptyset^* = \{\epsilon\}, \quad \emptyset^+ = \emptyset$ |
| ×: kart. Produkt | $L_1 \times L_2$ | $L_1 \times L_2 = \{(a,c),(a,d),(b,c),(b,d)\}$ |

**Idempotent Semirings S with 0 and 1**

$S = (M, + : M \times M \to M, 0 : M, * : M \times M \to M, 1 : M)$

Menge M, Addition wobei die kartesische Menge wieder die Menge M gibt, mit neutralem Element 0 aus M

Multiplikation wobei die kartesische Menge wieder die Menge M gibt, mit neutralem Element 1 aus M

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | $(a+b)+c = a+(b+c)$ | Assoziativgesetz + | semigroup + | abelian semigroup | abelian Monoid mit 0 | Semiring mit 0 und 1 | idempotent semiring with 0 and 1 |
| 2 | $a+b = b+a$ | Kummultativgesetz + | | | | | |
| 3 | $0+a = a = a+0$ | neutrales Element + | | | | | |
| 4 | $(a*b)*c = a*(b*c)$ | Assoziativgesetzt * | semigroup * | Monoid mit 1 | | | |
| 5 | $1*a = a = a*1$ | neutrales Element * | | | | | |
| 6 | $a*(b+c) = a*b+a*c$ | Distributivgesetz links | | | | | |
| 7 | $(a+b)*c = a*c+b*c$ | Distributivgesetz rechts | | | | | |
| 8 | $0*a = 0 = a*0$ | Absorbierung mit 0 | | | | | |
| 9 | $a+a = a$ | | | | | | |

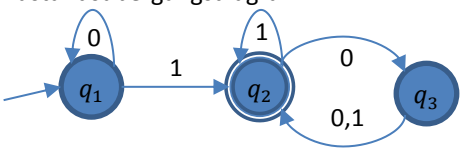
| **Vereinfachungen** | | |
|---|---|---|
| | $ab\epsilon = \epsilon ab$<br>$\epsilon\epsilon = \epsilon$ | $\epsilon \neq \{\epsilon\} \to genau\ nehmen$<br>$\lvert\{x,y\}\rvert = \begin{cases} 1, & falls\ x = y \\ 2, & sonst \end{cases}$ |

$$\prod_{i=1}^{n} x_i = \begin{cases} x_1 * x_2 * \ldots * x_n, & n \geq 1 \\ 1, & n = 0 \end{cases}$$

# Automaten

## DFA Deterministic Finite Automator

Zustandsübergangsdiagramm




Ein DFA ist ein 5-Tupel (5 wichtige Bestandteile, $Q, \Sigma, \delta, s, F$)

1. $Q$: endliche Menge von Zuständen $= \{q_1, q_2, q_3\}$
2. $\Sigma$: Eingabealphabet $= \{0,1\}$
3. $\delta: Q \times \Sigma \xrightarrow{total} Q: Transition$ siehe rechts
4. $s, q_0 \in Q$: Startzustand $= q_1$
5. $F \subseteq Q$: Menge akzeptierter Zustände $= \{q_2\}$

Eingabe: $1101 \in \Sigma^*$

1. Start: $q_1$
2. Lese **1**101, $q_1 \rightarrow q_2$
3. Lese 1**1**01, $q_2 \rightarrow q_2$
4. Lese 11**0**1, $q_2 \rightarrow q_3$
5. Lese 110**1**, $q_3 \rightarrow q_2 \rightarrow Endzustand$
6. Akzeptiere 1101, da $q_2$ akzeptierend

| $\delta$ | 0 | 1 |
|---|---|---|
| $q_1$ | $q_1$ | $q_2$ |
| $q_2$ | $q_3$ | $q_2$ |
| $q_3$ | $q_2$ | $q_2$ |

oder

$\delta(q_1, 0) = q_1$
$\delta(q_1, 1) = q_2$
...

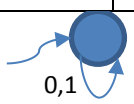
## akzeptieren / erkennen

M akzeptiert Wort w, falls:

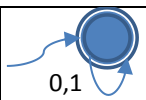
1. $q_0 = s$
2. $q_i = \delta(q_{i-1}, w_i), i = 1..n$
3. $q_n \in F$

M erkennt Sprache L, falls:

$$L = \{w \in \Sigma^* | M\ akzeptiert\ w\}$$




akzeptiert kein Wort
erkennt leere Sprache $\emptyset$




akzeptiert jedes Wort
erkennt $\Sigma^*$

## regulär

Eine Sprache heisst **regulär**, falls ein DFA existiert, der die Sprache erkennt.

## Unite DFAs $L_1 \cup L_2$

∪→∨
∩→∧

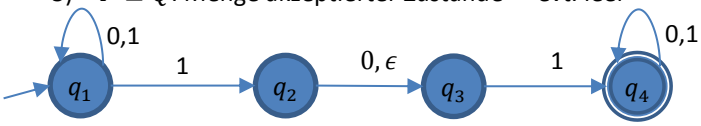
$$Q_\cup = Q_1 \times Q_2 = \{(q_1, q_2) | q_1 \in Q_1 \vee q_2 \in Q_2\}$$
$$F_\cup = \{(q_1, q_2) | q_1 \in F_1 \vee q_2 \in F_2\}$$
$$F_\cap = \{(q_1, q_2) | q_1 \in F_1 \wedge q_2 \in F_2\}$$
$$s = (s_1, s_2)$$
$$\delta_\cup((q_1, q_2), x) := (\delta_1(q_1, x), \delta_2(q_2, x))$$

$$Q_\cup = \{A, B, C\} \times \{1,2\} = \{A1, A2, B1, B2, C1, C2\}$$
$$F_1 = \{B\}; F_2 = \{2\} \rightarrow F_\cup = \{A\mathbf{2}, \mathbf{B}1, \mathbf{B}2, C\mathbf{2}\}$$
$$s_1 = A, \quad s_2 = 1, \quad s_\cup = A1$$
$$\delta_1(A, 0) = B$$
$$\delta_2(1, 0) = 2$$
$$\rightarrow \delta_\cup = ((A, 0), (1,0)) = B2$$

## NFA Non-deterministic Finite Automator

ein NFA ist ein 5 Tupel $(Q, \Sigma, \delta, s, F)$ mit

1) $Q$: endliche Menge von Zuständen
2) $\Sigma$: Eingabealphabet
3) $\delta: Q \times \Sigma_\epsilon \rightarrow P(Q)$ $(P = Potenzmenge)$ mit $\Sigma_\epsilon := \Sigma \cup \{\epsilon\}$
4) $s, q_0 \in Q$: Startzustand
5) $F \subseteq Q$: Menge akzeptierter Zustände -> evt. leer




Beispiel für $\delta$: Automat N

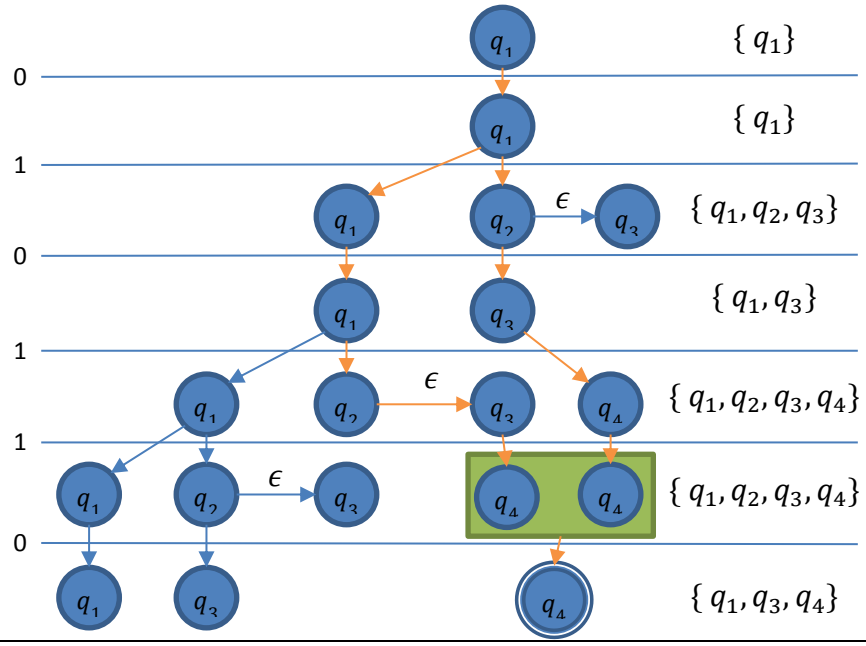
| $\delta$ | 0 | 1 | $\epsilon$ |
|---|---|---|---|
| $q_1$ | $\{q_1\}$ | $\{q_1, q_2\}$ | $\emptyset$ |
| $q_2$ | $\{q_3\}$ | $\emptyset$ | $\{q_3\}$ |
| $q_3$ | $\emptyset$ | $\{q_4\}$ | $\emptyset$ |
| $\{q_4\}$ | $\{q_4\}$ | $\{q_4\}$ | $\emptyset$ |

$\Sigma = \{0,1\}$

Unterschiede zum DFA

- $\epsilon$ Transitionen
- keine/eine/mehrere Transitionen

## Ausführung: 010110




010110 akzeptiert, da Weg von Start zu akzeptierenden Zustand existiert

Der NFA arbeitet wie folgt: Sobald der NFA mehrere Möglichkeiten anbietet, teilt sich die Maschine in die entsprechende Anzahl Kopien auf und verfolgt jede Möglichkeit mit einer Kopie. $\epsilon$-Transformationen ergeben ebenso eine Kopie. Wenn die Kopie nicht mehr weiterkommt stirbt sie. Sobald eine Kopie in einem akzeptierendem Zustand ist, akzeptiert der ganze NFA.

