03.04.2018

Theoretische Informatik II – Sommersemester 2018

1. Formale Sprachen

Bsp. Formale Sprache = Menge von Worten

Σ = ~~Hg~~ Alphabet $\neq \emptyset$, endlich

z.B. $\Sigma = \{a_1, a_2, \dots, a_n\}$

Leeres Wort ε ~ $w\varepsilon := w$

Menge aller Worte über Σ $:= \Sigma^*$

- Sprache der arithmetischen Ausdrücke: (EXPR)
  - a steht für beliebige Zahl
  - a+a, a-a, a, a+a/a, ..., (a+a)·a, (.(a)...)
    sind Ausdrücke
  - (a(ist kein Ausdruck!
  - Problem: Wie (algorithmisch) Prüfen, ob eine gegebene Eingabe ein korrekter arithmetischer Ausdruck ist?

⇒ Beschreibung: (→ Grammatik)

- E für arithmetischen Ausdruck,
- a für eine beliebige Zahl

$E \to a$, $E \to E+E$, $E \to E*E$, $E \to (E)$,
$E \to E/E$, $E \to E-E$

Regel

$\to$ ersetze durch

Formale Grammatik

- $\Sigma = \{a, (,), +, -, *, /\}$ (Terminale) ‖ E ist Nichtterminal

- Ableitung von

$$E \Rightarrow \underline{E}+E \Rightarrow \underline{E*E}+E \Rightarrow^3 a*a+a$$

$$E \Rightarrow E*E \Rightarrow E*(E) \Rightarrow a*(E) \Rightarrow^2 a*(a+a)$$

Problem:

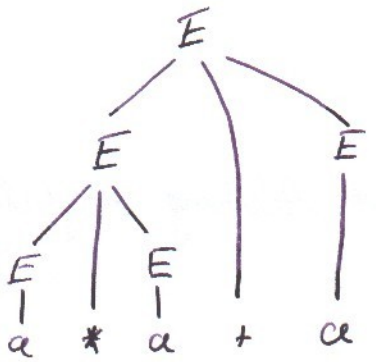
$$E \Rightarrow E*\underline{E} \Rightarrow E*\underline{E+E} \Rightarrow^3 a*a+a$$

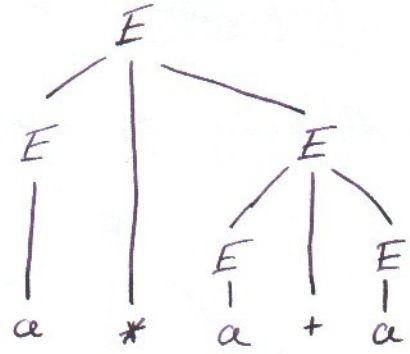
(Ableitung entspricht nicht „Punkt vor Strich-Regel")

03.04.2018

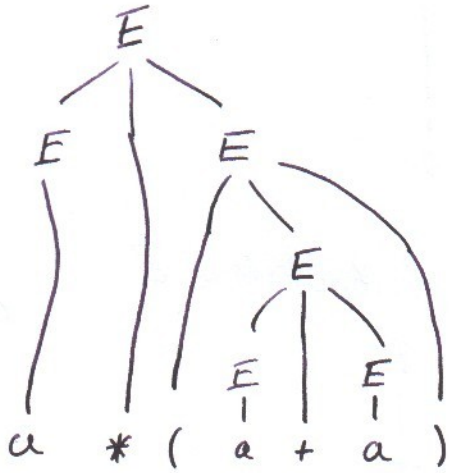
Ableitungsbäume (Darstellung der Ableitung)

[für a*a+a]





[für a*(a+a)]



Lösung: (Verhindern des „Problembaums")

- Weiteres Nichtterminal T; Aus T sollen alle (und F) Ausdrücke ...*..., oder auch (...) abgeleitet sind.
- Regeln für T:

~~T → T~~ T → F

T → T * F

F → a

F → (E)

(T ⇒ T*F

⇒ T*F*F

⇒* F*...*F)

Regeln für E:

