VORWORT

partially yoinked from Jasmin Fässler , Nina Grässli & Jannis Tschan and Lukas Hunziker. Motivated by Ian MacKaye

VORGEHEN ZU AUFGABENTYPEN

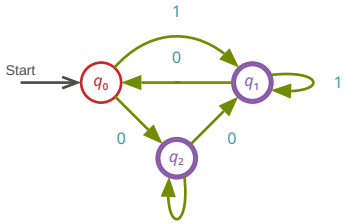
Frage	*Antwort*
Ist Sprache regulär?	Ja: DEA / NEA / RegEx Nein: Pumping Lemma
Ist Sprache kontextfrei?	Ja: Stackautomat / CNF / BNF Nein: Pumping Lemma (CFL)
Turing-Maschinen	Berechnungsgeschichte
Ist eine Sprache Turing-Vollständig?	Ja: TM-Simulation Nein: Halteproblem
Kann Problem von einem Computer gelöst werden?	Reduktion auf Halteproblem
Kann ein Programm entscheiden, ob zulässige Inputs eine Eigenschaft erfüllen?	Satz von Rice
Kann eine nichtdeterministische Maschine … entscheiden?	NP-Verifizierer
Warum kann ein Problem nicht gelöst werden?	Reduktion auf NP-Vollständiges Problem

DETERMINISTISCHE ENDLICHE AUTOMATEN (DEA)

$A = (Q, \Sigma, \delta, q, F)$

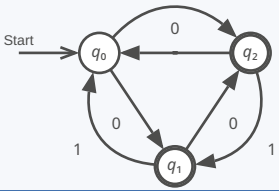
- Zustände: $Q = \{q_0, q_1, ..., q_n\}$
- Alphabet: Σ
- Übergangsfunktion: $\delta : Q \times \Sigma \to Q$
- Startzustand: $q \in Q$
- Akzeptierzustände: $F \subset Q$

		0	1	
Q	q_0	q_2	q_1	
F	q_1	q_0	q_1	δ
	q_2	q_1	q_2	

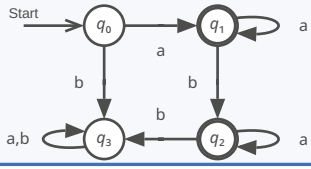


Beispiel 1: $L = \{w \in \{0,1\}^* \mid |w|_0 - |w|_1 \equiv 0 \bmod 2\}$

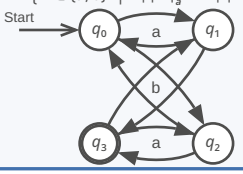
Beispiel 2: $L = \{w \in \{0,1\}^* \mid |w|_0 \neq |w|_1 \bmod 3\}$



Beispiel 3: a+ba+



Beispiel 4: $L = \{w \in \{a,b\}^* \mid 2 \mid |w|_a \wedge 2 \nmid |w|_b\}$



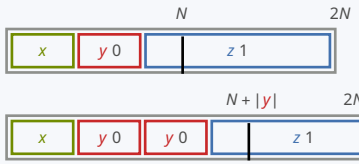
PUMPING LEMMA (DEA)

Um zu beweisen, dass eine Sprache **nicht** regulär ist

Vorgehen:
1) Annahme: L ist regulär
2) Gemäss Pumping Lemma gibt es die Pumping Length N
   – Es darf keine Annahme über die konkrete Grösse von N gemacht werden!
3) Wähle ein Wort $w \in L$ mit $|w| \geq N$
   – Die Definition **muss** N verwenden!
4) Aufteilung des Wortes gemäss Pumping Lemma
   – $w = xyz$, $|xy| \leq N$, $|y| > 0$
5) Auswirkung des Pumpens
   – $xy^kz \notin L$ für mindestens ein $k \in \mathbb{N}$ (mit Begründung!)
6) Widerspruch und Schlussfolgerung, dass die Annahme nicht zutreffen kann

Beispiel 5: $L = \{0^n1^n \mid n \geq 0\}$ **ist nicht regulär**

1) Annahme: L ist regulär
2) $\exists N \in \mathbb{N}$, Pumping Length
3) $w = 0^N1^N$
4) Unterteilung $w = xyz$



5) Pumpen: nur die Anzahl der 1 wird erhöht, Anzahl 1 bleibt
6) $xy^kz \notin L$ für $k \neq 1$, Widerspruch zum Pumping Lemma

Beispiel 6: $L =$ **alle korrekte Integralausdrücke**

1) Annahme: L ist regulär
2) $\exists N \in \mathbb{N}$, Pumping Length
3) $w = \int_{a1}^{b1}\int_{a2}^{b2}\dots\int_{aN}^{bN} dx_N \dots dx_2\, dx_1 \in L$
4) Unterteile $w = xyz$, wobei $|y| \geq 1 \wedge |xy| \leq N \wedge xy^kz \in L$
5) Pumpen: Der Teil xy enthält keine dx_k, beim pumpen nimmt also nur die Anzahl der Integralzeichen zu, so dass kein korrekter Integralausdruck stehen bleibt.
6) Der Widerspruch zeigt, dass die Annahme, L sei regulär, nicht haltbar ist. Also ist L nicht regulär.

---

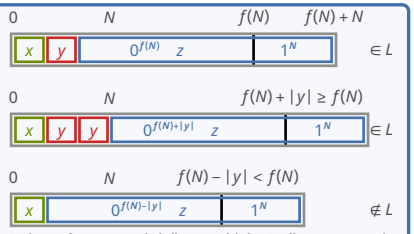
6 Schritte des Pumping-Lemma-Beweises: Annahme (A) 1 Punkt, Pumping Length (N) 1 Punkt, Wort (W) 1 Punkt, Unterteilung (U) 1 Punkt, Widerspruch beim Pumpen (P) 1 Punkt, Folgerung (F) 1 Punkt.

Beispiel 7: $L = w \in \{0,1\}^* \mid |w|_0 \geq f(|w|_1)$

Sei $f : \mathbb{N} \to \mathbb{N}$ eine streng monoton wachsende Funktion. Eine solche Funktion nimmt beliebig grosse Werte an und es gilt $f(n) \geq n$ für alle $n \in \mathbb{N}$.

---

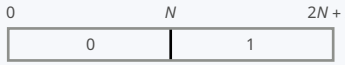
1) Annahme: L ist regulär.
2) Nach dem Pumping Lemma gibts die Pumping Length N
3) Wir wählen das Wort $0^{f(N)}1^N \in L$.
4) Nach dem Pumping Lemma lässt sich das Wort w in drei Teile $w = xyz$ zerlegen mit $|xy| \leq N$ und $|y| > 0$. Weil $f(N) \geq N$ enthält y ausschliesslich Nullen.



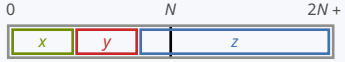
5) Beim Aufpumpen wird die Anzahl der Nullen grösser, das bedeutet aber nicht, dass das Wort nicht mehr in der Sprache enthalten ist, da nur $|w|_0 \geq f(|w|_1)$ verlangt ist. Beim Abpumpen wird allerdings die Anzahl der Nullen kleiner, xz enthält nur noch $f(N) - |y|$ Nullen. Damit haben wir $|xz|_0 = f(N) - |y| < f(N) = |xz|_1$, das Wort xz ist also nicht mehr in L, im Widerspruch zur Aussage des Pumping Lemmas.
6) Der Widerspruch zeigt, dass Annahme, L sie regulär, nicht haltbar ist. Also ist L nicht regulär.

Beispiel 8: $L = \{w \in \Sigma^* \mid |w|_0 - |w|_1 = \pm 2\}$, $\Sigma = \{0,1\}$

Wähle Wort:



Unterteile:



Da $|y|$ nur aus Nullen bestehen kann, hat das Wort $v = xz$, aus dem der Teil $|y|$ abgepumpt worden ist, weniger Nullen. Die Anzahl der Nullen ist daher mindestens um 3 geringer als die Anzahl der Einsen, das abgepumpte Wort v ist nicht mehr in der Sprache L: $v \notin L$.

Beispiel 9:

Betrachten Sie eine Sprache L über dem Alphabet $\Sigma = \{1\}$. Die Wörter sind durch ihre Länge vollständig beschrieben, sie können also der Länge nach sortiert werden. Zusätzlich sei bekannt, dass jedes Wort $w \in L$ weniger als halb so lang ist wie das nächstlängere Wort in L. Für das längere Wort w_1 gilt also $|w| < {}^1/_2|w_1|$ oder gleichbedeutend $2|w| < |w_1|$. Warum ist diese Sprache nicht regulär?

Nein, wie man mit dem Pumping-Lemma zeigen kann.
1) Wir nehmen an, die Sprache L sei regulär.
2) Nach dem Pumping-Lemma gibt es die Pumping Length N
3) Wir wählen das kürzeste Wort $w \in L$, welches Länge $|w| \geq N$ hat.
4) Nach dem Pumping-Lemma lässt sich das Wort in drei Teile xyz aufteilen, die natürlich alle aus 1 besteht.
5) Beim Pumpen wird das Wort um den Teil y länger, also um eine Länge $|y| \leq N$. Die Länge des gepumpten Wortes ist daher $|w| + |y| \leq 2|w|$. In der Beschreibung des Wortes steht aber auch, dass das gepumpte Wort eine Länge $> 2|w|$ haben muss, ein Widerspruch.
6) Dieser Widerspruch zeigt, dass die Sprache L nicht regulär sein kann.