## Potenzmenge

von M ist die Menge aller Teilmengen von M, $Q = \{q_1, q_2, q_3\}$

$$\mathcal{P}(Q) = 2^Q = \{\emptyset, \{q_1\}, \{q_2\}, \{q_3\}, \{q_1 q_2\}, \{q_1 q_3\}, \{q_2 q_3\}, Q\}$$
$$|\mathcal{P}(Q)| = 2^{|Q|} = 2^3 = 8$$

$$|\mathcal{P}(M)| = |2^M| = 2^{|M|}$$
$$Q = a$$
$$2^{2^Q} = \{\emptyset, \{\emptyset\}, \{\{a\}\}, \{\emptyset, \{a\}\}$$

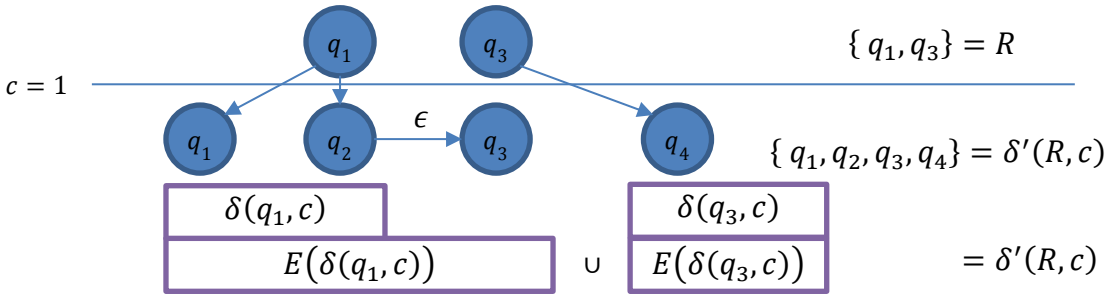
**Theorem** **Jeder NFA hat einen äquivalenten DFA.** Rabin und Scott, 1959, Turing-Award 1976

**Beispiel**

Sei $N = (Q, \Sigma, \delta, s, F)$ ein NFA, der $L$ erkennt

Wir **konstruieren** einen DFA $D = (Q', \Sigma, \delta', s', F')$ der ebenfalls L erkennt

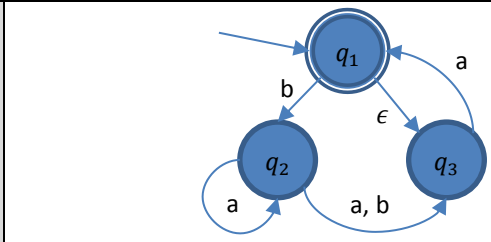
1. $Q' = 2^Q$
2. $\delta'(R, c) = \bigcup_{r \in R} E(\delta(r, c))$
3. $s' = E(\{s\})$
4. $F' = \{R \in Q' | R \cap F \neq \emptyset\}$ (R enthält mind. einen akzeptierenden Zustand von N)




Dabei ist $E(R) = \{q \in Q | q\ ist\ erreichbar\ von\ R\ mit\ 0\ oder\ mehr\ \epsilon - Transitionen\}, E(q_1, q_2, q_3) = \{q_3\}$

$$\delta'(\emptyset, c) = \bigcup_{r \in R} E(\delta(r, c)) = \emptyset, da\ neutral\ von\ \cup$$

**NFA ->**




$\Sigma = \{a, b\}$

$Q' = 2^Q$

$= \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$

$s' = \{1,3\} \rightarrow 3\ wegen\ \epsilon$

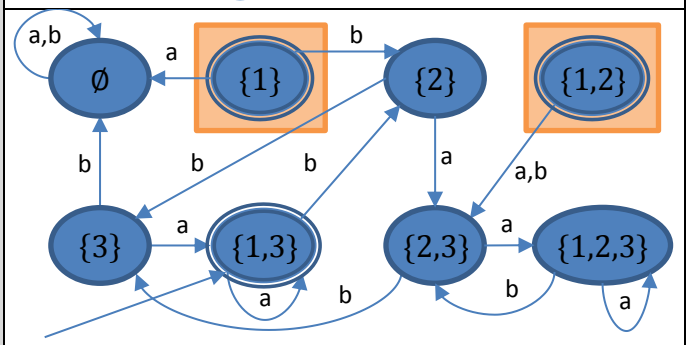
$F' = \{\{\mathbf{1}\}, \{\mathbf{1}, 2\}, \{\mathbf{1}, 3\}, \{\mathbf{1}, 2,3\}\}$

| **δ** | **a** | **b** | | **abbreviated** |
|---|---|---|---|---|
| {∅} | ∅ | ∅ | | |
| {1} | ∅ | {2} | | nicht erreichbar |
| {2} | {2,3} | {3} | | |
| {3} | {1,3} | ∅ | | |
| {1,2} | {2,3} | {2,3} | | nicht erreichbar |
| {1,3} | {1,3} | {2} | | |
| {2,3} | {1,2,3} | {3} | | |
| {1,2,3} | {1,2,3} | {2,3} | | |

$\bigcup_{r \in \emptyset} irgendwas = \emptyset$

NE = Nicht erreichbar

**DFA**




**Definition** Die Operationen $\cup$ $(union)$, $\cdot$ $(concat)$, $*$ $(Kleene'scher\ Stern)$ heissen **reguläre** Operationen

**Definition** Eine Menge M heisst **abgeschlossen** unter einer Operation $\circ$: $M \times M \rightarrow M$, falls für alle $x, y \in M$ gilt $x \circ y \in M$

**Bsp** $\mathbb{N} = \{0,1,2, \dots\}$

$(\mathbb{N}, +)$ ist abgeschlossen

$(\mathbb{N}, -)$ **nicht** abgeschlossen

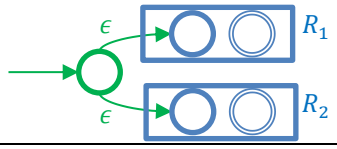
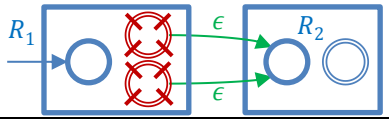
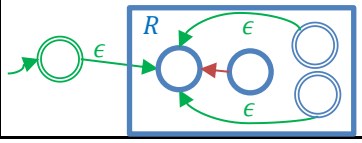
**Theorem** Die reguläre Sprachen sind abgeschlossen unter den regulären Operationen.

## Regular Expressions (Reguläre Ausdrücke)

**RE Regular Expressions**

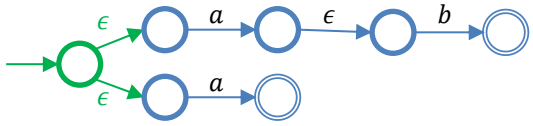
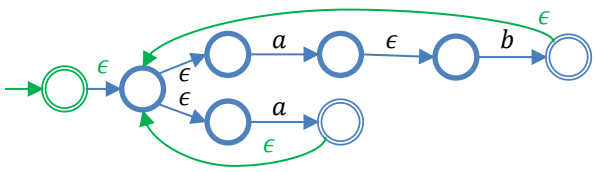
wachsende Priorität ↓

Sei R ein RE. Dann ist $\mathcal{L}[\![R]\!]$ die Sprache, die von R beschrieben wird.

| **Die sechs Formen** | **Syntax** | **Semantik** (quasi quotation) | **NFA** |
|---|---|---|---|
| 1. Zeichen aus dem Alphabet | $R = a,$ $a \in \Sigma$ | $\mathcal{L}[\![R]\!] = \{a\}$ **Wort mit genau 1 Symbol** |  |
| 2. Neutrales Element der Konkatenation | $R = \epsilon$ | $\mathcal{L}[\![R]\!] = \{\epsilon\}$ |  |
| 3. Neutrales Element der Vereinigung | $R = \emptyset$ | $\mathcal{L}[\![R]\!] = \emptyset$ |  |
| 4. Vereinigungsmenge $+ Normal \xrightarrow{\epsilon} Starts$ | $R = (R_1 \cup R_2)$ | $\mathcal{L}[\![R_1]\!] \cup \mathcal{L}[\![R_2]\!]$ (normale math. Vereinigung) |  |
| 5. Konkatenation $Akzept\ 1 \xrightarrow{\epsilon} Start\ 2$ $Akzept\ 1 \gg Normal$ | $R = (R_1 \cdot R_2)$ | $\mathcal{L}[\![R_1]\!] \cdot \mathcal{L}[\![R_2]\!]$ |  |
| 6. Klee'nescher Stern $+ Akzept$ $all\ Akzept \xrightarrow{\epsilon} Start$ | $R = (R^*)$ Syntak. Zucker: $R^+ \coloneqq RR^*$ (mind. einmal) | $(\mathcal{L}[\![R]\!])^*$ $L^* = L^0 \cup L^1 \cup L^2 \ldots$ $= \{\epsilon\} \cup L \cup LL \ldots$ $L^+ = L \cup LL \cup LLL \ldots$ |  |

**Bsp**

$\Sigma = \{a, b\}, \quad (ab \cup a)^*$

| | |
|---|---|
| $a \quad b$ |  |
| $ab$ |  |
| $ab \cup a$ |  |
| $(ab \cup a)^*$<br><br>nicht optimieren -> nicht Sinn -> erfolgt später |  |

**Satz** **Jede Sprache, die durch einen RE beschrieben wird, ist regulär. -> ein DFA existiert**

**Satz** **Zu jeder regulären Sprache gibt es einen äquivalenten Regulären Ausdruck.**
Beweisidee: DFA -> GNFA (Generaliced NFA) -> RE (nicht teil der Prüfung)

**Satz** **Minimierung**: Zu jedem DFA existiert ein äquivalenter DFA mit minimaler Anzahl Zustände.
Bis auf Benennung der Zustände eindeutig. Lässt sich konstruieren.

## Pumping-Lemma

**Goal**

Das Pumping Lemma wird verwendet, um zu beweisen, dass eine Sprache nicht regulär ist. Mittels Widerspruch.

**Pumping-Lemma**

**Annahme**: L **sei** regulär.
Dann gibt es nach unseren Pumping-Lemma ein $p \in \mathbb{N}$, so dass **jeder** String $\in L$ mit $|s| \geq p$ **p-pumpbar** ist.
Wir suchen und finden **einen** String $\in L$, der nicht **p-pumpbar** ist. -> L ist **nicht** regulär.

| s ist p-pumpbar, wenn<br>es **gibt** eine Zerlegung $s = xyz$ mit | s ist **nicht** p-pumpbar, wenn<br>für **jede** Zerlegung $s = xyz$ mit |
|---|---|
| a) $\|y\| \geq 1$<br>b) $\|xy\| \leq p$<br>c) $(\forall_i \in \mathbb{N} \mid xy^i z \in L),\ \mathbb{N} = \{\mathbf{0}, 1, 2, \dots\}$ | a) $\|y\| \geq 1$<br>b) $\|xy\| \leq p$<br>es gibt ein $i \geq 0$ mit $xy^i z \notin L$ |

**Anwendung**

L regulär ⇒ L kann gepumpt werden.

kontrapositiv: $(p \Rightarrow q) \equiv (\neg q \Rightarrow \neg p)$
L kann **nicht** gepumpt werden => L **nicht** regulär
~~L regulär <= L kann gepumpt werden.~~

**Beweis**

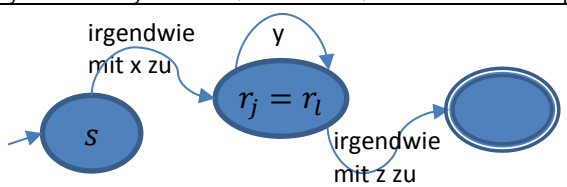
Sei $M = (Q, \Sigma, \delta, s, F)$ ein DFA, der L erkennt.
Sei p die Anzahl Zustände von M, also $p := |Q|$
Sei $w = s_1 s_2 \dots s_n \in L$ mit $s_i \in \Sigma$ und $n \geq p$
Sei $r_1, r_2, \dots, r_{n+1} \in F$ die Folge von Zuständen, durch die M läuft, um w zu akzeptieren:

$$s = r_1$$
$$w = r_1 s_1 r_2 s_2 r_3 s_3 r_4 \dots r_n s_n r_{n+1}$$
$$r_{i+1} = \delta(r_i, s_i), \qquad i = 1..n$$

Länge der Folge $r_i$: $(n+1) \geq (p+1), also\ (n+1) > p$
also $n + 1 > |Q|$. Weil M p Zustände hat wird mindestens ein Zustand irgendwann zweimal durchlaufen.
(Pigeonhole principle = Taubenschlag Prinzip) e.g. 10/7 gibt irgendwann ein Periode