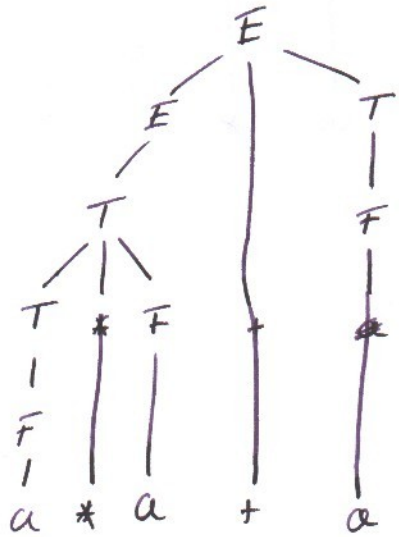
$E \to T$

$E \to E + T$

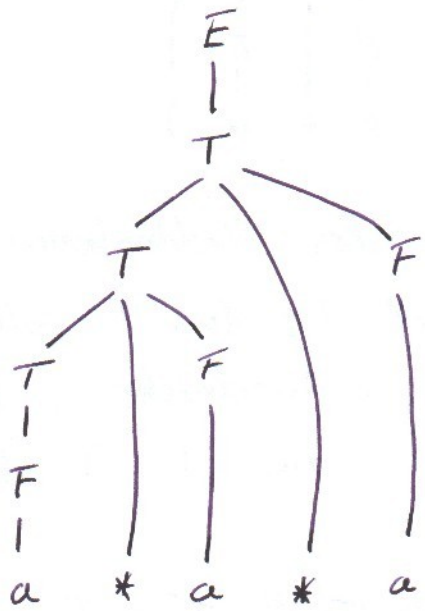
Startsymbol: E

- Ableitung zu $a * a + a$

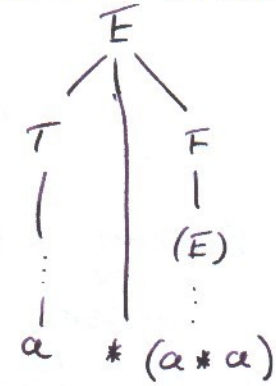
⇒ Keine andere Ableitung möglich!



- Assoziativität der Multiplikation?



andere Klammerung



„explizit".

(a * a) * a linksgeklammert

03.04.
2018

weitere Beispiele:

- Die Sprache $L = \{a^n b^n \mid n \geq 0\}$ $\Sigma = \{a, b\}$

$= \{\varepsilon, ab, aabb, aaabbb, \ldots\}$

($n=0$, $n=1$, $n=2$, $n=3$)

Grammatik:

Startsymbol: $S \to \varepsilon$

$S \to ab$

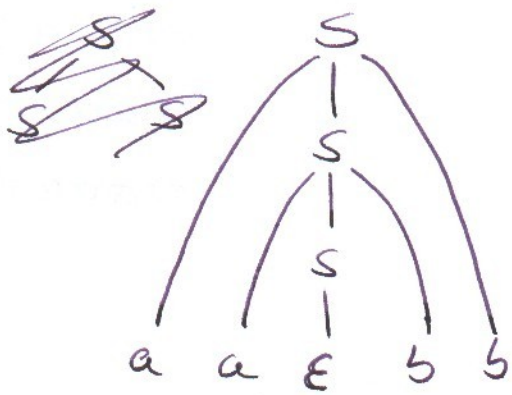
$S \to aabb$

$\vdots$?

$\approx$

$S \to \varepsilon$ „Rekursionsanfang"

$S \to aSb$ „Rekursion"



- $L = \{a^n b^n c^n \mid n \geq 0\}$ $\Sigma = \{a, b, c\}$

Auch Regeln der Art:

$TEF \to \ldots$

(Ersetzung von Worten.)

Kontextsensitiv

$S \to aSBC$

$S \to \varepsilon$

Was gibt das?

$S \Rightarrow aSBC \Rightarrow aaSBCBC \Rightarrow aaaSBCBCBC$

$\ldots \Rightarrow a^n (BC)^n$

Ziel: $(BC)^n$ soll # nach $b^n c^n$ gehen und nach nichts anderem!

Tauschregeln: $CB \to BC$ (Kontextsensitiv: Ersetzung in Abhängigkeit des Kontextes! Vorher: „kontextfrei")

Was können wir jetzt?

↳ $S \Rightarrow \ldots \Rightarrow aaaBCBCBC$

$\Rightarrow aaaBBCCBC$

$\Rightarrow aaaBBCBCC$

$\Rightarrow aaaBBBCCC$

($B \to b$, $C \to c$ dann fertig. Aber: geht nicht! $B \to b$, $C \to c$ zu früh anwendbar!)