---

6 Schritte des Pumping-Lemma-Beweises: Annahme (A) 1 Punkt, Pumping Length (N) 1 Punkt, Wort (W) 1 Punkt, Unterteilung (U) 1 Punkt, Widerspruch beim Pumpen (P) 1 Punkt, Folgerung (F) 1 Punkt.

NICHTDETERM. ENDLICHE AUTOMATEN (NEA)

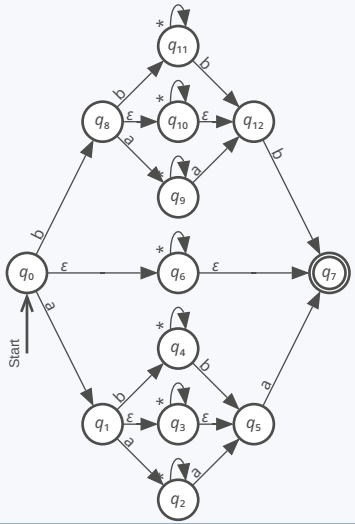
$A = (Q, \Sigma, \delta, q, F)$

- Endliche Menge von Zuständen: Q
- Alphabet: Σ
- Übergangsfunktion: $\delta : Q \times \Sigma \to P(Q)$
- Startzustand: $q \in Q$
- Akzeptierzustände: $F \subset Q$

Beispiel 10: $L = \{wxw^t \mid w, x \in \Sigma^* \wedge |w| \leq 2\}, \Sigma = \{a, b\}$

Das Wort w^t ist die gespiegelte Version des Wortes w. Ist die Sprache L regulär?

Ja, alle diese Beweise sind gültig:
1) $w = \varepsilon$ darf sein und daher kann $w = x \in \Sigma^*$ sein. Es folgt $L = \Sigma^*$, diese Sprache ist also regulär
2) Regex: `aa.*aa|ab.*ba|ba.*ab|bb.*bb|a.*a|b.*b|.*`
3) NEA:



VERALLGEMEINERTER NEA (VNEA)

Begriff	*Definition*
Regulär	Es gibt einen DEA A, der L akzeptiert, also $L(A) = L$
Regulärer Ausdruck	Zeichenkette r zur Beschreibung einer regulären Sprache $L = L(r)$
Reguläre Operationen	$L(r_1) \cup L(r_2) = L(r_1 \mid r_2)$ $L(r_1)L(r_2) = L(r_1r_2)$ $L(r_1)* = L(r_1*)$
Verallgemeinerter NEA	NEA_ε, dessen Übergänge mit regulären Ausdrücken beschriftet sind

PRIMITIVE REGULÄRE AUSDRÜCKE

Beispiel 11: **Prüfungsfrage**

```
/RewriteCond %{HTTP_USER_AGENT} Zeus.*?Webster/
```

Warum ist das Fragezeichen in diesem regulären Ausdruck unnötig?

Das Fragezeichen macht den Teil `.*` optional, aber dieser Teil kann durchaus das leere Wort sein. Der Webmaster hat offenbar nicht daran gedacht, dass die $*$-Operation auch das leere Wort beinhaltet.

Reguläre Ausdrücke für Wörter mit Länge ≤ 1

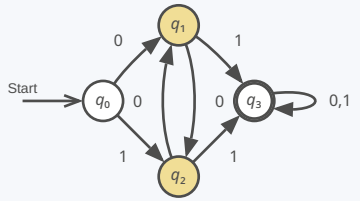
$L = L(r)$	r	NEA
$\varnothing$	$\varnothing$	Start →
$\{\varepsilon\}$	$\{\varepsilon\}$	Start → ◎
$\{a\}$	a	Start → ○ —a→ ◎
$\{o, s, t\}$	$[ost]$	Start → ○ —o,s,t→ ◎
$\{a, b, ..., s\}$	$[a-s]$	Start → ○ —[a-s]→ ◎
Σ	.	Start → ○ —Σ→ ◎

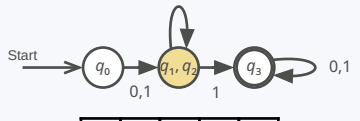
MINIMALAUTOMAT

Ziel: Nicht unterscheidbare Zustände zusammenlegen

Vorgehen:
1) Akzeptierzustand ≠ Nichtakzeptierzustand
2) Iteration: alle unterscheidbaren Paare suchen
3) Verbleibende Paare sind ununterscheidbar → zusammenlegen

Beispiel 12:





	q_0	q_1	q_2	q_3
q_0	≡	×	×	×
q_1	×	≡	≡	×
q_2	×	≡	≡	×
q_3	×	×	×	≡

Algorithmus

1) $q \in F, q' \notin F$
2) Unterscheidbar nach Übergang
3) Iteration bis keine Änderung mehr

⇒ q_1 und q_2 sind nicht unterscheidbar

KONTEXTFREIE SPRACHEN (CFL)

Kontextfrei: L kann nur von einem ND PDA erkannt werden

$$G = (V, \Sigma, R, S)$$

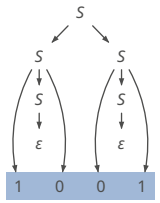
- V: Variablen
- Σ: Terminalsymbole (Alphabet)
- R: Regeln der Form $A \to x_1 x_2 ... x_n$ mit $A \in V, x_i \in V \cup \Sigma$
- S: Startvariable

Parse Tree

$L = \{w \in \{0,1\}^* \mid |w|_0 = |w|_1\}$

mit der Grammatik

$S \to 0S1 \mid 1S0 \mid SS \mid \varepsilon$



$\in L$

Eine kontextfreie Grammatik G ist dann ***eindeutig***, wenn für jedes Wort aus $L(G)$ genau eine mögliche Ableitung aus dem Startsymbol existiert.

Beispiel 13: **Ist L kontextfrei?**

$$L = \{wuw^t \mid w, u \in \Sigma^* \wedge |u| \leq 3\}, \Sigma = \{a, b\}$$

(w^t ist das gespiegelte Wort von w)

Die Sprache ist kontextfrei, weil die folgende kontextfreie Grammatik die Sprache erzeugen kann:

$$S \to aSa \mid bSb \mid U$$
$$U \to \varepsilon \mid A \mid AA \mid AAA$$
$$A \to a \mid b$$

Alternativ kann man auch einen Stackautomaten angeben, der die Sprache akzeptiert.

Grammatik für A (A) 1 Punkt, Schachtelungsidee (S) 2 Punkt, der mittlere Teil kann leer sein (U) 1 Punkt, der mittlere Teil kann aus a und b bestehen (B) 1 Punkt, Schlussfolgerung kontextfrei (K) 1 Punkt.

PUMPING LEMMA (CFL)

Um zu beweisen, dass eine Sprache **nicht** kontextfrei ist

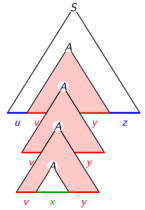
Voraussetzungen:
1) $|vy| > 0$
2) $|vxy| \leq N$
3) $\forall k \in \mathbb{N}. (uv^k xy^k z \in L)$

Grammatik G in CNF
$w \in L(G) \Rightarrow S \overset{*}{\Rightarrow} w$

Wiederverwendete Variable
$|w| \geq N$ gross genug $\Rightarrow$ Variablen werden im Parse Tree wiederverwendet
Die "unterste" wiederverwendete Variable A erzeugt zwei Wörter:

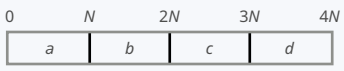
$A \overset{*}{\Rightarrow} vxy$
$A \overset{*}{\Rightarrow} x$

Pumpen
$A \overset{*}{\Rightarrow} vxy$ anstelle des "untersten" $A \overset{*}{\Rightarrow} x$ verwenden

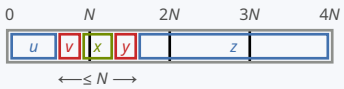


Beispiel 14: $L = \{a^i b^j c^k d^l \mid i = k \wedge j = l\}$

Wähle Wort:



Unterteile:



Wegen $|vxy| \leq N$ können v und y Zeichen aus höchstens zwei der vier Blöcke enthalten. Beim Pumpen werden sich daher die Anzahlen von zwei benachbarten Arten von Zeichen ändern, nicht aber der anderen Zeichen, die in gleicher Zahl vorhanden sein sollten. Damit ist ein gepumptes Wort nicht mehr in L.

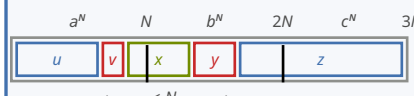
Beispiel 15: $L = \{w \in \{0,1\}^* \mid w = 0^k 1 0^l 1 0^k 1 0^l\}$

1) Annahme: L ist kontextfrei
2) Nach dem Pumping Lemma $\exists N \in \mathbb{N}$, Pumping Length
3) Wähle das Wort $w = 0^N 1 0^N 1 0^N$
4) Nach dem Pumping Lemma gibt es eine Aufteilung von w in fünf Teile $w = uvxyz$ derart, dass $|vxy| \leq N \wedge |vy| \geq 1$. Ausserdem ist jedes gepumpte Wort $uv^k xy^k z \in L$.
5) Da sich die Anzahl der Einsen beim Pumpen nicht ändern darf, müssen v und y vollständig in einem Nullen-Block enthalten sein. Daher kann sich nur die Anzahl der Nullen in höchstens zwei Nullen-Blöcken ändern. Die beiden Blöcke müssen wegen $|vxy| \leq N$ ausserdem benachbart sein.
   Zum ersten Nullen-Block gehört der dritte, der gleich viele Nullen enthalten muss, zum zweiten gehört der vierte. Wie auch immer die beiden Blöcke gewählt werden, ändert sich die Anzahl Nullen in den Blöcken, aber nicht in den zugehörigen Blöcken. Das gepumpte Wort kann also nicht mehr in L sein.
6) Dieser Widerspruch zeigt, dass die Annahme, L sei kontextfrei, nicht haltbar ist. Also ist L nicht kontextfrei.