Pigeonhole

$r_1 \quad r_2 \quad r_3 \quad \dots \quad r_j \quad \dots \quad r_l \quad \dots \quad r_p \quad r_{p+1} \quad \dots \quad r_{n+1}$
$s_1 \quad s_2 \quad s_3 \quad s_{j-1} \quad s_j \quad s_{l-1} \quad s_l \quad s_p \quad s_n$

| | | |
|---|---|---|
| $l \leq p + 1$<br>a. $\|y\| \geq 1$<br>b. $\|xy\| = l - 1 \leq p + 1 - 1 = p$<br>c. M akzeptiert $xy^i z$<br>für alle $i \in \mathbb{N} (mit\ 0)$ |  | $\|y\| \geq 1$<br>$\|xy\| \leq p$<br>$\|xyz\| \geq p = \|Q\|$ |

mehr formal

Sei L regulär über Σ, dann:

| | | | |
|---|---|---|---|
| $\exists (es\ existiert)\ p \in \mathbb{N}$ | $\forall_s \in L\ mit\ \|s\| \geq p$<br>$\exists\ x, y, z \in \Sigma^*\ mit\ s = xyz$ | a. $\|y\| \geq 1$<br>b. $\|xy\| \leq p$<br>c. $(\forall_i \in \mathbb{N} \mid xy^i z \in L)$ | **pump up** $i \geq 1$<br>**pump down** $i = 0$ |

**Beispiel**
E5P1

$$B = \{0^n 1^n | n \geq 0\} = \{\epsilon, 01, 0011, 000111, \dots\}$$

Intuition = ich muss mir merken wie viele 0 ich genommen habe und dann gleich viel 1 nemen -> kein Beweis

**Beweis 1**

Sei B regulär. Sei p eine Pumping-Länge.

$$p' := \begin{cases} p\ falls\ p\ gerade\ ist \\ p + 1\ falls\ g\ ungerade \end{cases}$$

Sei $\boldsymbol{s = 0^{\frac{p'}{2}} 1^{\frac{p'}{2}} \in B}$

$$|s| = \frac{p'}{2} + \frac{p'}{2} = p' \geq p$$

Also s zerlegen: $s = xyz\ mit\ Eigenschaften\ a, b, c$

| Variante 1 | Variante 2 | Variante 3 |
|---|---|---|
| $s = \underbrace{00}_{x}\ \underbrace{0\dots0}_{y}\ \underbrace{011\dots111}_{z}$ | $s = \underbrace{000\dots0011}_{x}\ \underbrace{1..1}_{y}\ \underbrace{111}_{z}$ | $s = \underbrace{000}_{x}\ \underbrace{0011}_{y}\ \underbrace{111}_{z}$ |
| $y\ enthält\ nur\ 0en$<br>$\rightarrow xy^2z \notin B$ | $y\ enthält\ nur\ 1en$<br>$\rightarrow xy^2z \notin B$ | $y\ enthält\ 0en\ und\ 1en$<br>Reihenfolge ist aber kaputt<br>$\rightarrow xy^2z \notin B$ |

keine weitere Zerlegungen -> nicht pumpbar

**Beweis 2**

Wähle

$$\boldsymbol{s = 0^p 1^p \in B}$$

$$|s| = 2p \geq p$$

Zerlegung $s = xyz$

| Variante 1 |
|---|
| $s = \underbrace{00}_{x}\ \underbrace{0\dots0}_{y}\ \underbrace{011\dots111}_{z}$ |
| $y\ enthält\ nur\ 0en$<br>$\rightarrow xy^2z \notin B$ |

keine weitere Zerlegungen, da $|xy| \leq p$

**Minimal Pumping Length**

Bezeichnet den kleinstmöglichen Wert für $p \in \mathbb{N}$, so dass das Wort beim Aufpumpen noch Teil der Sprache ist.

## CFG (Context Free Grammar)

**Kontextfreie Grammatik**

e.g. grösster gemeinsamer Teiler

```
while m != n do
  if m > n then
     m := m - n
  else
     n := n - m
  endif
endwhile
```

```
commands = while, if, :=
expression = '!=', '>', '-'

whileCmd -> while expr do cmd endwhile
ifCmd -> if expr then cmd else cmd endif
assignCmd -> ident := expr
cmd -> whileCmd
cmd -> ifCmd
cmd -> assiCmd
expr -> ...
```

**Beispiel**

A -> 0 A 1
A -> B
B -> #

nicht terminal symbole (NTS) / variable
terminal symbole (TS)
Produktionen
Startsymbol
Meta-Symbol: z.B. "kann die Form haben"

**CFG (Context free grammar)**

Eine CFG ist ein 4-Tupel $(V, \Sigma, P, S)$

$$V \cap \Sigma = \emptyset$$

1. $V$: Alphabet der NTS
2. $\Sigma$: Alphabet der TS
3. $P$: endliche Menge von Produktion $\in V \times (V \cup \Sigma)^*$
4. $S$: Startsymbol $\in V$

**Herleitung**

$$A \rightarrow 0A1$$
$$A \rightarrow B$$
$$B \rightarrow \#$$

$$A \Rightarrow 0A1$$
$$A \Rightarrow 00A11$$
$$A \Rightarrow 00B11$$
$$A \Rightarrow 00\#11 \in \Sigma^*$$

„=>“: leitet her
alles $\in (V \cup \Sigma)^*$
AA => BB // mehrere links ist ok

**Sprache einer CFG**

Sprache L(G) einer CFG G ist die Menge aller Wörter aus $\Sigma^*$, die vom Startsymbol hergeleitet werden können.

$$L(G_1) = \{0^i\#1^i | i \in \mathbb{N}\}$$

**Definition**

Sprache heisst **kontextfrei**, wenn sie durch eine kontextfreie Grammatik erzeugt werden kannn.

**Herleitung formal**

Seien $u, v, w \in (V \cup \Sigma)^*$, $"A \rightarrow w" \in P$
Dann $uAv \Rightarrow uwv$
u leitet v her, $u \overset{*}{\Rightarrow} v, falls\ u = v$
oder es gibt eine Folge

$$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_k \Rightarrow v$$

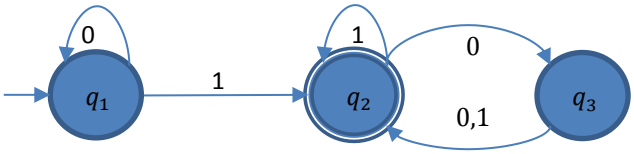
$$L(G) = \left\{ w \in \Sigma^* \middle| S \overset{*}{\Rightarrow} w \right\}$$

für $G = (V, \Sigma, P, S)$

einzelner Herleitungsschritt

**Satz**

**Jede reguläre Sprache ist kontextfrei**.
Beweis-Idee:




$$R_1 \rightarrow 0R_1$$
$$R_1 \rightarrow 1R_2$$
$$R_2 \rightarrow 0R_3$$
$$R_2 \rightarrow 1R_2$$
$$R_3 \rightarrow 0R_2$$
$$R_3 \rightarrow 1R_2$$
$$R_2 \rightarrow \epsilon$$

$$R_1 \Rightarrow 0R_1$$
$$\Rightarrow 01R_2$$
$$\Rightarrow 011R_2$$
$$\Rightarrow 0110R_3$$
$$\Rightarrow 01100R_2$$
$$\Rightarrow 01100\epsilon$$
$$\Rightarrow 01100$$

**Palindrom**

Wort von hinten = Wort von vorne

z.B. Anna

**PDA** Push Down Automat

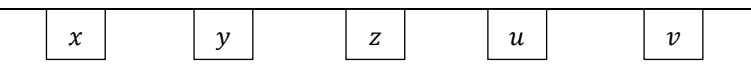
nicht deterministische PDA <=> CFG
determinischte PDA (schwächer) – teilmenge von CFG

$$\{a^i b^j c^k | i = j \vee i = k\}$$

Diese Sprache **braucht** Nichtdeterminismus inhärent!

$$S \rightarrow RC|T$$
$$R \rightarrow aRb|\epsilon$$
$$C \rightarrow cC|\epsilon$$
$$T \rightarrow aTc|B$$
$$B \rightarrow bB|\epsilon$$

Pumping-Lemma für CFG




$$x\,y^i\,z\,u^i\,v \in L, \quad \forall_i \in \mathbb{N}$$

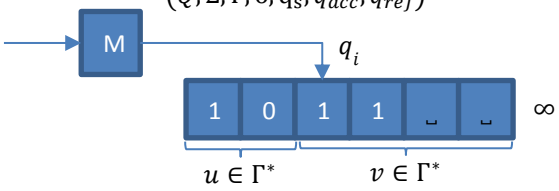
Pidgeonhole: endlich viele Produktionen.
Machen wir aus Zeitgründen nicht

# Turing Maschinen

## Turing Maschinen nach Alan Turing 1936

Turing-Maschine (TM) ist ein 7-Tupel

$$(Q, \Sigma, \Gamma, \delta, q_s, q_{acc}, q_{rej})$$




**length**: potentiell unendlich lang
**operationen**: read / write / shift left / shift right
**Schreibweise**: 1 0 q1 1
q ist vor dem lesendem, Kopf zeigt auf rechts von u
**Start** = Wort w von links auffüllen, Rest ist blank
$q_s$: zeigt auf erstes Element
**Bewegung** links/rechts: $\delta(q_i, 0) = (q_j, 1, L)$:
Spezialfall $u = \epsilon$ →Kopf bleibt am Anfang

1. Q: endliche Menge von Zuständen
   $Q = Q_C \cup Q_H$ $|Q| = c + 2$
   $Q_C = \{q_0, \dots, q_{c-1}\}$ $c(continue) \geq 0$
   $Q_H = \{q_{acc}, q_{rej}\}$
2. $\Sigma$ (Eingabealphabet)
   $\Sigma = \{\sigma_1, \dots, \sigma_s\}$ $s \geq 1$
3. $\Gamma$: Bandalphabet
   $\Gamma = \{\text{␣}\} \cup \Sigma \cup \Theta$ $|\Gamma| = 1 + s + t$
   $\Gamma = \{\gamma_0, \gamma_1, \dots, \gamma_t\}$
   ergänzende Zeichen $t \geq 0$
   $\Theta = \{\theta_1, \dots, \theta_t\}$
4. $\delta: Q_C \times \Gamma \xrightarrow{total} Q \times \Gamma \times \{L, R\}$
   // $q_{acc}, q_{rej}$ keine Nachfolgekonfiguration
5. $q_s \in Q$:Startzustand
6. $q_{acc} \in Q$: *akzeptierender Zustand*
7. $q_{rej} \in Q$: *verwerfender Zustand*
   $q_{acc} \neq q_{rej}$

## Encoding einer TM

$\Sigma_u := \{B, 0, 1, ', ', C, Q, A, J, D, L, R, W\}$

| Deklaration | Startzustand | Transition | Eingabewort |
|---|---|---|---|
| $\boldsymbol{B0^s1^t, C0^c,}$ | $\boldsymbol{Q(A\|J\|0^q,)}$ | $D\langle\delta_0\rangle\langle\delta_1\rangle \dots \langle\delta_{d-1}\rangle$ | $\boldsymbol{W\langle s_0\rangle, \dots, \langle s_{w-1}\rangle,}$ |
| $s = \|\Sigma\| \geq 1$ | $A$: *accept* | einzelne Transition: | $\langle s\rangle: 0^i$ |
| $t \geq 0$ | $J$: *reject* | $\langle\boldsymbol{\delta}\rangle : \mathbf{0}^i, \mathbf{0}^j, \boldsymbol{(A\|J\|0^k,)}\mathbf{0}^l, \boldsymbol{(L\|R)}$ | |
| $c = \|Q\| - 2 \geq 0$ | | beschreibt: | |
| | | $\delta(q_i, \gamma_j) = (q_{acc}\|q_{rej}\|q_k, \gamma_l, l\|r)$ | |
| Beispiel | Beispiel | Beispiel | Beispiel |
| $B0001, C0,$ | $Q00,$ | $00{,}0{,}000{,}00, L$ | $W00{,}0{,}000, codiert\ \sigma_2, \sigma_1, \sigma_3$ |
| $\rightarrow s = \|\Sigma\| = 3$ | $\rightarrow q_2$ | $codiert\ (q_2, \gamma_1) = (q_3, \gamma_2, left)$ | |
| $\rightarrow \Sigma = \{\sigma_1, \sigma_2, \sigma_3\}$ | | | |

consider: default transitions allowed (with blank to reject), multiple transitions allowed (first is taken), run word