- Statt $B \to b$, $C \to c$ nehmen wir: $aB \to ab$ (nur B ersetzen, ~~die~~ das rechts neben „a" ~~steht~~ steht) z.B.

  $\#bB \to bb$ (nur B ersetzen, wenn links davon nur „b" stehen)

  $bC \to bc$

  $cC \to cc$ (analog)

03.04.2018

Wie kann man hier „stecken bleiben"?

$S \Rightarrow^3 aaBCBC \Rightarrow aabCBC \Rightarrow aabcBC$

Keine Regel mehr anwendbar!

~~Sprache~~

Am Ende Worte nur über dem Terminalalphabet.

„richtige" Ableitung:

$S \Rightarrow^* aaBCBC \Rightarrow^* aaBBCC$

$\Rightarrow^* aabbcc$

(auch möglich: $C \to c$,
„falsche" Anwendung führt zu „Sackgasse",
es kann kein Terminalwort entstehen.)

Alternative: $S \to \varepsilon$

$S \to SABC$

$(S \Rightarrow^* ABC\,ABC \ldots ABC)$

$CA \to AC$

$BA \to AB$

$CB \to BC$

$aB \to ab$

$bB \to bb$

$bC \to bc$

$cC \to cc$

Wie bekommt man das erste „a"?

09.04.2018

(Wiederholung $L = \{a^n b^n \mid n \geq 0\}$

$S \to \varepsilon$

$S \to aSb$)

weiter bei $L = \{a^n b^n c^n \mid n \geq 0\}$ (Wdh.)

Bsp: $S \Rightarrow^{\#3} aaBCBC \Rightarrow aaBBCC \Rightarrow \ldots \Rightarrow a^2b^2c^2$

$\Rightarrow aabBCC \Rightarrow \ldots \Rightarrow$ – " –

$\Rightarrow aaBcBC$ (mit $C \to c$)

$\Rightarrow \ldots \Rightarrow aabcBc$

(Ende: Kein „Terminalwort")

Bsp.: alternatives Prinzip

- $S \to \varepsilon$, $S \to SABC$ Idee: $S \Rightarrow \ldots \Rightarrow (ABC)^n$
- $BA \to AB$, $CA \to AC$, $CB \to BC$

  Idee: $(ABC)^n \Rightarrow \ldots \Rightarrow A^n B^n C^n$
- $aB \to ab$, $bB \to bb$, $bC \to bc$, $cC \to cc$
- Falsch: $A \to a$

  geht nicht, denn:

  $S \Rightarrow \ldots \Rightarrow ABCABC \Rightarrow aBCABC$

  $\Rightarrow \ldots \Rightarrow abcabc$ ↯

Lösung: Erstes A kennzeichnen als A'

$S \to \varepsilon \mid SABC \mid A'BC$

$BA \to AB$
$CA \to AC$
$CB \to BC$
} Tauschregeln

$A' \to a$ } 1. a erzeugen, A' immer vorne

$aA \to aa$
$aB \to ab$
$bB \to bb$
$bC' \to bc$
$cC' \to cc$
} oder $C' \to c$

} restliche Terminale in richtiger Reihenfolge erzeugen!

Definition Grammatik:

- V Menge von Nichtterminalsymbolen, $V \neq \emptyset$, endlich
- Σ Menge von Terminalsymbolen, $\Sigma \neq \emptyset$, endlich
- $S \in V$ Startsymbol
- Eine Menge mit Elementen der Art $w \to v$,
  (wobei w mindestens ein Nichtterminalsymbol enthält und $w, v \in (V \cup \Sigma)^*$
  P, endlich

$(V \cup \Sigma = \emptyset)$

Regelanwendung: $xwy \Rightarrow xvy$ mit $w \to v$ Regel
$x, y \in (V \cup \Sigma)^*$

- Wiederholte Regelanwendung:

$$x_0 \Rightarrow x_1 \Rightarrow x_2 \Rightarrow \dots \Rightarrow x_n$$ (Anwendung von n Regeln)

- Schreibweise: $x_0 \Rightarrow^* x_n$

- Grammatik: $G = (V, \Sigma, S, P)$

- Sprache der Grammatik: L_G bzw. $L(G)$

$$L(G) = \{ w \in \Sigma^* \mid \text{Es gibt eine Ableitung } S \Rightarrow \dots \Rightarrow w \ (S \Rightarrow^* w) \}$$

Verschiedene Typen von Regeln:

- $A \to w$, $S \to aSb$

  ⇝ Kontextfrei

- $CB \to BC$, $aB \to ab$

  ⇝ Kontextsensitiv!