Pumping Lemma und Annahme L kontextfrei (PL) 1 Punkt, Pumping Length (N) 1 Punkt, Wahl eines Wortes (W) 1 Punkt, Unterteilung (U) 1 Punkt, Widerspruch beim Pumpen (P) 1 Punkt, Schlussfolgerung (S) 1 Punkt.

Beispiel 16: $\{a^n b^n c^n \mid n \geq 0\}$

Wort: $w = a^N b^N c^N$



Beim Pumpen nimmt die Anzahl der a und b zu, nicht aber die Anzahl der $c \Rightarrow \forall k \neq 1. (uv^k xy^k z \notin L)$

CHOMSKY-NORMALFORM (CNF)

- S kommt auf der rechten Seite nicht vor
- Jede Regel von der Form $A \to BC$ oder $A \to a$
- $S \to \varepsilon$ ist erlaubt

UMWANDLUNG

1) Neue Startvariable $S_0 \to S$ (wenn nötig)
2) ε-Regeln:

$$\left.\begin{matrix} A \to \varepsilon \\ B \to AC \end{matrix}\right\} \Rightarrow \begin{cases} B \to AC \\ \to \not{A}C \end{cases}$$

3) Unit-Rules:

$$\left.\begin{matrix} A \to B \\ B \to CD \end{matrix}\right\} \Rightarrow \begin{cases} A \to CD \\ B \to CD \end{cases}$$

4) Verkettungen:

$A \to u_1 u_2 ... u_n$
$A \to u_1 A_1, A_1 \to u_2 A_2, ..., A_{n-2} \to u_{n-1} u_n$
falls u_i ein Terminalsymbol: $A_{i-1} \to U_i A_i, U_i \to u_i$

Beispiel 17: $S \to ASA \mid aB, \quad A \to B \mid S, \quad B \to b \mid \varepsilon$

1) Startvariable

$S_0 \to S$
$S \to ASA \mid aB$
$A \to B \mid S$
$B \to b \mid \varepsilon$

2) ε-Regel

$S_0 \to S$
$S \to ASA \mid aB \mid a$
$A \to B \mid \varepsilon \mid S$
$B \to b$

$\rightsquigarrow$

$S_0 \to S$
$S \to ASA \mid SA \mid AS \mid aB \mid a$
$A \to B \mid S$
$B \to b$

3) Unit-Regel

$S_0 \to ASA \mid SA \mid AS \mid aB \mid a$
$S \to ASA \mid SA \mid AS \mid aB \mid a$
$A \to B \mid ASA \mid SA \mid AS \mid aB \mid a$
$B \to b$

$\rightsquigarrow$

$S_0 \to ASA \mid SA \mid AS \mid aB \mid a$
$S \to ASA \mid SA \mid AS \mid aB \mid a$
$A \to b \mid ASA \mid SA \mid AS \mid aB \mid a$
$B \to b$

4) Komplexe Regeln

$S_0 \to AA_1 \mid SA \mid AS \mid UB \mid a$
$S \to AA_1 \mid SA \mid AS \mid UB \mid a$
$A \to b \mid AA_1 \mid SA \mid AS \mid UB \mid a$
$A_1 \to SA$
$U \to a$
$B \to b$

ABLEITUNGSDREIECK

Ideen

- Parse Tree aus $A \to BC$ und $T \to t$
- Variablen in Tabelle füllen

Prinzip

- Einem Teilwort entspricht ein Feld der Tabelle (rot hinterlegt)
- Das Feld wird mit den Variablen gefüllt, aus denen das Teilwort abgeleitet werden kann.

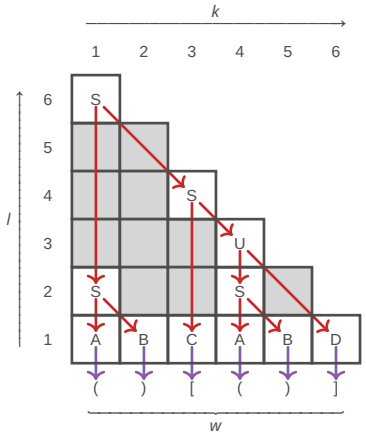
Beispiel

Parsen des Wortes `()[()]`

$S \to AB \mid CD \mid CU \mid SS$
$U \to SD$
$A \to ($
$B \to)$
$C \to [$
$D \to]$

Definitionen

Die Felder (k, l_1) und $(k + l_1, l - l_1)$ heissen ***korrespondierende Felder*** im Ableitungsdreieck des Feldes (k, l).



BNF

- Variablen: `<variablen-name>`
- Einzelne Zeichen: `A`
- Zeichenketten: `'BEISPIEL'`
- Regeln: `<variablen-name> ::= Ausdruck`
- Ausdrücke sind Folgen von Variablen, einzelnen Zeichen oder Zeichenketten, getrennt durch `|`

Beispiel 18: **BNF für Expression-Term-Factor Grammatik**

Ausgangsgrammatik

expression → expression + term | term
term → term * factor | factor
factor → (expression) | N
$N \to NZ \mid Z$
$Z \to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