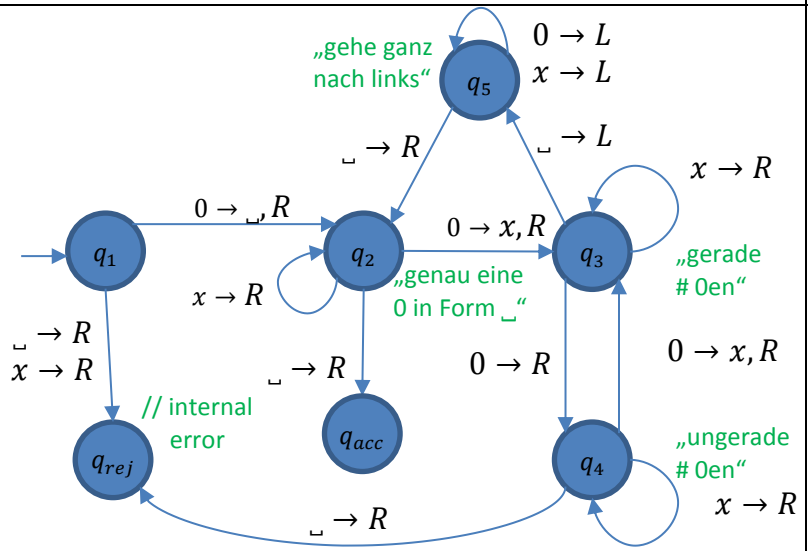
**Konfiguration** =Gesamtzustand der Maschine

| | |
|---|---|
| akzeptierende Konfiguration | $q = q_{acc}$ |
| verwerfende Konfiguration | $q = q_{rej}$ |
| haltende Konfiguration | $q = q_{acc} \vee q = q_{rej}$ |

Beispiel

$$A = \{0^{2^n} | n \geq 0\} = \{0{,}00{,}0000{,}00000000, \dots\}$$

| 1. accept | 2. reject |
|---|---|
| 00000000␣␣␣ | 000000␣␣␣ |
| ␣x0x0x0x | ␣x0x0x |
| ␣xxx0xxx | ␣xxx0x |
| ␣xxxxxxx | |




```
10 0 0 0 ␣ ␣
 ␣20 0 0 ␣ ␣
 ␣ x30 0 ␣ ␣
 ␣ x 040 ␣ ␣
 ␣ x 0 x3␣ ␣
 ␣ x 05x ␣ ␣
 ␣ x50 x ␣ ␣
 ␣5x 0 x ␣ ␣
5␣ x 0 x ␣ ␣
 ␣2x 0 x ␣ ␣
 ␣ x20 x ␣ ␣
 ␣ x x3x ␣ ␣
 ␣ x x x3␣ ␣
 ␣ x x5x ␣ ␣
...
5␣ x x x ␣ ␣
...
␣ x x x2␣ ␣
accept
```

## Berechenbarkeit / Entscheidbarkeit (~21%)

**TM akzeptieren / verwerfen**

TM M akzeptiert (oder verwirft) Eingabe w, falls eine Folge von Konfigurationen $C_1, C_2, \dots C_k$ existiert mit:

1. $C_1$ ist Start-Konfig für $w$
2. jedes $C_i$ liefert $C_{i+1}$
3. $C_k$ ist akzeptierend (oder verwerfend)

Sei $L(M)$ die von M erkannte (recognized) Sprache. Das ist die Menge aller von M akzeptierter Wörter.
Eine Sprache heisst **Turing-erkennbar** (oder rekursive aufzählbar), falls TM existiert, die die Sprache erkennt.

a) akzeptiert
b) verwirft
c) ∞-Schleife

**halten**

= akzeptieren oder verwerfen

**BlackBox**

Solange die Maschine läuft, wissen wir nicht, ob sie weiterläuft oder anhalten wird.
Falls Eingabe ∞-Schleife erzeugt, wissen wir **nie**, ob eine Endlosschleife vorliegt.

**Entscheider**

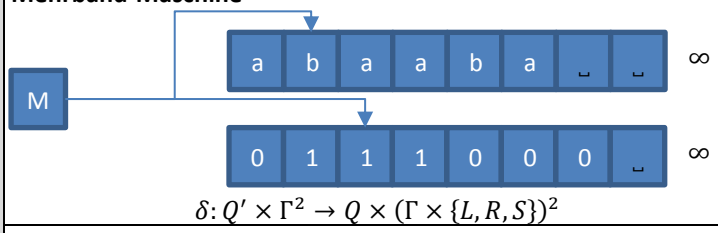
Besser: TM, die **immer** hält, für jede Eingabe. Diese heissen **Entscheider** (decider)

**entscheidbar**

Eine Sprache heisst **entscheidbar** (decidable), falls eine TM die Sprache erkennt und immer hält.

**Maschinen-Varianten**

**Mehrband-Maschine**




$\delta: Q' \times \Gamma^2 \to Q \times (\Gamma \times \{L, R, S\})^2$

Kann die mehr?
Nein wir können die auch schreiben als:
a b a a b a # 0 1 1 1 0 0 0 #
-> nicht drastisch langsamer

Alle Varianten von TM können exakt dasselbe: Jede lässt sich durch andere simulieren.
$\lambda$-Kalkül, while-schleifen, TM, etc. erkennen exakt diesselbe Sprache.
Jedes Modell kann jedes simulieren, die man bis heute gefunden hat.

**CHURCH-TURING-These** (kein Theorem)

informal: Intuitiver Begriff Algorithmus

Anno dazumal:
"Ein Prozess, durch den in endlich vielen Schritten bestimmt werden kann..."

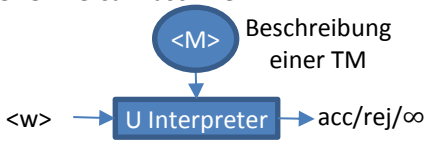
formal: $\lambda$-Kalkül Algorithmen ≡ TM Algorithmen ≡ viele weitere

**Axiom** der Informatik

**Universal-Maschine (U)**

**TM: Turing Maschine**


**U: Universalmaschine**




$U[\![\langle M \rangle \langle w \rangle]\!] = M[\![w]\!]$

**Akzeptanz-problem** $A_{TM}$

$$A_{TM} := \left\{ mw \in \Sigma_u^* \;\middle|\; \begin{array}{c} mw \in E(\langle tmw \rangle) \\ let(M,w) := dec[\![mw]\!] \\ M[\![w]\!] = acc \end{array} \right\}$$

$E(\langle tmw \rangle)$: Sprache aller legalen Kodierungen von einer TM und einem Wort
$mw$(Maschinenwort) $\in E$: Syntax check

**Halteproblem** $HALT_{TM}$

$$HALT_{TM} := \left\{ mw \in \Sigma_u^* \;\middle|\; \begin{array}{c} mw \in E(\langle tmw \rangle) \\ let(M,w) := dec[\![mw]\!] \\ M[\![w]\!] = acc \;\vee\; M[\![w]\!] = rej \end{array} \right\}$$

**Undefiniert ⊥ "bottom"**

$c(x) = 5$
$c(\bot) = \bot \to$ *in eager SML*
$c(\bot) = 5 \to$ *in lazy Haskel*

Nebenbemerkung: Man benutzt dieses 'bottom' auch für Programmiersprachen-Semantik.

**$TM_\Sigma$**

Sammlung aller TM, die auf Σ laufen.

**Funktionen: magic und flip**

```
magic: {acc,rej,⊥} -> {acc,rej}
magic(acc) = acc
magic(rej) = rej
magic(⊥)   = rej
```

```
flip: {acc,rej,⊥} -> {acc, rej, ⊥}
flip(acc) = rej
flip(rej) = acc
flip(⊥)   = ⊥
```

$U: \Sigma_u^* \xrightarrow{partiell} \{acc|rej\}, \qquad U: \Sigma_u^* \xrightarrow{total} \{acc|rej|\bot\}$

-> recognizer/Erkenner

**Spec Universal Maschine**

$$U[\![mw]\!] = rej \; if \; mw \notin E(\langle tmw \rangle)$$
$$otherwise$$
$$let(M,w) := dec[\![mw]\!]$$

| | | |
|---|---|---|
| $acc \; if \; M[\![w]\!] = acc$<br>$rej \; if \; M[\![w]\!] = rej$<br>$\infty \; if \; M[\![w]\!] = \infty$ | $acc \; if \; M[\![w]\!] = acc$<br>$rej \; if \; M[\![w]\!] = rej$<br>$\perp \; if \; M[\![w]\!] = \perp$ | $M[\![w]\!]$ |

$U[\![w]\!]$: Maschine laufen lassen mit $w$

partiell: sie muss nicht an aller Stelle definiert sein, kann aber. $Syntax: \nrightarrow \; oder \xrightarrow{partiell}$

2 und 3 Spalte: mathematisch exakt

**Satz**

U **erkennt** $A_{TM}$, aber **entscheidet** $A_{TM}$ **nicht**.

Beweis: Vergleich $A_{TM}$ mit U.

**Satz**

**U kann implementiert werden.**
Theorem: (U) sei $M \in TM_{\Sigma}, w \in \Sigma^*$.
Dann $U[\![\langle M \rangle \langle U \rangle]\!] = M[\![w]\!]$

Beweis: siehe 3-Band-Maschinen.

letzte Lektion

**Spec Acceptance Maschine**

$$A: \Sigma_u^* \xrightarrow{total} \{acc|rej\}$$

$$A[\![mw]\!] = rej \; if \; mw \notin E(\langle tmw \rangle)$$
$$otherwise$$
$$let(M,w) := dec[\![mw]\!]$$

| | | |
|---|---|---|
| $acc \; if \; M[\![w]\!] = acc$<br>$rej \; if \; M[\![w]\!] = rej$<br>$rej \; if \; M[\![w]\!] = \infty$ | $acc \; if \; M[\![w]\!] = acc$<br>$rej \; if \; M[\![w]\!] = rej$<br>$rej \; if \; M[\![w]\!] = \perp$ | $magic(M[\![w]\!])$ |

-> Entscheider

**Satz**

(A) Sei $mw \in \Sigma_u^*$. Dann $A[\![mw]\!] = magic(U[\![w]\!])$

$A[\![x]\!]: A \; angewandt \; auf \; x, A \; läuft \; auf \; x$

**Satz**

**A entscheidet $A_{TM}$**

Beweis: Vergleich $A_{TM}$ mit der Spec A.

**Satz**

A kann **nicht** implementiert werden.

Bew: Durch Diagonalisierung; folgt...