09.04.2018

Chomsky Hierarchie

- Typ ∅: Keine weiteren Einschränkung an die Regeln
- Typ 1: (Kontext-sensitiv) Für jede Regel $w \to v$ gilt: $|w| \leq |v|$
  $|w|$ = Länge von w als Wort über $\Sigma \cup V$.
  $|\varepsilon| = 0$
- Typ 2: (Kontextfrei) Regeln der Form $A \to v$, $|v| \geq 1$
- Typ 3: (regulär) Alle Regeln von der Form $A \to a$, $A \to aB$, $A, B \in V$ oder alle von der Form $a \in \Sigma$ $A \to a$, $A \to Ba$

(Typ 3 ⇒ Typ 2 ⇒ Typ 1 ⇒ Typ ∅)

Problem: Leeres Wort ε lässt sich nach strikter Definition bei Typ 3, 2 und 1 nicht ableiten!

⇒ Wenn wir ε dabei haben wollen, dann ~~muss alles~~ Regel $S \to \varepsilon$ und S kommt nie auf der rechten Regelseite vor!

(ε-Sonderregel)

(ε hinzufügen: Neues Startsymbol S', $S' \to \varepsilon$, $S' \to S$)

Wortproblem einer Sprache $L \subseteq \Sigma^*$:

- Eingabe: $w \in \Sigma^*$
- Frage: $w \in L$ oder $w \notin L$?

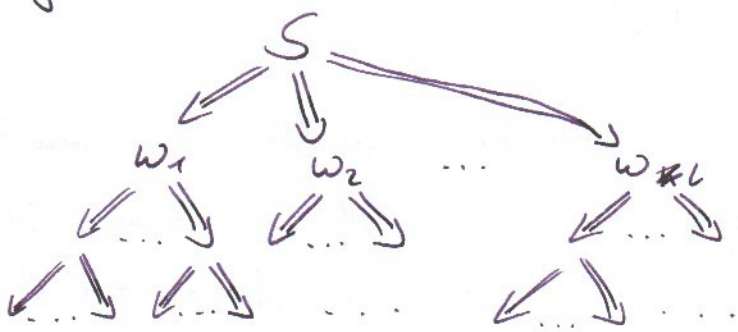
$\rightsquigarrow$ Algorithmen dafür?!

Beobachtung: Ist L vom Typ 1, so haben wir einen Algorithmus für das Wortproblem von L!

Beweis:
- Nehmen uns eine Typ 1 Grammatik von L. Eingabe ist $w \in \Sigma^*$
- 1. Fall: $w = \varepsilon$, prüfe ob Regel $S \to \varepsilon$ existiert.
- 2. Fall: $w \neq \varepsilon$,

$w \in L \Leftrightarrow$ Es gibt eine Ableitung in G. $S \Rightarrow^* w$

Grammatikbaum



Tiefe k = alle Worte, die durch Ableitung der Länge k aus S kommen.

09.04.
2018

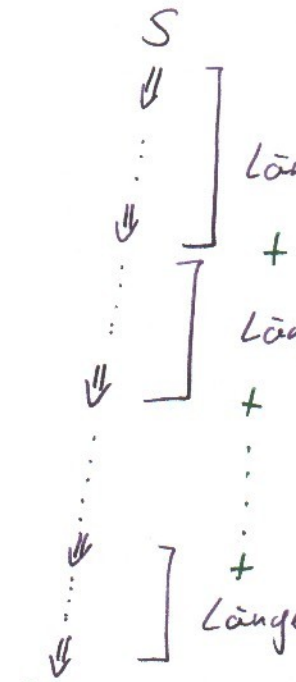
Frage: Kommt w in den Baum vor?

⇒ Wie groß muß der Baum werden, um sicher zu sehen, ob $w \in L$ oder nicht?

- Ist $|w_j| \geq |w|$, dann unter w_j im Baum kein w! (wegen Typ 1)
- Haben wir die Situation w_j, dann am unteren w_j aufhören.

(Mit diesen 2 Bedingungen ist der Baum endlich!)