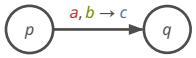
Backus-Naur-Form

```
<expression>  ::= <expression> + <term> | <term>
<term>        ::= <term> * <factor> | <factor>
<factor>      ::= ( <expression> ) | <number>
<number>      ::= <number> <digit> | <digit>
<digit>       ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

STACKAUTOMAT (PDA)

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

1) Q: Zustände
2) Σ: Eingabe-Alphabet
3) Γ: Stack-Alphabet
4) $\delta: Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to P(Q \times \Gamma_\varepsilon)$
5) $q_0 \in Q$: Startzustand
6) $F \subset Q$: Akzeptierzustände



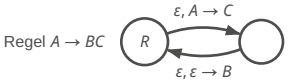
a vom Input

b vom Stack entfernen (Bedingung)

c auf den Stack

GRAMMATIK ABLESEN

Ist L eine kontextfreie Sprache, dann gibt es einen Stackautomaten P, der L akzeptiert, $L = L(P)$.

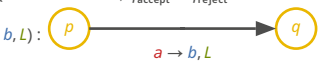


TURING MASCHINEN

(deterministische) Turing-Maschine
$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$

- Q: Zustände
- Σ: Alphabet
- Γ: Bandalphabet, $\_ \in \Gamma \setminus \Sigma$
- $\delta: Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$
- $q_0 \in Q$: Startzustand
- $q_{accept} \in Q$: Akzeptierzustand
- $q_{reject} \in Q$: Ablehnzustand, $q_{accept} \neq q_{reject}$

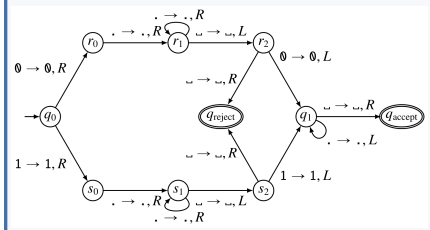
$\delta(p, a) = (q, b, L)$:



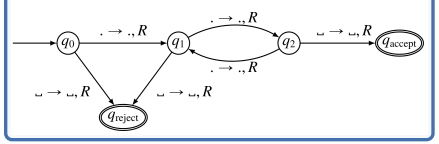
- Übergang möglich, wenn a unter dem Schreib- / Lese-Kopf
- Aktuelles Feld auf dem Band wird mit b überschrieben
- Kopfbewegung: L links, R rechts

Beispiel 19: $\Sigma = \{0, 1\}$

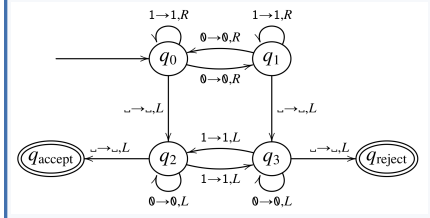
$L_1 = \{w \in \Sigma^* \mid |w| > 1 \wedge \text{first}(w) = \text{last}(w)\}$, wobei first($w$) das erste Zeichen von w und last(w) das letzte.



$L_2 = \{w \in \Sigma^* \mid |w| > 1 \wedge |w|_0 \equiv |w|_1 \bmod 2\}$



Beispiel 20: $\Gamma = \{0, 1, \sqcup\}$



Welche der Wörter 0110, 10100, 100100, 101111 und 1101000 werden akzeptiert?

akzeptiert: 0110, 100100, 101111,
nicht akzeptiert: 10100, 1101000

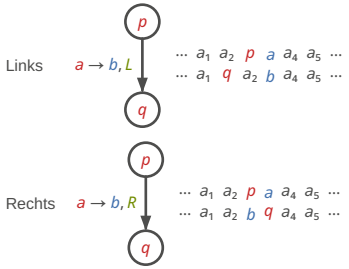
Welche Sprache wird von der Turing-Maschine akzeptiert?

Die oberen zwei Zustände durchlaufen das Wort von links nach rechts, wobei der Zustand mit jeder gefundenen 0 wechselt. Der Zustand q_0 steht also für eine gerade Anzahl gefundener Nullen, q_1 für eine ungerade Anzahl. Am Ende des Wortes wechselt die Maschine zu den unteren zwei Zuständen und tut dort dasselbe für die Einsen. Der Zustand q_2 steht dafür, dass noch eine gerade Anzahl von Einsen erwartet wird, q_3 steht für eine ungerade Anzahl.
d. h. es werden genau die Wörter gerader Länge akzeptiert.

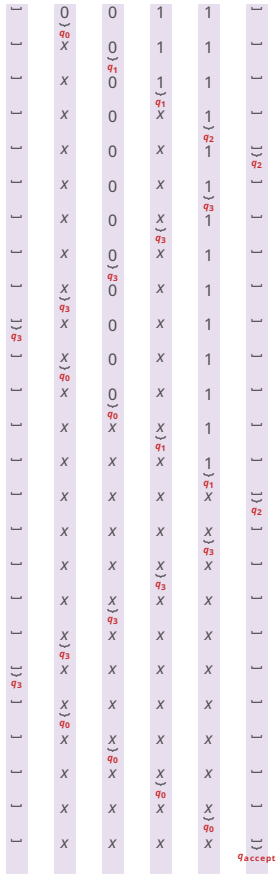
PRÜFUNG

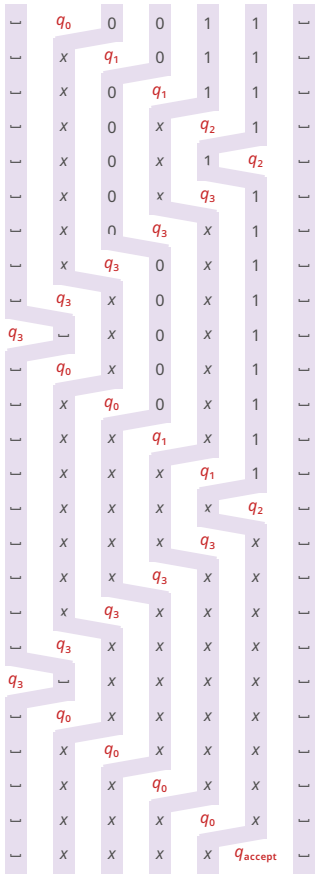
Bei der Letzten Aufgabe (Welche Sprache wird von M akzeptiert?) muss eine ausformulierte Begründung stehen.

BERECHNUNGSGESCHICHTE



$L = \{0^n 1^n \mid n \geq 0\}, w = 0011$





VARIANTEN

NICHTDETERMINISTISCHE TM

Übergangsfunktion

Bei jedem Übergang maximal N verschiedene Möglichkeiten:
$\delta : Q \times \Gamma \to P(Q \times \Gamma \times \{L, R\})$
$\Rightarrow$ Mehrere mögliche Berechnungswege

Wort akzeptieren

$w \in L(M)$ wenn es einen Berechnungsweg gibt, der zu q_{accept} führt. (auch wenn es Wege gibt, die auf q_{reject} führen!)

Simulation auf Standard-TM

Verwende 3 Bänder:

1) Arbeitsband
2) Kopie von w
3) Liste aller Folgen von Wahlmöglichkeiten

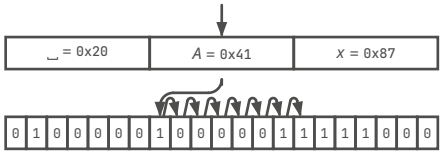
Simulation

1) Kopiere w von Band 2 auf Band 1
2) Führe TM aus auf Band 1 mit Wahlmöglichkeiten von B. 3
3) Inkrementiere zur nächsten Folge auf Band 3

Simulierbar in $O(N^{t(n)}) = 2^{O(t(n))}$

VERSCHIEDENE BANDALPHABETE

Jedes andere Alphabet kann binär codiert werden.



Simulierbar in Zeit $O(t(n))$

MEHRSPURMASCHINE

Simulierbar in Zeit $O(t(n))$

MEHRBANDMASCHINE

Die unabhängige Bewegung der Schreib-/Leseköpfe auf einer n-Band Maschine kann mithilfe eines $2n$-spurigen Bandes (Daten + Zeiger für Schreib-/Lesekopf) nachgebildet werden.

Simulierbar in Zeit $O(t(n)^2)$

EINSEITIG UNENDLICHES BAND

Beidseitig unendliches Band kann durch ein Alphabet doppelter Wortbreite realisiert werden, somit äquivalent.

TURING-ERKENNBARE SPRACHEN

Eine TM erkennt das Wort $w \in \Sigma^*$, wenn die Maschine auf dem Input w im Zustand q_{accept} anhält.

Sind L_1 und L_2 Turing-erkennbare Sprachen, dann sind auch der Durchschnitt $L_1 \cap L_2$ und die Vereinigungsmenge $L_1 \cup L_2$ Turing-erkennbar.

AUFZÄHLER

Aufzähler: TM mit einem Drucker, mit dem Wörter ausgedruckt werden können.

L ist ***rekursiv aufzählbar*** wenn es einen Aufzähler gibt, der alle Wörter aufzählen kann. L ist dann Turing-erkennbar.

ENTSCHEIDBARKEIT

Entscheider: eine Turing-Maschine, die auf jedem beliebigen Input anhält.

Turing-entscheidbare Sprache: L heisst ***Turing-entscheidbar***, wenn es einen Entscheider M gibt mit $L = L(M)$.

ENTSCHEIDBARE PROBLEME

LEERHEITSPROBLEME

✓ Entscheidbar $E_{DEA} = \{\langle A \rangle \mid L(A) = \varnothing\}$

Wie: Minimalautomat hat keinen Akzeptierzustand

✓ Entscheidbar $E_{CFG} = \{\langle G \rangle \mid L(G) = \varnothing\}$

Wie: Chomsky-Normalform

✗ Entscheidbar $E_{TM} = \{\langle M \rangle \mid L(M) = \varnothing\}$

Denn: Reduktion möglich: $A_{TM} \leq E_{TM}$

GLEICHEITSPROBLEME

✓ Entscheidbar $EQ_{DEA} = \{\langle A_1, A_2 \rangle \mid L(A_1) = L(A_2)\}$

Wie: Vergleich der minimalen Automaten, oder Leerheitsproblem bei: $L(A_1) \triangle L(A_2)$

✗ Entscheidbar $EQ_{CFG} = \{\langle G_1, G_2 \rangle \mid L(G_1) = L(G_2)\}$

Denn: Es gibt eine Reduktion $ALL_{CFG} \leq EQ_{CFG}$. Da ALL_{CFG} nicht entscheidbar, auch EQ_{CFG} nicht entscheidbar.

✗ Entscheidbar $EQ_{TM} = \{\langle M_1, M_2 \rangle \mid L(M_1) = L(M_2)\}$