**Spec**

D (Diagonalmaschine) $D \in TM_{\Sigma_T}$
D: $\Sigma_T^* \xrightarrow{total} \{acc, rej\}$ $\quad D[\![m]\!] = flip(A[\![m\langle m \rangle]\!])$

// Entscheider

**$\Sigma_T$**

$\Sigma_u \backslash \{W\}$ zur Beschreibung von TMs

**Veranschaulichung**

D konstruiert $[\![m\langle m \rangle]\!]$ für gegebenes m.
Bsp:
B00000000000111,C...L: $m \in \Sigma_T^*$
81111111111122239...5
W00000000,0,0,...000000000,...00000,

| | |
|---|---|
| U: 0 -> 1 | ... |
| 1 -> 2 | B -> 8 |
| , -> 3 | C -> 9 |
| R -> 4 | ... |
| L -> 5 | D -> 11 |

**Beweis dass A nicht implementiert werden kann.**

$\langle x \rangle: Beschreibung \; von \; x$

| $\boldsymbol{U}$ | $\langle M_1 \rangle$ | $\langle M_2 \rangle$ | $\langle M_3 \rangle$ | ... | $\in \Sigma_T^*$ |
|---|---|---|---|---|---|
| $M_1$ | acc | rej | $\perp$ | | |
| $M_2$ | acc | acc | acc | | |
| $M_3$ | rej | rej | $\perp$ | | |
| ... | | | | | |
| $\in TM_{\Sigma_T}$ | | | | | |

Weil $D \in TM_{\Sigma_T}$, tritt in einer Zeile auf. D ist Entscheider.

| $\boldsymbol{A}$ | $\langle M_1 \rangle$ | $\langle M_2 \rangle$ | $\langle M_3 \rangle$ | ... | $\langle D \rangle$ | $\in \Sigma_T^*$ |
|---|---|---|---|---|---|---|
| $M_1$ | acc | rej | rej | | | |
| $M_2$ | acc | acc | acc | | | |
| $M_3$ | rej | rej | rej | | | |
| ... | | | | | | |
| $D$ | rej | rej | acc | | Problem da flip | |
| $\in TM_{\Sigma_T}$ | | | | | | |

Einträge

$$M_i[\![\langle M_j \rangle]\!] = U[\![\langle M_i \rangle \langle \langle M_j \rangle \rangle]\!]$$

$$magic(M_i[\![\langle M_j \rangle]\!]) = magic(U[\![\langle M_i \rangle \langle \langle M_j \rangle \rangle]\!]) = A[\![\langle M_i \rangle \langle \langle M_j \rangle \rangle]\!]$$

$$magic(D[\![\langle M_j \rangle]\!]) = [\![\langle M_j \rangle]\!]$$

$$D[\![\langle M_j \rangle]\!] = flip(A[\![\langle M_j \rangle \langle \langle M_j \rangle \rangle]\!] = flip\left(magic\left(\underbrace{M_j[\![\langle M_j \rangle]\!]}_{diagonaleinträge}\right)\right)$$

$$D[\![\langle D \rangle]\!] = flip(magic(D[\![\langle D \rangle]\!])) = flip(D[\![\langle D \rangle]\!])$$

Widerspruch in beiden Fällen acc und rej.
-> D kann nicht existieren. -> A existiert nicht.

Komplement $\bar{L}$

$$\bar{L} = \Sigma^* \backslash L$$

L quer ist sigma stern ohne L

**Satz**

**Falls L und $\bar{L}$ beide erkennbar sind, dann ist L entscheidbar.**
Beweis: Sei $M_L$ ein Erkenner für L, $M_{\bar{L}}$ Erkenner für $\bar{L}$.
Sei $w \in \Sigma^*$. Konstruiert TM M, die abwechselnd einen Schritt von $M_L$ und einen von $M_{\bar{L}}$.
Falls $w \in L, \rightarrow M_L$ in endlicher Zeit.
Falls $w \notin L, \rightarrow w \in \bar{L}, \rightarrow M_{\bar{L}}$ in endlicher Zeit.
-> M terminiert
Falls $M_L$ accept -> M accept
Falls $M_{\bar{L}}$ accept -> M reject

**Korollar (Folgesatz)**
**$\overline{A_{TM}} = \Sigma_u^* \backslash A_{TM}$ ist nicht erkennbar.**
Beweis:
Annahme: $\overline{A_{TM}}$ wäre erkennbar. $A_{TM}$ ist erkennbar, durch U.
-> $A_{TM}$ wäre entscheidbar. Aber es ist ja nicht.
$\rightarrow \overline{A_{TM}}$ nicht erkennbar.

**Theorem**

$HALT_{TM}$ ist undecidable.
Proof by reduction: reduce $A_{TM}$ to $HALT_{TM}$
That is: if $HALT_{TM}$ would be decidable, then would be $A_{TM}$
If I could solve $HALT_{TM}$, then I could run U with this and solve $A_{TM}$.

# Aussagenlogik

**Alphabet**

Ein Alphabet setzt sich aus folgenden Zeichen zusammen:
1. Aussagenvariablen $Var = \{p_1, p_2, \dots\}$
2. Logischen Junktoren $J = \{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$
3. Hilfszeichen $HZ = \{(,)\}$

¬ Negation
∧ Und-Verknüpfung (Konjunktion)
∨ Oder-Verknüfung (Disjunktion)
→ Wenn-Dann-Verknüpfung (Implikation)
↔ Genau-Dann-Verknüpfung (Äquivalenz)

**Sprache $L_{AL}$**

Context freie Grammatik

$$G := (\mathcal{N}, \mathcal{T}, \mathcal{R}, \langle formel \rangle$$
$$\mathcal{N} = \{\langle formel \rangle \langle variable \rangle\}$$
$$\mathcal{T} = \{\wedge, \vee, \neg, \rightarrow, \leftrightarrow (,), p_1, p_2, \dots p_n\}$$

$LA_n$:

$$L(G) = \{w \in \mathcal{T}^* | \langle formel \rangle \Rightarrow^* w\}$$

$LA_i$: Sprache der Aussagenlogik

$$L_{AL} = \bigcup_{i=1}^{\infty} LA_i$$

mit den Regeln

$$\langle variable \rangle \Rightarrow p_1$$
$$\langle variable \rangle \Rightarrow p_2$$
$$\dots$$
$$\langle variable \rangle \Rightarrow p_n$$
$$\langle formel \rangle \Rightarrow (\neg \langle formel \rangle)$$
$$\langle formel \rangle \Rightarrow (\langle formel \rangle \wedge \langle formel \rangle)$$
$$\langle formel \rangle \Rightarrow (\langle formel \rangle \vee \langle formel \rangle)$$
$$\langle formel \rangle \Rightarrow (\langle formel \rangle \rightarrow \langle formel \rangle)$$
$$\langle formel \rangle \Rightarrow (\langle formel \rangle \leftrightarrow \langle formel \rangle)$$

Startsymbol:

$$\langle formel \rangle$$

**Semantik der $L_{AL}$**

Belegung der Variablen

$$w: Var \rightarrow \{0,1\}$$
$$p \rightarrow w(p)$$

Beispiel:

$$w(p_1) = 1, w(p_2) = 0$$
$$p_1 \wedge p_2 = 0$$

**Wahrheits-tabellen**

| $p_1$ | $p_2$ | $\neg p_1$ | $p_1 \wedge p_2$ | $p_1 \vee p_2$ | $p_1 \rightarrow p_2$ | $p_1 \leftrightarrow p_2$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 |

Interpretation der Formeln unter Belegung w

$$I_w: L_{AL} \rightarrow \{0,1\}$$
$$\phi \rightarrow I_w(\phi)$$

$$\boldsymbol{p_1 \rightarrow p_2 \equiv \neg p_1 \vee p_2}$$

**Modell**

Eine Belegung $w$ heisst eine Modell für die Formel $\phi$, wenn $I_w(\phi) = 1$ ist, d.h. wenn die Formel unter der Belegung $w$ wahr wird.

Beispiel: $\phi = \Big(\big((p_1 \rightarrow (\neg p_2)) \vee p_3\big) \wedge \neg p_1\Big)$
$w(p_1) = 0, w(p_2) = w(p_3) = 1 \rightarrow I_w(\phi) = 1$

**Definitionen**

$\phi$ heisst **erfüllbar**, wenn ein Model für $\phi$ existiert.
$\phi$ heisst **falsifizierbar**, wenn ein Modell für $(\neg\phi)$ existert.
$\phi$ heisst **Tautologie**, wenn jede Belegung Modell für $\phi$ ist.
$\phi$ heisst **Kontradiktion**, wenn keine Belegung Modell für $\phi$ ist.

**äquivalent**

Zwei Formeln $\phi, \psi \in L_{AL}$ heissen (logisch) äquivalent, wenn sie unter allen Belegungen denselben Wert annehmen. Wir schreiben $\phi \approx \psi$, wenn $\phi$ und $\psi$ äquivalent sind.

**wichtige Äquivalenzen**

| | | |
|---|---|---|
| Kommutivität | $\phi \vee \psi \approx \psi \vee \phi$ | $\phi \wedge \psi \approx \psi \wedge \phi$ |
| Assoziativität | $\phi \vee (\psi \vee p) \approx (\phi \vee \psi) \vee p$ | $\phi \wedge (\psi \wedge p) \approx (\phi \wedge \psi) \wedge p$ |
| Distributivität | $\phi \vee (\psi \wedge p) \approx (\phi \vee \psi) \wedge (\phi \vee p)$ | $\phi \wedge (\psi \vee p) \approx (\phi \wedge \psi) \vee (\phi \wedge p)$ |
| de Morgan | $\neg(\phi \vee \psi) \approx (\neg\phi) \wedge (\neg\psi)$ | $\neg(\phi \wedge \psi) \approx (\neg\phi) \vee (\neg\psi)$ |
| Negation | $\big(\neg(\neg\phi)\big) \approx \phi$ | |

**Normalformen**

| | Bedingungen | Beispiel | Nicht-Beispiel |
|---|---|---|---|
| Negationsnormalform | nur ∨,∧ enhalten<br>¬ nur vor Variablen | $(\neg p_2) \wedge (\neg p_3) \vee p_4$ | $\neg(p_2 \vee p_3)$ |
| Konjunktive Normalform | Konjunktion von Klauseln | $K_1 \wedge K_2 \wedge K_3$<br>$(p_1 \vee p_2) \wedge (p_3 \vee \neg p_4)$ | $(p_1 \wedge p_2) \wedge (p_3 \wedge p_4)$ |
| Disjunktive Normalform | Literale mit ∧ verknüpft<br>danach mit ∨ | $L_1 \vee L_2 \vee L_3$<br>$(p_1 \wedge \neg p_2) \vee (\neg p_1 \wedge p_3)$ | $(p_1 \vee \neg p_2) \vee (p_1 \vee p_3)$ |

Satz: Jede Formel kann äquivalent in alle Normalformen umgeformt werden.

**Chomsky-Normalform**

Alle Regeln von der Form. $A \rightarrow BC, A \rightarrow b$. Entweder nur Terminal oder nur Nicht-Terminal-Symbole. Verschiedene in Oder '|' sind erlaubt. Max 2 Zeichen.

**Literale**

Eine Variable $p_1$ oder ihre Negation $(\neg p_1)$ nennt man auch Literal. Wir bezeichnen Literale mit $L_i$.

**Klauseln**

Werden Literale nur mit dem Junktor ∨ verknüpft, sprechen wir von Klauseln und bezeichnen sie mit $K_i$.