Denn: EQ_{PDA} nicht entscheidbar

AKZEPTANZPROBLEME

✓ Entscheidbar $A_{DEA} = \{\langle A, w \rangle \mid w \in L(A)\}$

Wie: Regex-Engines simulieren beliebige DEAs auf beliebigen Input-Wörtern w

✓ Entscheidbar $A_{CFG} = \{\langle G, w \rangle \mid w \in L(G)\}$

Wie: CYK, deterministischer Parse-Algorithmus

✓ Entscheidbar A $\varepsilon_{CFG} = \{\langle G, w \rangle \mid \varepsilon \in L(G)\}$

Wie: wenn CNF Regel $S \to \varepsilon$ enthält

✗ Entscheidbar $A_{TM} = \{\langle M, w \rangle \mid w \in L(M)\}$

Denn: Halteproblem

ALLE WÖRTER PRODUZIEREN

✗ Entscheidbar $ALL_{CFG} = \{\langle G \rangle \mid G \text{ produziert } \Sigma^*\}$

Denn: (Gegenteil von E Problem mit TM) $A_{TM} \leq ALL_{PDA}$

✗ Entscheidbar $ALL_{PDA} = \{\langle P \rangle \mid P \text{ akzeptiert } \Sigma^*\}$

Denn: Reduktionsabbildung der Gegenteil akzeptiert

WEITERE

✗ Entscheidbar $HALT_{TM} = \{\langle M, w \rangle \mid M \text{ hält auf Input } w\}$

Denn: Auf A_{TM} reduziert → nicht entscheidbar

SATZ VON RICE

Ist P eine nichttriviale Eigenschaft Turing-erkennbarer Sprachen, dann ist P_{TM} nicht entscheidbar.

Eine Eigenschaft P Turing-erkennbarer Sprachen heisst ***nichttrivial***, wenn es zwei Turing-Maschinen M_1 und M_2 gibt, wobei M_1 die Eigenschaft hat und M_2 nicht.

***Beispiel** 21:* **Links-kürzbar**

Man sagt, die Sprache sei links-kürzbar, wenn man von einem Wort ein beliebiges Anfangsstück entfernen kann und das verkürzte Wort immer noch ein Wort der Sprache ist. Also zum Beispiel $ASDF \in L \Rightarrow SDF, DF, F, \varepsilon \in L$. Wie könnte man ein Programm aufbauen, welches immer anhält und mit welchem man andere Programme analysieren kann, ob deren akzeptierte Sprache links-kürzbar ist?

Die Eigenschaft einer Sprache, links-kürzbar zu sein, ist eine nichttriviale Eigenschaft. Die Liste der Wörter in der Aufgabenstellung bildet eine links-kürzbare Sprache, die Sprache bestehend nur aus dem Wort BIBER hat diese Eigenschaft nicht. Der Satz von Rice besagt jetzt, dass man kein Programm schreiben kann, welches entscheiden könnte, ob die akzeptierte Sprache die Eigenschaft hat, links-kürzbar zu sein.

Satz von Rice (R) 1 Punkt, Eigenschaft (E) 1 Punkt, zwei Sprachen (Z) 1 Punkt, Eigenschaft ist nicht trivial (T) 1 Punkt, Schlussfolgerung nicht entscheidbar (N) 2 Punkt.

LÖSUNGSANLEITUNG EINER PRÜFUNGSFRAGE:

- Nichttriviale Eigenschaft P aufschreiben
- Die beiden Sprachen $L(M_1)$ und $L(M_2)$ bilden (meistens kann man die leere Menge oder Σ^* als eine dieser Sprachen verwenden)
- Gibt es ein Programm, welches beide Sprachen erkennen kann? Sind beide Sprachen Turing erkennbar?
- Dann besagt der Satz von Rice, dass die Sprache nicht entscheidbar ist

***Beispiel** 22:* **Datum**

Ein häufig wiederkehrende Aufgabe in der Softwareentwicklung ist die Beurteilung, ob eine Zeichen- kette ein gültiges Datum ist. Daher könnte es nützlich sein, ein Tool zur Verfügung zu haben, welches ein Programm analysieren kann und eine Aussage liefern kann, ob das Programm nur korrekte Daten akzeptiert. Gibt es ein solches Tool?

Nein. Dies ist das Entscheidungsproblem für die Eigenschaft DATES_{TM}, wobei die Eigenschaft DATES besagt, dass die Sprache nur aus korrekten Daten besteht. Diese Eigenschaft ist nicht trivial, denn

$L_1 = \{2024-07-19\}$ hat die Eigenschaft DATES,
$L_2 = \{blubb\}$ hat die Eigenschaft nicht.

Beide Sprachen können von einer Turing-Maschine erkannt werden. Nach dem Satz von Rice ist DATES_{TM} nicht entscheidbar.

Eigenschaft DATES (P) 1 Punkt, zwei Sprachen (L) 1 Punkt, Eigenschaft ist nicht trivial (T) 1 Punkt, Anwendung des Satzes von Rice (R) 2 Punkte.

NP

Eine Sprache L gehört zur Klasse NP, wenn L von einer nichtdeterministischen TM in polynomieller Zeit **entschieden** werden kann

POLYNOMIELLE VERIFIZIERER

Ein ***polynomieller Verifizierer*** für die Sprache L ist eine TM V, so dass es für jedes $w \in \Sigma^*$ ein Wort c (das Lösungszertifikat) gibt, für das gilt

$$w \in L \Leftrightarrow V \text{ akzeptiert } \langle w, c \rangle$$

Die Laufzeit von V ist polynomiell in $|w|$.

Eine Sprache ist genau dann in NP, wenn sie in polynomieller Zeit **verifiziert** werden kann.

Bei der Prüfung Entscheidbarkeit und Zertifikat erwähnen!

***Beispiel** 23:* $n \times m$ **Kuromasu**

	Schritt	Aufwand
1	Für jedes Feld mit einer Zahl überprüfe, ob es weiss ist.	$O(nm)$
2	Für jedes schwarz Feld überprüfe, ob die vertikalen und horizontalen Nachbarn weiss sind	$O(nm)$
3	Ausgehend vom ersten weissen Feld, färbe alle horizontal oder vertikal benachbarten weissen oder roten Felder rot ein, bis kein weiteren Felder eingefärbt werden können. Wenn ein weisse Feld übrig bleibt, ist die Regel 3 verletzt.	$O(n^2m^2)$
4	Von jedem Feld mit einer Zahl ausgehend zähle die horizontal und ver- tikal benachbarten sichtbaren weissen Felder und überprüfe, ob die Anzahl mit der Zahl übereinstimmt.	$O(n^2m + nm^2)$
	Total	$O(n^2m^2)$

***Beispiel** 24:* **Square killer**

Square Killer ist eine Variante von Sudoku, die auf einem $n^2 \times n^2$ Spielfeld gespielt wird. Die normalen Sudoku-Regeln gelten, zusätzlich muss aber die Zahl der Ziffern in einem zusammenhängenden grauen Gebiet («Käfig») eine Quadratzahl sein.

Das Problem ist natürlich entscheidbar, man kann die endlich vielen möglichen Zahlenverteilungen durchprobieren und die Regeln überprüfen.

Die Fragestellung ist gleichbedeutend mit der Frage, ob es einen polynomiellen Verifizierer gibt.

Für einen polynomiellen Verifizierer brauchen wir ein Lösungszertifikat, wir nehmen dafür die Zahlen, die in den Feldern stehen. Folgende Verifikationsschritte sind noch nötig:

	Schritt	Aufwand
1	Sudoku-Verifizierer	$O(n^6)$
2	Für jedes Feld überprüfen, ob die Summe der Zahlen im gleichen Käfig eine Quadratzahl ist.	$O(n^4 \cdot n^4)$
	Total	$O(n^8)$

Entscheidbarkeit (E) 1 Punkt, Verifizierer (V) 1 Punkt, Zertifikat (Z) 1 Punkt, bestehende Sudoku Regeln sind in polynomieller Zeit verifizierbar (S) 1 Punkt, Verifikation von zwei zusätzlichen Regel (R) 1 Punkt, Laufzeit polynomiell (L) 1 Punkt.

m-SUDOKU

Entscheidbar? Ja: $n = m^4$ = Anzahl Felder
Zertifikat: Vollständig ausgefülltes Sudoku
Vorgehen:

	Schritt	Aufwand
1	Jedes Feld: Zeichen kommt auf der Zeile nicht mehr vor	$O(n^4 \cdot n^2)$
2	Jedes Feld: Zeichen kommt in der Spalte nicht mehr vor	$O(n^4 \cdot n^2)$
3	Jedes Feld: Zeichen kommt im Unterquadrat nicht vor	$O(n^4 \cdot n^2)$
	Total	$O(n^6)$

NURIKABE

Entscheidbar? Ja: $n = uv$ = Anzahl Felder. Alle Belegungen des Spielfeldes mit schwarzen Feldern können überprüft werden, ob sie die Regeln einhalten.
Zertifikat: Liste der schwarzen Felder
Vorgehen:

	Schritt	Aufwand
1	Für jedes Zahlfeld ($O(n)$) verwendet man einen Markieralgorithmus, der alle zum Gebiet dieses Zahlfeldes gehörenden weissen Felder bestimmt ($O(n)$ Durchgänge mit Aufwand $O(n)$).	$O(n^3)$
2	Trifft dieser Algorithmus auf ein weiteres Zahlfeld, ist der erste Teil von Regel 1 verletzt ($\rightarrow q_{\text{reject}}$).	$O(n)$
3	Weicht die Zahl der gefundenen weissen Felder vom Inhalt des Zahlfeldes ab, ist der zweite Teil von Regel 1 verletzt ($\rightarrow q_{\text{reject}}$).	$n \cdot O(n)$
4	Ebenfalls mit einem Markieralgorithmus wird überprüft, ob das schwarze Gebiet zusammenhängend ist.	$O(n^2)$
5	Für jedes schwarze Feld wird überprüft, ob es Teil eines 2×2- Tümpels ist.	$O(n)$
	Total	$O(n^3)$

DAMEN

Das n-Damenproblem ist die Aufgabe, eine Platzierung von n Damen auf einem $n \times n$-Schachfeld zu finden, so dass sich die Damen nicht gegenseitig schlagen können.

Entscheidbar? Ja, es kann ein polynomieller Verifizierer erstellt werden.
Zertifikat: Das Wort $(n, r_1, r_2, ..., r_n) \in$ QUEENS, wobei r_i die Zeilennummer der Dame in Spalte i ist.
Vorgehen:

	Schritt	Aufwand
1	Enthält das Wort genau $n+1$ natürliche Zahlen zwischen 1 und n?	$O(n \log(n))$
2	Sind die Zahlen r_i alle verschieden?	$O(n^2 \log(n)^2)$
3	Sind keine Damen in der gleichen Diagonalen platziert?	$O(n^2 \log(n)^2)$
	Total	$O(n^2 \log(n)^2)$

Die Faktoren $\log(n)$ kommen daher, dass mit n auch die Länge der Zahlen wie $\log(n)$ anwächst. Entsprechend nimmt auch die Zeit zu, jede dieser Zahlen zu verarbeiten. Die gesamte Laufzeit ist schneller als $O(n^3)$, also polynomiell. Damit ist der oben beschriebene Algorithmus ein polynomieller Verifizierer.

REDUKTION

Polynomielle Reduktion: $A \leq_P B$. Lies: A ist polynomiell leichter entscheidbar als B.

NP-VOLLSTÄNDIG

Reduktion auf NP-Vollständiges Problem

Zu beachten (Punkteverteilung):
- Lösungsprinzip der Reduktion erwähnen (1P)
- Auswahl eines geeigneten Vergleichsproblems (1P)
- Reduktionsabbildung (*P)
- Schlussfolgerung: NP-Vollständig und somit nicht effizient lösbar (1P)

TURING-VOLLSTÄNDIG

Eine Sprache A heisst eine ***Programmiersprache***, wenn es eine Abbildung $c : A \mapsto \Sigma^*$ gibt, wobei $c(w)$ ein Programm für eine universelle Turing-Maschine ist.

Eine Programmiersprache heisst ***Turing-vollständig***, wenn in ihr jede beliebige Turing-Maschine simuliert werden kann.

KARP-KATALOG

PROBLEME MIT k ZAHLEN

VERTEX-COLORING	CLIQUE-COVER
EXACT-CLIQUE-COVER	K-CLIQUE
VERTEX-COVER	SET-PACKING
SET-COVERING	MAX-CUT
STEINER-TREE	SEQUENCING
FEEDBACK-NODE-SET	FEEDBACK-ARC-SET

GRAPH-ÜBERDECKUNGEN

Handeln davon, dass ein gegebener Graph von einer Menge beschränkter Größe mit einer besonderen Eigenschaft aus erreicht werden kann.

VERTEX-COLORING

Gegeben ist ein Graph $G = (V, E)$ mit Knoten V und Kanten K und eine Zahl k. Ist es möglich, die Knoten des Graphen mit k Farben so einzufärben, dass durch Kanten verbundene Knoten verschiedene Farben haben?

$k = 4$

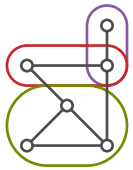
Beispiel: Stundenplan: Planung der Fächer $f \in F$ auf k Zeitfenster, damit keine Anmeldungen $\{f_1, f_2\} \in A$ in gleichen Zeitfenstern liegen

Lösung: **Knoten** V = Fach, **Kanten** E = Anmeldungen, **Farben** = Zeitfenster

CLIQUE-COVER

Gibt es k Cliquen so, dass jede Ecke in genau einer der Cliquen ist?

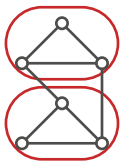
$k = 3$



EXACT-CLIQUE-COVER

Wie CLIQUE-COVER, nur müssen die Mengen disjunkt sein.

$k = 2$



Beispiel: Bilden von k Gruppen von Leuten, die sich kennen. Alle Leute sollen eingeteilt sein.

Lösung: **Knoten** V = Teilnehmer, **Kanten** E = Kennen sich, k = Anzahl Gruppen, **Clique** G_k = Gruppe

K-CLIQUE

Wie CLIQUE-COVER, nur ist k bekannt.

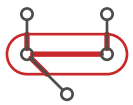
Beispiel: Job-Parallelisierbarkeit: Menge von n Jobs, die jeweils exklusiv auf m Ressourcen zugreifen, Jobs dürfen nicht gleichzeitig laufen. Ziel: Entscheiden, ob zu irgendeinem Zeitpunkt mehr als k Prozessoren ausgelastet werden können

Lösung: **Graph** G = Job-Parallelisierbarkeit, **Knoten** V = n Job, **Kanten** E = m gleiche Ressourcen, **k-Clique** G_k = Auswahl Knoten ohne Verbindung, k = Auslastbare Anzahl Prozessoren

VERTEX-COVER

Gibt es eine Menge W von k Knoten derart, dass jede Kante von G einen Endpunkt in W hat?

$k = 2$



Beispiel: Kontrolle des Verkehrsnetzes: Mitarbeiter haben ihre Basis an einzelnen Knotenpunkten. Ziel: An welchen Stationen muss man Kontrolleure stationieren, damit jede Strecke bei einem Kontrolleur endet?

Lösung: **Knoten** V = Stationen, **Kanten** E = Strecke, k = Anzahl Kontrolleure, $W \subset V$ = Knoten mit Kontrolleur

MENGEN

Handeln von einer Familie $(S_i)_{i \in I}$ von nichtleeren endlichen Mengen. In diesen Problemen geht es um die existenz einer Teilfamilie mit bestimmten Eigenschaften.

SET-PACKING