**Modelle**

Die Formel $\phi$ folgt logisch oder semantisch aus den Formeln $\psi_1, \psi_2, \dots \psi_n$, wenn gilt, dass alle Modelle der Formeln $\psi_1, \psi_2, \dots \psi_n$ auch Modelle der Formel $\phi$ sind. Wir schreiben dafür: $\psi_1, \psi_2, \dots, \psi_n \vDash \phi$.

**Vorgehen**:
1. Bestimme alle Variablen in den Formeln $\psi_1, \psi_2, \dots, \psi_n \; und \; \phi$
2. Bestimme alle Modelle für $\psi_1 \wedge \psi_2 \wedge \dots \wedge \psi_n$
3. Teste, ob alle diese Modelle Modelle für $\phi$ sind.

**Spezialfall**

Tautologie $\vDash \psi$

**Deduktionssatz**

Es gilt $\phi \vDash \psi$ genau dann, wenn $\vDash \phi \rightarrow \psi$ gilt. Wir müssen also zeigen dass $\phi \rightarrow \psi$ eine Tautologie ist.

# Komplexitätstheorie (KT) (~43%)

## Komplexität allgemein

**Algorithmische Probleme**

$f: \Sigma^* \to \Sigma^*$

Das allgemeine Modell war, dass eine Berechnung die schrittweise Umformung eines Eingabestrings in einen Ausgabestring ist, die durch eine endlich beschreibbare Maschine M ausgeführt wird.

a) Es gibt Zustände und die Berechnung ist eine Folge von Zustandsänderungen.
b) Die Maschine M kann ein Automat, Algorithmus, Grammatik oder Programm sein.

**Komplexitäts-theorie**

Nun geht es darum, den Aufwand und die Ressourcen für eine Berechnung zu bestimmen.
Uns interessiert nicht ein konkretes Problem (=Instanz), sondern die zusammengefasste Menge (=Problem).

**Problemvarianten**

1. **Variante: Optimierungsproblem** (e.g. min Knotenüberdeckung)
2. **Variante: Werteproblem** (e.g Anzahl Elemente für min Knotenüb.)
3. **Variante: EP: Entscheidungsproblem**
   e.g. Gibt es eine Knotenüberdeckung für G mit k Elementen, $\omega \in L$

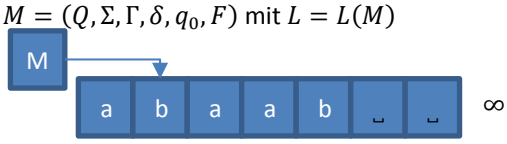
Wenn ich ein Oracle für 1 habe, dann kann ich 2 bestimmen.
Wenn ich ein Oracle für 2 habe, kann ich 3 bestimmen.

**Codierung**

wichtig, um in einer Turing Maschine eingeben zu können.

**Das 1-Band TM-Modell**

Gegeben: Alphabet $\Sigma\{0,1\}$ und Sprache $L \subset \Sigma^*$
Wir betrachten eine 1-Band TM
$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ mit $L = L(M)$




Unser Interesse gilt z.B. den Sprachen über Σ.
Wir möchten etwas über die Komplezität des EP (Entscheidungsproblem) einer Sprache wissen.

$TM \Rightarrow natürliche\ Kostenmasse$

| | | |
|---|---|---|
| $t_M$ | $\Sigma^* \to \mathbb{N}^*$ | // Ressource Zeit |
| | $\omega \to t_M(\omega)$ | Anzahl Schritte bis Halt |
| $s_M$ | $\Sigma^* \to \mathbb{N}^*$ | // Ressource Raum/Platz |
| | $\omega \to s_M(\omega)$ | Anzahl Zellen die die Maschine braucht bis Stopp // inklusive Platz für Wort $\omega$ |

Bemerkung 1: kleinste Änderung für "1Bit" ≡ 1Schritt
Bemerkung 2: TM sind "lokale" Modelle

**Bsp**

1: $L_1 = \Sigma^*$
2: $L_2 = \{0^k 1^k | k \in \mathbb{N}\}$
3: $L_3 = \{10,11,101,111,1011,1101, \dots\}$ (Primzahlen)

$L_1 \to reguläre\ Sprache \to O(n) = n$
$L_2 \to CFG \to O(n) = n^2$

**Komplezität Worst case complexity**

ist eine Funktion die den Aufwand misst in Abhängigkeit von der Inputlänge $n = |\omega|$.

$T_M: \mathbb{N} \to \mathbb{N}$
$n \to T_M(n) := max\{t_M(\omega) | \omega \in \Sigma^*, |\omega| = n\}$

$S_M: \mathbb{N} \to \mathbb{N}$
$n \to S_M(n) := max\{s_M(\omega) | \omega \in \Sigma^*, |\omega| = n\}$

**asympt. Kompl.**

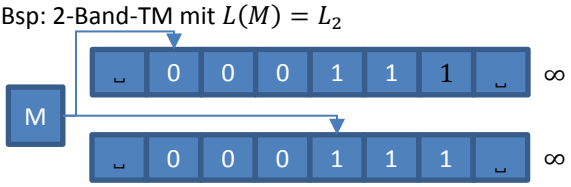
Wie verhält sich $T_M(n) \to ?\ für\ n \to \infty$

**Bsp $L_2$**

$T_M(n) = O(n^2)| \leq Cn^2\ für\ n \geq n_o$
Unter und obere Schranke (für gross n): $C_1 n \leq T_M(|\omega|) \leq C_2 n^2$

**Das k-Band TM-Modell**

Bsp: 2-Band-TM mit $L(M) = L_2$




1. Im zweiten Band erstes abschreiben - n
2. Zweite zurück - n
3. Gemeinsam zur Mitte – n

$3n = O(n)$

Problem: dies sagt uns etwas über die Maschine aus, jedoch nicht über den Algorithmus.

**Bsp $L_2$**

$T_{M2}(n) = O(n)$
$T_M(n) = \Omega(n)$
$T_{M2}(n) = \Theta(n)$

$L_2$ kann mit k-Band TM optimal in linearer Zeit entschieden werden.

**Komplexitäts-klassen**

Eine Komplexitätsklasse ist mit 4 Parameter bestimmt. **Was braucht wie wieviel von was?**

1) Berechnungsmodell (z.B. 1-Band, k-Band-TM, RAM, …)
2) Berechnungsmodus (z.B. deterministisch / nicht determ. / parallel)
3) Resource (z.B. Zeit / Platz)
4) Schranke (z.B. $\log n, n^k, n, n^2, exp^n$)

**Bsp**

$L_2$ gehört in die Klasse der in **quadratischer Zeit** mit einer **det. 1-Band TM** entscheidbaren Sprachen.

**$TM_\Sigma^1$**

Menge der deterministischen 1-Band Turing Maschinen über dem Alphabet Σ

**$TM_\Sigma$**

Menge der deterministischen Mehrband-Turing Maschinen über dem Alphabet Σ. $TM_\Sigma^1 \subset TM_\Sigma$

**$NTM_\Sigma$**

Menge der nicht deterministischen Mehrband Turing Maschinen über dem Alphabet Σ.

**DTIME DSPACE NTIME**

$DTIME_1(f(n)) := \{L \subset \Sigma^* | \exists M \in TM_\Sigma^1: L = L(M)\ und\ T_M(n) \leq O(f(n))\}$
$DSPACE_1(f(n)) := \{L \subset \Sigma^* | \exists M \in TM_\Sigma^1: L = L(M)\ und\ S_M(n) \leq O(f(n))\}$
$NTIME(f(n)) := \{L \subset \Sigma^* | \exists M \in NTM_\Sigma: L = L(M)\ und\ T_M(n) \leq O(f(n))\}$
$DTIME_1(f(t)) \subset DSPACE_1(f(t))$
$DTIME(n^k) \subset DTIME_1(n^{2k})$

$L_1 = 0^k 1^k$
$L_1 \in DTIME_1(n^2)$
$L_1 \in DTIME(n)$
$L_1 \in DSPACE_1(n)$
$L_2 = \{\epsilon, 01\} \to DTime_1(1)$
$L_3 = (01)^k \to DTime_1(n)$

**m-adische Darstellung**

$n = \underbrace{(51)_{10}}_{dezimal} = \underbrace{(110011)_2}_{binär} = \underbrace{1111111111\dots11111}_{unär}$

Anzahl Zeichen zur unären Darstellung:
$|n|_m = [\log_m(n)] + 1$

**non-determ. TM**

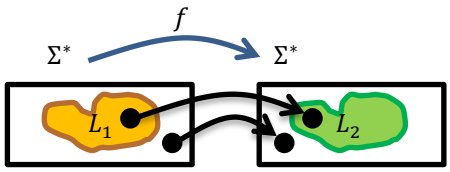
mehrere Berechnungspfade

$\delta: Q \times \Gamma \to P(Q \times \Gamma \times \{L, H, R\})$

## Komplexitätsklassen

| | |
|---|---|
| **Klasse P** | $P = \{L \subset \Sigma^* \mid \exists M \in TM_\Sigma \; und \; p(x) \; Polynom \; mit \; L = L(M) \; und \; T_M(n) = \mathcal{O}(p(n))\}$<br>**P ist die Klasse der in polynominaler Laufzeit, deterministisch entscheidbaren Sprachen.**<br>Es gibt eine TM die das Entscheidungsproblem in polynomialer Zeit löst. Ich kann eine Antwort geben. |
| **These von Cobham und Edmonds** | Wenn zwei universelle deterministische Berechnungsmodelle (mit logarithmischem Kostenmass) gegeben sind, dann existiert ein Polynom $p(n)$, so dass $t$ Schritte im ersten Modell durch $p(t)$ Schritte im zweiten Modell simuliert werden können. |
| **Klasse NP** | **Die Klasse der NP-Entscheidungsprobleme ist genau die Klasse, der in polynomialer Zeit mit einer nicht-deterministischen Turing-Maschine entscheidbaren Sprachen.**<br>Ich habe in vernünftiger Zeit keine Antwort, aber wenn ich die Antwort habe, kann ich diese verifizieren.<br>Lösung überprüfen geht linear. z.B: Faktorisierungsproblem liegt in NP als Funktionsproblem. |
| | Eine Sprache $L \subset \Sigma^*$ liegt in der Klasse NP wenn gilt: — $L \in NP: \Leftrightarrow$<br>Es existiert eine deterministische TM M mit polynominaler Laufzeit und eine Polynom p, — $\exists M \in TM_\Sigma, k \in \mathbb{N}^y \; mit \; T_M(n) = O(n^k)$ $\exists p(n) \in \mathbb{Z}[n]$<br>sodass für alle $x \in \Sigma^*$ gilt: — $\forall x \in \Sigma^*$:<br>Ein u, dass zu x gehört, nennen wir ein Zertifikat von x. u darf nur polynominal länger sein als x — $x \in L \Leftrightarrow \exists \, u \in \Sigma^* mit \; \lvert u\rvert \leq p(\lvert x\rvert)$ mit $M(x,u) = 1$ |
| | Klasse P ist eine Teilmenge von NP $P \subset NP, setzte \; u = \epsilon \; (leer)$ |
| **co-NP** | $\forall u \in \Sigma^*: \lvert u\rvert \leq p(\lvert x\rvert) \; mit \; M(x,u) = 0$<br>$co-NP := \{L \subset \Sigma^* \mid \bar{L} := \Sigma^* ohne \; L \in NP\}$<br>$coNP \not\subset NP$<br>$P \subset NP \cap coNP$ |
| **NP-hart** | Sprache L ist NP-hart genau dann wenn für alle Sprachen die in NP liegen gilt, dass diese reduzierbar auf L sind sind.<br>$\Leftrightarrow \forall \tilde{L} \in NP: \tilde{L} \leq_p L$<br>z.B. Faktorisierungsproblem liegt nicht in NP-hart |
| **NP-vollständig** | Wenn ich eines löse, haben ich alle anderen gelöst. Enthalten die schwierigsten EP in NP.<br>$\Leftrightarrow 1. \; L \; ist \; NP-hart,$ $2. L \in NP$ |

## Reduktion

| | | |
|---|---|---|
| **Reduktion in der KT** | Problem A kann auf Problem B zurückgeführt werden.<br>$A \leq B$ | Problem A mapping auf Problem B.<br>A ist in irgendeiner Form einfacher als B. |
| **Reduktionen** | **Turing Reduktion** (allgemeiner)<br>Gegeben: Zwei algorithmische Probleme A, B und ein Algorithmus $Alg_B$ um das Problem B zu lösen mit Laufzeit $T_B(n)$<br><br>Def: A ist Turing reduzierbar auf $B: \Leftrightarrow$<br>$\exists \; Alg_B$, der Problem B löst<br>$\exists \; Alg_A$, der A löst, mithilfe von Subprogram B<br>$\exists \; Polynome \; p(n), q(n), r(n)$ mit:<br>$T_A(n) \leq p(n) + q(n) * T_B(r(n))$<br>$p(n)$:Laufzeit von A ohne B<br>$q(n)$:Anzahl Aufrufe von $Alg_B$<br>$r(n)$: Abschätzung der Grösse des Inputs für $Alg_B$<br>Bemerkung: Relation | **Karp-Reduktion** (Spezifischer, polynom.-reduzierbar)<br>1-Schritt Reduktion: $q(n) = 1$<br>$\exists f: \Sigma^* \to \Sigma^*$<br>$mit \; \forall \omega \in \Sigma^*: \omega \in L_1 \Leftrightarrow f(\omega) \in L_2$<br>Es gibt eine polynominale TM<br><br>z.B. $L_1 = \{0^k 1^k \mid k \in \mathbb{N}\}, \; L_2 = \{0,1\}$<br>Die Funktion f sagt, das jedes Wort von $L_1$ auf $L_2$ abgebildet werden kann. $\omega = 0^k 1^k \to 0$<br>Jedes Wort ausserhalb von $L_1$ kann ich ausserhalb von $L_2$ abbilden. $\omega \in \Sigma^* ohne \; L_1 \to 00$<br>Beide sind jedoch polynominal, desshalb ist dieses Bsp uninteressant.<br>$L_2 \in \boldsymbol{P} \Rightarrow L_1 \in \boldsymbol{P}, \qquad L_1 \notin \boldsymbol{P} \Rightarrow L_2 \notin \boldsymbol{P}$ |
| **Notation** | $A \leq_T B \; oder \; A \leq_m B$ | $L_1 \leq_p L_2$ |
| reflex&transit | $\leq_T$ ist reflexiv und transitiv | $\leq_p$ist reflexiv und transitiv |
| äquivalent | $A =_T B,$ A und B sind Turing-äquivalent<br>$\Leftrightarrow A \leq_T B \; und \; B \leq_T A$ | $A =_p B$, A und B sind Karp-äquivalent<br>$\Leftrightarrow A \leq_p B \; und \; B \leq_p A$ |
| | Im Allgemeinen sind Werteproblem, Suchproblem und Entscheidungsproblem (EP) eines Problems Turing-äquivalent.<br>**Beispiel**<br>$L_1 = \{01, 0101, 010101, \dots\}$<br>$L_2 = \{0^{2k}(1110)^k \mid k \in \mathbb{N}^*\}$<br>$\omega \to \begin{cases} 001110, falls \; \omega \in L_1 \; ist \\ 101, sonst \end{cases}$ | Lemma Var $L_1 \leq_p L_2$<br>  <br><br>$\forall \omega \in \Sigma^*: \omega \in L_1 \Leftrightarrow f(\omega) \in L_2$ |
| Äquivalent | Optimierungsproblem<br>$CLIQUE =_T IS =_T VC$ | Entscheidungsproblem<br>$CLIQUE =_p IS =_p = VC$ |

## Reduktionen konkret

### $SAT \leq_p 3SAT$

$$\phi \in 3SAT \Leftrightarrow \phi \in SAT$$

Wir benützen den Satz von Cook-Levin: Es gilt $\forall L \in NP: L \leq_p SAT$

Wir müssen zeigen: $SAT \leq_p 3SAT, weil \leq_p$ transitiv ist, da $\forall L \in NP: L \leq_p SAT \leq_p 3SAT$

Satz: $3SAT \in NPC$

1) $3SAT \in NP$
2) $3SAT\ ist\ NP-hart$

zu zeigen: $\forall L \in NP: L \leq_p 3SAT$

| Vorgehen | Beweis |
|---|---|
| Sei $\phi = k_1 \vee k_2 \vee \dots \vee k_i$ ($k$ sind Klauseln)<br>Situation a) $k_i = l \quad \Rightarrow k_i = l \vee l \vee l$<br>Situation b) $k_i = l_1 \vee l_2 \quad \Rightarrow k_i = L_1 \vee L_2 \vee L_2$<br>Situation c) $k_i = l_1 \vee l_2 \vee \dots \vee l_n, n > 3$<br>$n-2$ neu Klauseln mit<br>$n-3$ zusätzlichen Variablen $y_{i1}, \dots, y_{i\,n-3}$<br>1 Klausel: $k_{i1} := l_1 \vee l_2 \vee y_{i1}$<br>2 Klausel: $k_{i2} := \neg y_{i1} \vee l_3 \vee y_{i2}$<br>3 Klausel: $k_{i3} := \neg y_{i2} \vee l_4 \vee y_{i3}$<br>... Klausel: $k_{i\,n-3} := \neg y_{i\,n-4} \vee l_{n-2} \vee y_{i\,n-3}$<br>... Klausel: $k_{i\,n-2} := \neg y_{i\,n-3} \vee l_{n-1} \vee l_n$ | **Behauptung**: Sei $\omega$ eine Belegung für die Variablen der Klausel $k_i$, dann gilt:<br>$I_\omega(k_i) = 1$<br>$\Leftrightarrow \exists$ Belegung für $y_{i1}, \dots, y_{i\,n-3}$ mit $I_\omega(k_{i1} \wedge \dots \wedge k_{i\,n-2}) = 1$<br>**Beweis**:<br>Fall 1: $l_1 = 1$ oder $l_2 = 1$ — Alle $y_{ij} = 0$<br>Fall 2: $l_{n-1} = 1$ oder $l_n = 1$ — Alle $y_{ij} = 1$<br>Fall 3: $l_j = 1$, $2 < j < n-1$ — $y_{i1} \dots y_{i\,j-2} = 1$, $y_{i\,j-1} \dots y_{i\,n-3} = 0$<br>Fall 4: $l_1 = l_2 = \dots = l_n = 0$ |

Diese Reduktion ist in höchstens quadratischer zeit-komplexität lösbar.

### $3SAT \leq_p SSS$

Bsp 2 SSS (SubsetSum-Problem) $\in NPC$

$$\forall L \in NP:$$

zu zeigen: $L \leq_p 3SAT \leq_p SSS$

$$\phi = \overbrace{(x_1 \vee x_2 \vee x_3)}^{c_1} \wedge \overbrace{(\overline{x_2} \vee x_3 \vee \overline{x_4})}^{c_2} \wedge \overbrace{(\overline{x_1} \vee x_2 \vee x_4)}^{c_3}$$

$$a_1, \dots, a; t, \qquad t = 1'111'333$$

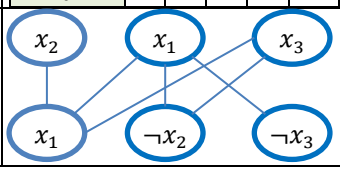
$\phi\ ist\ erfüllbar\ wenn\ y_1 z_1 \dots g_1 h_1 \dots t$ eine Lösung besitzt

$$y_1 = 1000\ 100$$
$$z_1 = 1000\ 001$$
$$\dots$$
$$\omega(x_1) = 1, \omega(x_2) = 1, \omega(x_3 = 1), \omega(x_4 = 0) \rightarrow I_\omega(\phi) = 1$$
$$\rightarrow y_1 + y_2 + y_3 + z_4 + g_2 + g_3 + h_3 = t$$
$$\dots$$
$$t = 1111\ 333$$
$$y = positiver(affirmativ)\ Wert$$
$$z = negativer\ Wert$$

| | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $c_1$ | $c_2$ | $c_3$ |
|---|---|---|---|---|---|---|---|
| $y_1$ | 1 | | | | 1 | | |
| $z_1$ | 1 | | | | | | 1 |
| $y_2$ | | 1 | | | 1 | | 1 |
| $z_2$ | | 1 | | | | 1 | |
| $y_3$ | | | 1 | | 1 | 1 | |
| $z_3$ | | | 1 | | | | |
| $y_4$ | | | | 1 | | | 1 |
| $z_4$ | | | | 1 | | 1 | |
| $g_1$ | 0 | | | | 1 | | |
| $h_1$ | | | | | 1 | | |
| $g_2$ | | | | | | 1 | |
| $h_2$ | | | | | | 1 | |
| $g_3$ | | | | | | | 1 |
| $h_3$ | | | | | | | 1 |
| $t$ | 1 | 1 | 1 | 1 | 3 | 3 | 3 |

### $3SAT \leq_p Clique$

Zeichne einen Graphen mit allen Klauseln an den Kanten.
Verbinde **keine** Literale in der **gleichen Klausel**.
Verbinde **keine** Literale mit **unterschiedlichen Werten**.



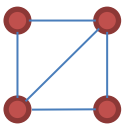

### $3SAT \leq_p kGC$

$$GF = GC$$

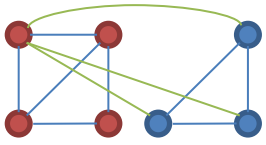
#### a) $3GC \leq_p kGC$ für $k \geq 3$

Sei $k > 5\ und\ r := k-3$,
G ein Graph $G \rightarrow \tilde{G} := G \dot{\cup} K_r$

$G$:
$k = 6$
$r = 3$



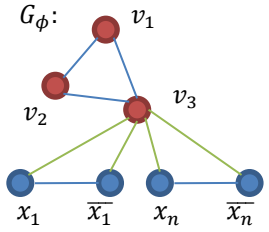
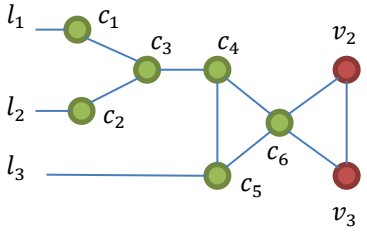
$\tilde{G} := G \dot{\cup} K_3$



grüne Linien zu allen roten Punkten.