Wie viele der Mengen S_i kann man in U hineinpacken, ohne dass sie sich überlappen? Gibt es k Teilmengen in S, welche disjunkt sind und die Menge U abdecken?

$k = 5$

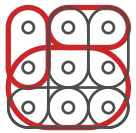
Beispiel: Medizinische Studie: 17 Allergene wählen, sodass jeder Proband maximal auf eines davon reagiert.

Lösung: I = Allergene, S_i = auf Allegien i allergische Probanden, J = Ausgewählte Allergene, $S_i \cap S_j = \emptyset \rightarrow$ Ausschlussbedingung zwischen Allergenen

SET-COVERING

Gibt es eine Unterfamilie bestehend aus k Mengen, die die gleiche Vereinigung U hat? Kann man k Teilmengen bilden, welche die Menge S komplett abdecken?

$k = 3$



Beispiel: Ein Tourist möchte mit dem Besuch von nur k Aussichtspunkten alles sehen, was man an Sehenswürdigkeiten von allen Aussichtspunkten aus sehen könnte.

Lösung: U = Alle Sehenswürdigkeiten, $i \in I$ = Aussichtspunkt i, S_i = Sichtbare Sehenswürdigkeiten von i aus, $J \subset I$ = Ausgewählte Aussichtspunkte, $\bigcup_{j \in J} S_j = U$ = Bedingung, alles sehen zu können

EXACT-COVER

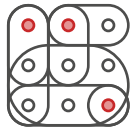
Gibt es eine Unterfamilie von Mengen, die disjunkt sind, aber die gleiche Vereinigung haben? Jedes Element in U soll genau in einer der Teilmengen der Familie S vorkommen. Die gesuchte Menge bildet eine exakte Überdeckung.

Beispiel: Es stehen binäre Eigenschaften von Kunden zur Verfügung, z.B. ob sie ein bestimmtes Produkt gekauft haben oder volljährig sind. Ziel: Eine Teilmenge von Kriterien finden, dass jeder Kunde genau eine der Eigenschaften hat.

Lösung: I = Eigenschaften, $(S_i)_{i \in I}$ = Kunden, auf die die Eigenschaft zutrifft , $J \subset I$ = Teilmenge

HITTING-SET

Sucht zu U eine Menge von Punkten (Elemente), die jede Menge der Familie in genau einem Punkt treffen. Gibt es eine Menge $W \subset U$ derart, dass W mit jeder Menge S_i nur ein Element gemeinsam hat?



Beispiel: Finden eines Einzelnen Vertreters für Gruppen von Menschen.

Lösung: $i \in I$ = Gruppierungskriterien, S_i = Menschen in Gruppe i, $W \subset \bigcup_{i \in I} S_i$ = Teilmenge von Vertretern, $|W \cap S_i| = 1 \forall i \rightarrow H$ hat mit jeder der Mengen S_i genau einen Vertreter gemeinsam

AUFTEILUNG IN ZWEI MENGEN

PARTITION

Gibt es eine Unterteilung einer Liste S von ganzen Zahlen in zwei disjunkte Listen mit gleicher Summe?

$[\underbrace{1, 2, 4, 5}_{\rightarrow 12}, \underbrace{6, 8}_{\rightarrow 12}]$

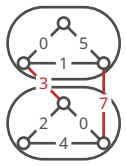
Beispiel: Unterteilung der Prominenten + Entourage auf zwei Säle des Film-Festivals.

Lösung: $i \in I$ = Promi, c_i = Anz. Mitglieder seiner Entourage, A = Stars samt Entourage für Saal A, B = für Saal B, $I = A \cup B, \sum_{i \in A} c_i = \sum_{i \in B} c_i$

MAX-CUT

Gegeben ein Graph $G = (V, E)$ mit einer Gewichtsfunktion $\varphi : E \rightarrow \mathbb{Z}$ der Kanten und einer ganzen Zahl $k \in \mathbb{Z}$, gibt es eine Aufteilung der Knotenmenge $V = V_1 \cup V_2$ in zwei disjunkte Teilmengen, so dass das Gewicht der Kanten, die V_1 und V_2 verbinden, mindestens w ist:

$k = 10$



$$\varphi(V_1, V_2) = \sum_{v_1 \in V_1, v_2 \in V_2, e = (v_1, v_2) \in E} \varphi(e) \geq k?$$

Beispiel: Feindliche Übernahme einer Firma, mit resultierender Aufteilung der Abteilung, sodass diese möglichst ineffizient miteinander kommunizieren können.

Lösung: V = Abteilung, E = Kommunikationsbeziehung, φ = Kommunikationsvolumen

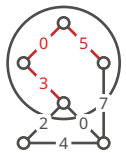
KOMBINATIRISCHE OPTIMIERUNG

Handelt um das finden eines Maximums oder Minimums einer Zielfunktion.

STEINER-TREE

Gibt es zu einem Graphen $G = (V, E)$ mit einer Gewichtsfunktion $\varphi : E \rightarrow \mathbb{Z}$, einer Teilmenge $R \subset V$ der Knoten und einer Zahl k einen Teilbaum $T = (V_1, E_1) \subset G$ des Graphen, der alle Knoten von R enthält und dessen Kantengewicht $\sum_{e \in E_1} \varphi(e) \leq k$ ist?

$k = 8$



Beispiel: Stromnetzausbau zwischen Ortschaften gegeben eines Budgets.

Lösung: **STEINER-TREE** = Stromnetz, **Knoten** V = Ortschaften, **Knoten** $R \subset V$ = zu erschliessende Ortschaften, **Gewicht** φ = Baukosten einer Verbindung, **Max. Gewicht** k = Budget

SEQUENCING

Die Komponenten der drei Vektoren

$$\text{Laufzeiten: } T = (T_1, ..., T_p) \in \mathbb{Z}^p$$
$$\text{Deadlines: } D = (D_1, ..., D_p) \in \mathbb{Z}^p$$
$$\text{Strafen: } P = (P_1, ..., P_p) \in \mathbb{Z}^p$$

beschreiben je einen Job. Job i hat Laufzeit T_i, sollte zur Zeit D_i abgeschlossen sein und kostet die Strafe P_i, wenn er nicht rechtzeitig fertig ist. Die Jobs werden sequenziell ohne unterbruch abgearbeitet. Ist es möglich, die Reihenfolge der Ausführung der Jobs so zu planen, dass die entstehenden Strafen $\leq k$ sind?

Beispiel: Eine Firma hat eine bestimmte Anzahl laufende Verträge. Es ist nicht möglich, alle Verträge in einer bestimmten Zeit abzuarbeiten. Sie versucht also möglichst viele Verträge in der verbleibenden Zeit abzuarbeiten, die eine hohe Entlöhnung haben.

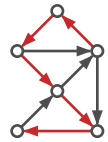
Lösung: T = Zeitaufwände für Vertragserfüllung, P = Entlöhnung der Verträge, D = Deadlines der Verträge

PFADE IN GRAPHEN

Ein hamiltonscher Pfad in einem Graphen $G = (V, E)$ ist ein Pfad, der jeden Knoten des Graphen genau einmal besucht. Ein solcher Pfad definiert eine Reihenfolge $a_1, ..., a_n$ von Knoten, so dass jede Kante von a_i nach a_{i+1} in E ist. Für einen gerichteten Graphen sind dies die Kanten $(a_i, a_{i+1}) \in E$, für einen ungerichteten Graphen gilt $\{a_i, a_{i+1}\} \in E$. Ein hamiltonscher Zyklus ist ein geschlossener hamiltonscher Pfad.

HAMPATH

Gibt es einen gerichteten hamiltonschen Pfad im gerichteten Graphen G?



Beispiel: Ganze Schweiz bereisen, ohne eine Stadt mehrmals zu besuchen