#### b) $3SAT \leq_p 3GC$

$\phi \in 3KNF \rightarrow G_{\phi(V_\phi, E_\phi)}$

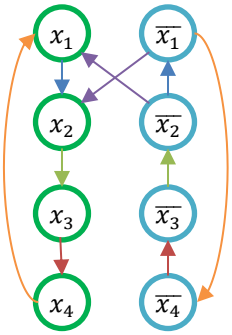
| **Variablen** (alle an $v_3$) | **Klauseln** (alle zu $v_2 - v_3$) |
|---|---|
| $Anzahl\ \phi$: $n$: $x_1 .. x_n$ | $Anzahl\ \phi$: $m$: $c_1 \dots c_m$ |
|  |  |





$$|V_\phi| = 3 + 2n + 6m$$
$$F(v_1) = 1\ (für\ wahr\ 1)$$
$$F(v_2) = 2\ (für\ falsch\ 0)$$
$$F(v_3) = 3\ (weder\ wahr\ noch\ falsch)$$

Eine Graphkomponente ist genau dann 3-färbbar, wenn mindestens eines der Literale $l_1, l_2, l_3$ den Wert 1 trägt.

| | |
|---|---|
| **$2SAT \in P$**<br>→ gerichteter Graph | **Beweis**:$\phi \in 2KNF \rightarrow \phi = k_1 \wedge k_2 \wedge \ldots \wedge k_r \; mit \; k_i = (l_{i_1} \vee l_{i_2})$<br>**Bemerkung**: $l_{i_1} \vee l_{i_2} \approx \neg l_{i_1} \rightarrow l_{i_2} \approx \neg l_{i_2} \rightarrow l_{i_1}$<br>$V_\phi$ enthält je einen Knoten für $x_i$ und $\overline{x_i}$<br>Behauptung: $\phi \in 2SAT \Leftrightarrow \neg\left(\exists\, x_i : \left(x_i \overset{*}{\Rightarrow} \overline{x_i}\right) \wedge \left(\overline{x_i} \overset{*}{\Rightarrow} x_i\right)\right)$ (hin und zurück)<br>Beweis: $\exists Belegung \; \omega \; mit \; I_\omega(\phi) = 1, d.h. x_i \rightarrow \cdots x_j, \rightarrow x_n \rightarrow \cdots \rightarrow \overline{x_i}$<br>**Bsp**: $\phi = (x_1 \vee x_2) \wedge (\overline{x_3} \vee x_4) \wedge (\overline{x_1} \vee x_2) \wedge (x_1 \vee \overline{x_4}) \wedge (x_3 \vee \overline{x_2})$<br>Sei $\omega(x_i) = 1 \rightarrow \omega(\overline{x_i}) = 1 \rightarrow falsch$<br>Beweis: Konstruktion von $\omega$.<br>Wähle einen Knoten $x_i$ der noch nicht belegt ist mit der Eigenschaft dass es keinen Weg gibt von $x_i$ nach $\overline{x_i}$. $\neg\left(\exists x_i \overset{*}{\rightarrow} \overline{x_i}\right)$<br>starke Komponente von einem Punkt = jeder andere Punkt erreichbar |  <br>Algorithmus von Valiant, Kosaroju, Gabow |
| **$2GC \in P$** | Beweis: Ich kann irgendwo beginnen und die Farben abwechselnd anzeigen, oder ich kann alles umkehren, bekomme jedoch wieder den selben Wiederspruch. Greedy-Algorithmus | |
| **$k-d-GC$** | k-Färbbarkeit eingeschränkt auf Graphen $G = (V, E)$ mit maximalem Grad $\max\{\deg(v) \mid v \in V\} \leq d\}$<br>Die Probleme k-d-GC mit k>d sind trivial! Da wir mindestens soviel Farben haben, wie wir benötigen.<br>k: Anzahl Farben<br>d: Anzahl Kanten pro Knoten<br>Bem: $kGC_{planar}$ ist für $k \geq 4$ trivial.<br>Es gilt: $3GC \leq_p 3GC_{planar} \leq_p 3\,4\; GC_{planar}$ | |
| **Graphen** | Ebener Graph: Zeichnen ohne sich die Kanten überschneiden.<br>Planarer Graph: Graph verzerren bis sich die Kanten nicht mehr überschneiden. | |
| **$SSS \in NPC$** | 1. Subset-Sum in NPC<br>$a_1, a_2, \ldots, a_n; t$<br>2,3, …,1102<br>Algorithmus für SSS<br>Array mit boolean values, length t+1 | |

| | 0 | 1 | 2 | 3 | | | | t |
|---|---|---|---|---|---|---|---|---|
| 1. Schritt<br>1, danach 0en | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2. Schritt<br>$for\; i \in \{1..n\}$ | 1 | 0 | 1<br>$a_1$ | 1<br>$a_2$ | 0 | 1<br>$a_1 + a_2$ | 0 | |

Am Schluss schaue ich ob t 1 oder 0 ist.

$$\#Schritte \; = n * t$$

Pseudo-nominaler Algorithmus
Schwach NP-vollständiges Problem: Nur kleine Zahlen, dann ist es polynominal.

## ofach öpis

| | |
|---|---|
| **aSLP: Algebraic straight-line program**<br>ohne loops<br>ohne if's | Sei $F$ ein Körper (oder ein Ring)<br>Ein aSLP der Länge $r$ über $F$ mit Input $I = \{x_1, \ldots, x_n\}$ und built-in constants $C = \{c_1, \ldots, c_r\} \subset F$<br>ist eine Folge von Anweisungen der Form $y_i = z_{i_1} op\; z_{i_2}, i = 1, \ldots, r$ mit $z_{i_1}, z_{i_2} \in I \cup C \cup \{y_j \mid j < i\}$<br>und $op \in O \subset \{+, *\}$. Der Output ist $y_r$.<br>**Bsp**: AK: $c_1 = 1, O = \{+\}$ |

## Übungen

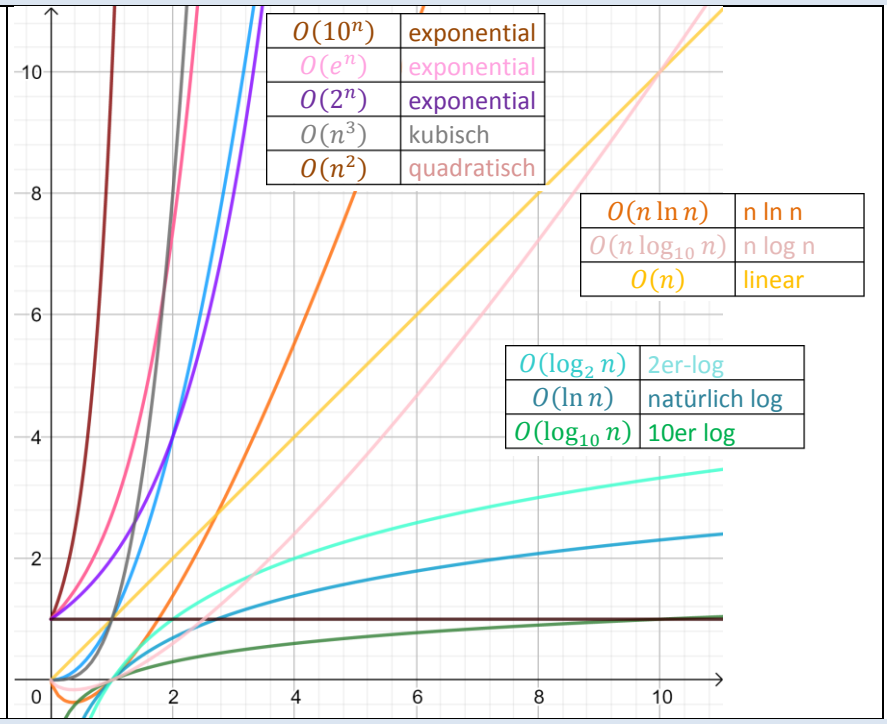
| | | |
|---|---|---|
| **Algebra** | Taylor-entwicklung<br>$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \cdots \rightarrow e^x > \frac{x^{n+1}}{(n+1)!}$ | General<br>$x^a = e^{a \log x}$ |
| | $\mid \; \mid = Kardinalität = Betrag$<br>$\mathbb{N} = \{0,1,2, \ldots\}$<br>$\mathbb{N}^* = \{1,2,3, \ldots\}$ | Stirling Formel<br>$\sqrt{2\pi}\, n^{n+\frac{1}{2}} e^{-n} < n! \leq \sqrt{2\pi}\, n^{n+\frac{1}{2}} e^{-n+\frac{1}{4n}}$ |

## Big-O-Notation

**Anzahl ausführungen abhängig von n**

$$n^2 + 3n + 5 = \mathcal{O}(n^2)$$
$$n \log n = \mathcal{O}(n^2 \log \log n)$$
$$n^{1000} = \mathcal{O}(e^n)$$
$$n^2 \log n = \mathcal{O}(n^{2+\epsilon})$$

$\mathcal{O}(n)$ = Obergrenze
$\Omega(n)$ = Untergrenze
$\Theta(n)$ = Ober- und Untergrenze




## Approximations-klassen

**Notation**

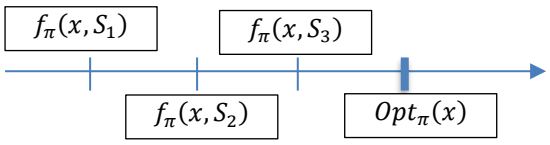
$$\pi: Optimierungsproblem$$
$$I_\pi: Menge\ der\ Instanzen\ von\ \pi$$
$$x \in I_\pi: S(x)\ Lösungen\ des\ Problems$$
$$opt_\pi(x) = \begin{Bmatrix} \min\big(f_\pi(x,s), s \in S(x)\big), falls\ \pi\ Minimum \\ \max\big(f_\pi(x,s), s \in S(x)\big), falls\ \pi\ Maximum \end{Bmatrix}$$

**Definition**

Gegeben: $\pi$

Ein polynomialer Algorithmus A heisst $\alpha$-Approximationsalgorithmus für $\pi$.

$$\Leftrightarrow 1) \forall x \in I_\pi: A(x) \in S(x)$$
$$2) \exists \alpha \in [1, \infty]:$$
$$Opt_\pi(x) \leq \alpha f_\pi\big(x, A(x)\big), falls\ \pi\ ein\ Max$$
$$f_\pi\big(x, A(x)\big) \leq \alpha Opt_\pi(x), falls\ \pi\ ein\ Min$$




Beispiel: ich wähle $\alpha$ 2, dann ist z.B. nur $S_2$ und $S_3$ einen möglichen Algorithmus.

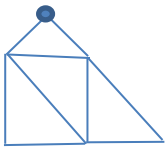
**Def**

$$APX(\alpha) = Optimierungsproblem\ für\ der\ ein\ \alpha - Approx - Alg\ existiert.$$

$$APX = \bigcup_{\alpha \geq 1} APX(\alpha) \qquad APX^* = \bigcap_{\alpha \geq 1} APX(\alpha)$$

**Bsp (VC)**

Gegeben:



Greedy 1
Ich wähle einen Knoten und streiche die Knoten, usw … -> eher schlecht
Greedy 2
Ich sortiere zuerst nach k (Anzahl Kanten), danach gehe ich von hoch k zu tief k.
Beides sind keine alpha-Approx. Algs
Minmal:

$$\frac{f_\pi\big(x, A(x)\big)}{Opt_\pi(x)} \leq \alpha$$

Matching-Alg (2-Approx Alg)
Ich wähle irgendeine Kante und notiere mir die Knoten, danach wähle ich eine zweite Kante die ein Matching ist zur ersten Kante.
Sei $mC$ ein minimaler Cover:

$$|mC| \leq |VC| \leq 2|mC|$$

Beweis: Erste Ungleichung, mC ist das optimum welches kleiner oder gleich ist.
Zweite Ungleichung: Ich brauche mindestens die Hälfte einer Kante, da sie sonst nicht abgedeckt ist.