Lösung: G = Liniennetz, V = Stadt, E = ÖV-Verbindung, **Hamiltonscher Pfad** = Reise

HAMCIRCUIT

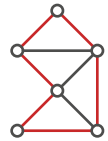
Wie HAMPATH, aber zusätzlich muss der Pfad geschlossen sein

UHAMPATH

Wie HAMPATH, aber ungerichtet

UHAMCIRCUIT

Wie UHAMPATH, aber zusätzlich muss der Pfad geschlossen sein



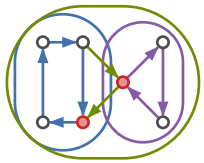
GERICHTETE GRAPHEN

Es handelt sich um die Eigenschaften von Zyklen in gerichteten Graphen $G = (V, E)$. Die Kantenmenge E besteht aus gerichteten Kanten (a, e) mit $a, e \in V$. Ein gerichteter Zyklus in G ist eine Folge von Knoten $a_0, ..., a_n = a_0$, die jeweils Enden von Kanten sind, also $(a_i, a_{i+1}) \in E, i = 0, ..., n - 1$

FEEDBACK-NODE-SET

Gibt es zu einem gerichteten Graphen G und einer Zahl k eine Menge $W \subset V$ von k Knoten derart, dass jeder geschlossene Zyklus durch einen dieser Knoten geht?

$k = 2$



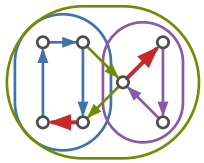
Beispiel: Es gibt mehrere Buslinien. Wo muss das Putzpersonal platziert werden, damit alle Linien geputzt werden können? Man möchte möglichst wenig Personal einsetzen.

Lösung: V = Stationen, $W \subset V$ = Stationen, von denen aus alle Linien erreicht werden, k = Anzahl Personal, G = Buslinienplan, E = Gerichtete Buslinien

FEEDBACK-ARC-SET

Gibt es zu einem gerichteten Graphen G und einer Zahl k eine Menge $F \subset E$ von k Kanten derart, dass jeder geschlossene Zyklus eine dieser Kanten enthält?

$k = 2$



LOGIK

SAT

Gegeben einer aussagenlogischen Formel, gibt es eine Zusammensetzung der Variablenwerte, die die Aussage «Wahr» werden lassen?

Beispiel: Elektriker: Konfiguration der Schalter zweier Stromkreise, die in allen Räumen Licht brennen lassen sollten

Lösung: x_i = Schalter, $T \vee F$ = Stromkreise (werte der Variable x_i, Lampen des ersten Kreises leuchten, wenn $x_i = T$), $x_{i_1} \vee ... \vee x_{i_k}$ = ob Licht im Raum der mit x_i verbundenen Schalter brennt

3SAT

Wie SAT, nur hat jede Klausel der Normalform maximal 3 Literale.

Beispiel: SAT → 3SAT

Lösung:

$$(x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (x_5 \vee x_6)$$
$$\rightarrow (x_1 \vee x_2 \vee z_1) \wedge (\overline{z_1} \vee x_3 \vee x_4) \wedge (x_5 \vee x_6 \vee x_6)$$

WEITERE

SUBSET-SUM

Gegeben einer Liste S von natürlichen Zahlen $s_i \in \mathbb{N}$ und einer natürlichen Zahl $t \in \mathbb{N}$, ist es möglich eine teilliste $T \subset S$ auszuwählen, sodass die Elemente von T die Summe $t = \sum_{s \in T} s$ haben?

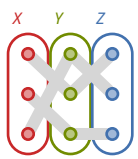
$$S = [a, b, c, d, e]$$
$$\downarrow \quad \downarrow \quad \downarrow$$
$$T = [a, \quad c, \quad e]$$
$$t \overset{!}{=} a + c + e$$

Beispiel: Rucksack-Problem (Ziel: Rucksack füllen)

Lösung: S = Grössen der Gegenstände, t = Max. Platz im Rucksack, $T \subset S$ = Grössen der Gegenstände, die in den Rucksack passen

3D-MATCHING

Gegeben sind drei endliche Mengen X, Y und Z mit gleich vielen $n = |X| = |Y| = |Z|$ Elementen und $T \subset X \times Y \times Z$. Gibt es eine n-elementige Teilmenge $M \subset T$ derart, dass keine zwei Tripel von M in einer Koordinate übereinstimmen?



Beispiel: Ein Koch hat n Rezepte für Vorspeisen, Hauptspeisen und Desserts. Nicht alle Vorspeisen lassen sich mit jeder Hauptspeise kombinieren, dasselbe gilt auch für Desserts. Er möchte n Menus zusammenstellen, sodass jedes Rezept in genau einem der Menus vorkommt.

Lösung: X = Vorspeisen, Y = Hauptspeisen, Z = Desserts, $T \subset X \times Y \times Z$ = Menge aller Menus, $M \subset T$ = Menge der zusammengestellter Menus

BIP

BIP = Binary Integer Programming

Zu einer ganzzahligen Matrix C und einem ganzzahligen Vektor d, ist ein binärer Vektor x zu finden mit $C \cdot x = d$

Beispiel:

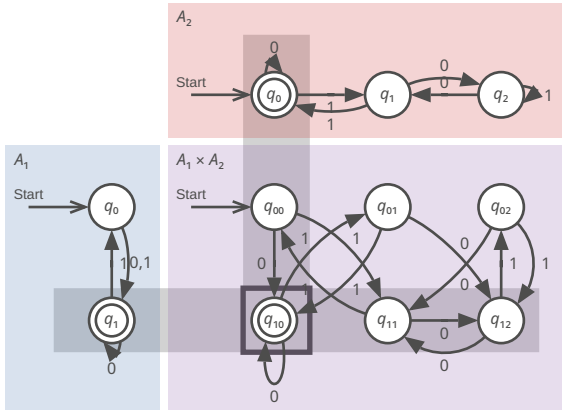
$$C = \begin{pmatrix} 1 & 3 & 0 \\ 0 & 2 & 5 \end{pmatrix} \quad d = \begin{pmatrix} 1 \\ 5 \end{pmatrix}$$

Lösung:

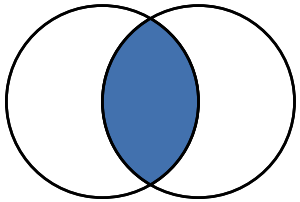
$$x = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \quad \text{da} \quad \begin{pmatrix} 1 & 3 & 0 \\ 0 & 2 & 5 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 5 \end{pmatrix}$$

MENGEN

SCHNITTMENGE $L(A_1) \cap L(A_2)$



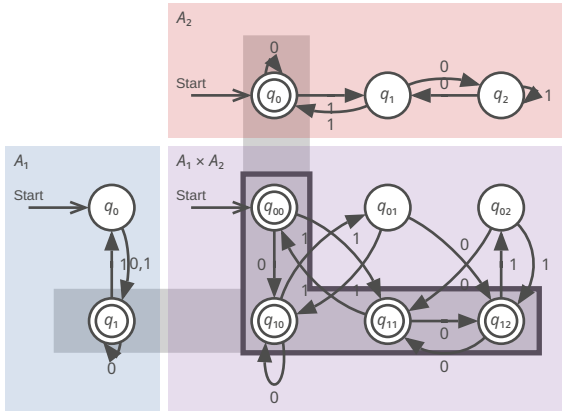
$A_1 \cap A_2$



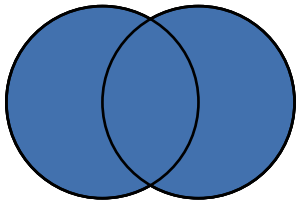
Endzustände:

$F = F_1 \times F_2$

VEREINIGUNGSMENGE $L(A_1) \cup L(A_2)$



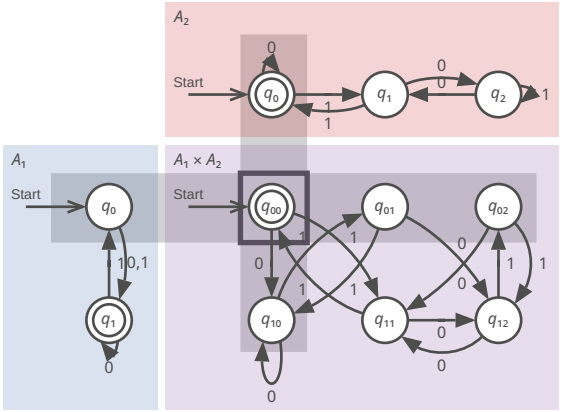
$A_1 \cup A_2$



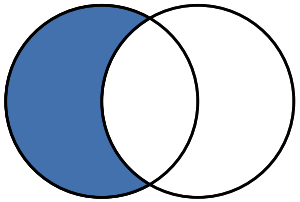
Endzustände:

$F = F_1 \times Q_2 \cup Q_1 \times F_2$

DIFFERENZMENGE $L(A_1) \setminus L(A_2)$



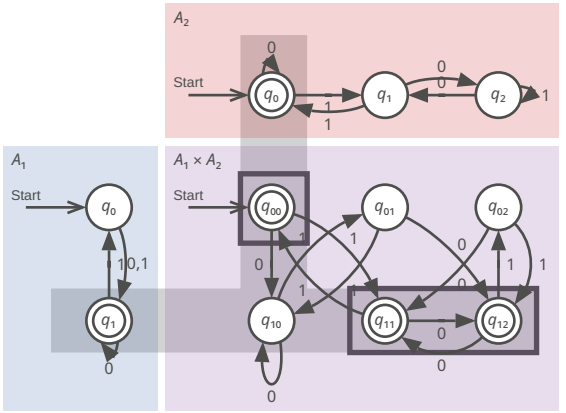
$A_1 \setminus A_2$



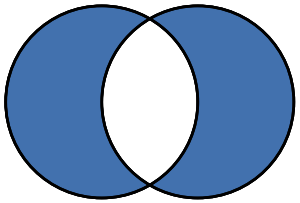
Endzustände:

$F = F_1 \times (Q_2 \setminus F_2)$

SYMMETRISCHE DIFFERENZ $L(A_1) \triangle L(A_2)$



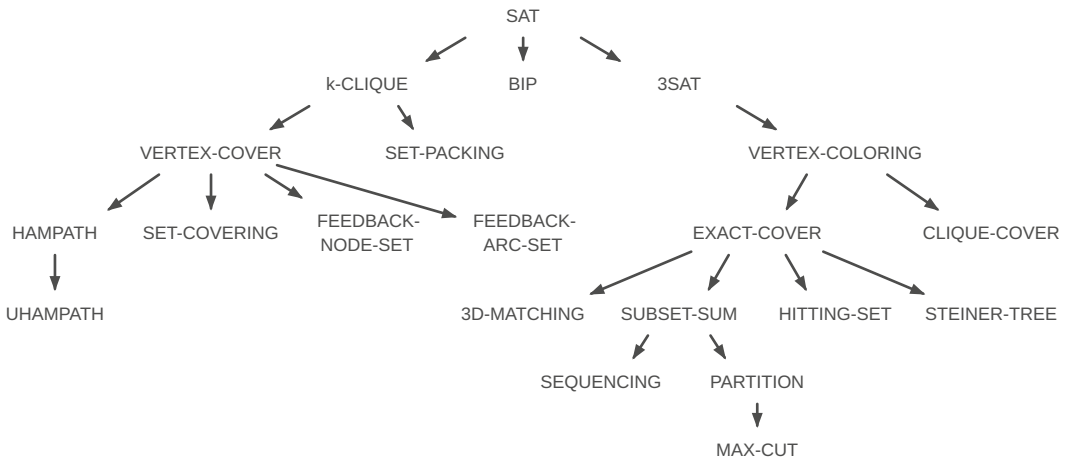
$A_1 \triangle A_2$



Endzustände:

$F = F_1 \times (Q_2 \setminus F_2) \cup (Q_1 \setminus F_1) \times F_2$

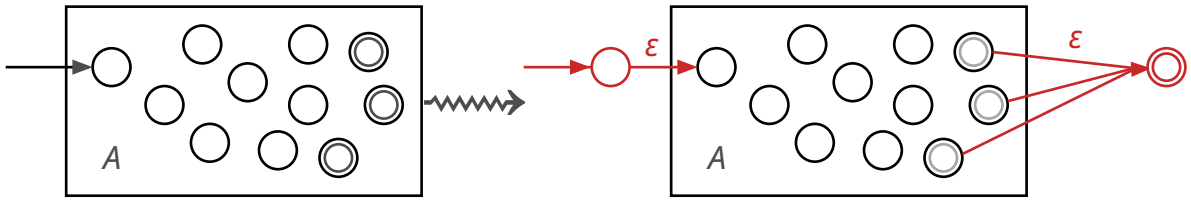
KARP KATALOG



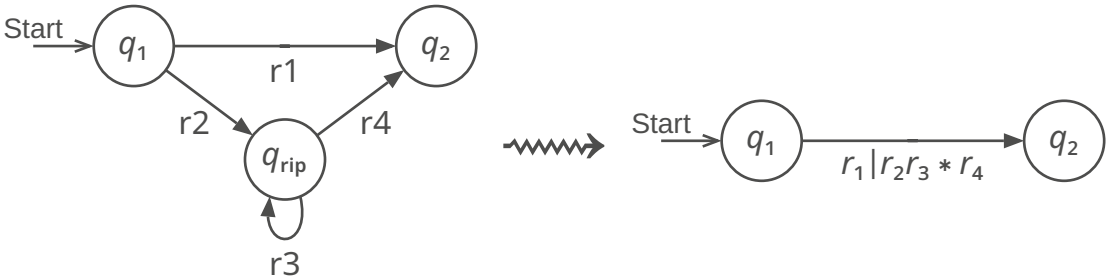
REGULÄRE AUSDRÜCKE

VNEA ZU REGEX

Keine Übergänge nach q_0 und nur ein Akzeptierzustand



Nach Entfernen aller Zwischenzustände $q_{\text{rip}} \in Q$ bleibt ein regulärer Ausdruck r von A

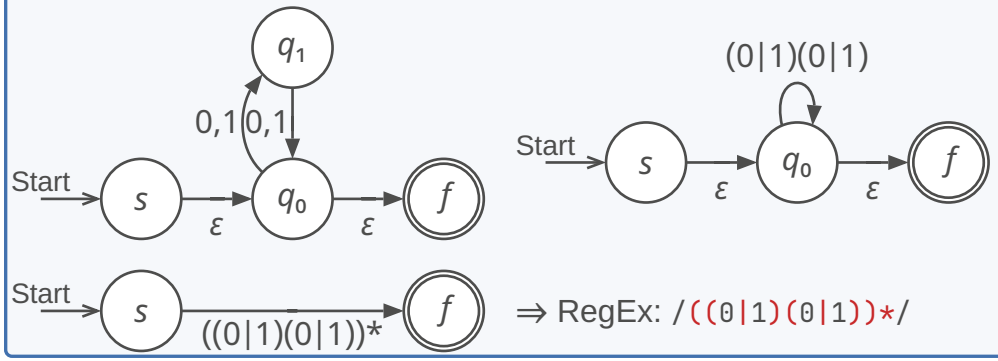


***Beispiel** 25:* $L = \{w \in \Sigma^* \mid |w|_0 - |w|_1 \equiv 0 \bmod 2\},\ \Sigma = \{0,1\}$

Es gibt den Automaten

⇒ L ist regulär

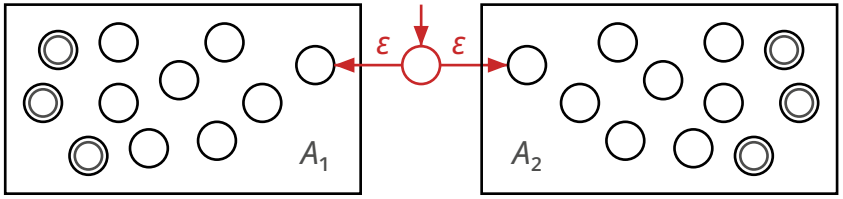
Umwandlung zu RegEx:



⇒ RegEx: `/((0|1)(0|1))*/`

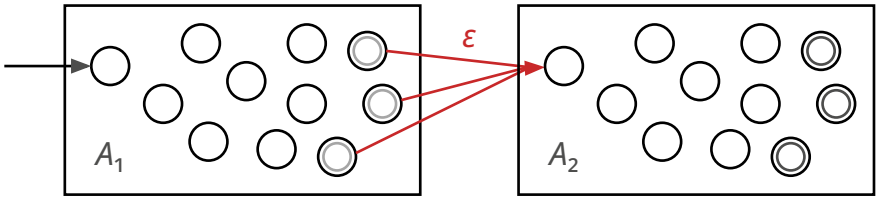
ALTERNATIVE

$$L = L_1 \cup L_2$$
$$= L(A_1) \cup L(A_2)$$
$$= L(A_1) \mid L(A_2)$$



VERKETTUNG

$$L = L_1L_2$$
$$= L(A_1)L(A_2)$$
$$= \{w_1w_2 \mid w_i \in L_i\}$$



*-OPERATION

$$L^* = \{\varepsilon\} \cup L \cup L^2 \cup \ldots$$
$$= \bigcup_{k=0}^{\infty} L^k$$