Wieviele Worte auf einem Weg? (maximale Weglänge; # Satzformen mit Länge $\leq |w|$)



(• Länge $|w|+1$)
↳ +1

$$\left(\sum_{j=1}^{|w|} |V \cup \Sigma|^j \right) + 1 = \sum_{j=0}^{|w|} |V \cup \Sigma|^j$$

$$= \frac{|V \cup \Sigma|^{|w|+1} - 1}{|V \cup \Sigma| - 1}$$

(Exponentiell in $|w|$)

10.04.2018

⇒ Typ 1, Typ 2, Typ 3 entscheidbar

⇒ Vorgehen hier führt zu exponentieller Laufzeit

⇒ für Typ 0 geht dies so nicht!

- Tatsächlich ist das Wortproblem für Typ 0 nicht entscheidbar!

→ Da Ableitungen auch wieder kürzer werden können!

10.04.2018

2. Reguläre Sprachen

- Regeln: $A \to a$, $A \to aB$ $\quad a \in \Sigma$
- Bsp.: Zahlen mit Dezimalpunkt: 0.111 , 1.1223

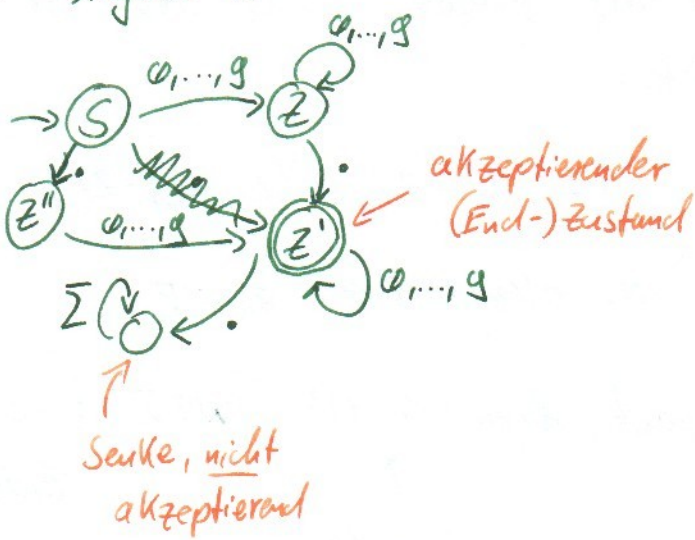
Grammatik:

$S \to 0Z \mid 1Z \mid \dots \mid 9Z \mid .Z'$

$Z \to 0Z \mid 1Z \mid \dots \mid 9Z \mid . \mid .Z'$

$Z' \to 0Z' \mid \dots \mid 9Z' \mid 0 \mid \dots \mid 9 \mid$

Automat: Eingabe w



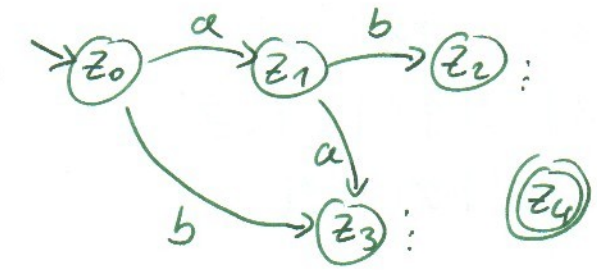
⇒ deterministischer endlicher Automat

⇒ Wortproblem in linearer Zeit $O(n)$

Frage: geht das für alle regulären Sprachen?

- **Endlicher Automat**
  - Q Menge der Zustände, endlich, $Q \neq \emptyset$
  - Σ Eingabealphabet
  - Die Funktion $\delta : (Q \times \Sigma) \to Q$ (δ vollständig definiert!)
  - $z_0 \in Q$ Startzustand
  - $E \subseteq Q$ Menge der Endzustände

Bsp.:



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

Rechnung auf Eingabe $a_1 a_2 \dots a_n$ ist Folge von Zuständen:

$$z_0, \delta(z_0, a_1) = z_1, \delta(z_1, a_2) = z_2, \dots, \delta(z_{n-1}, a_n) = z_n$$

$a_1 \dots a_n$ akzeptiert $\Leftrightarrow z_n \in E$

M Automat, dann $T(M) = \{w \in \Sigma^* \mid w$ führt am Ende zu Zustand aus $E\}$

↑ Sprache von M.

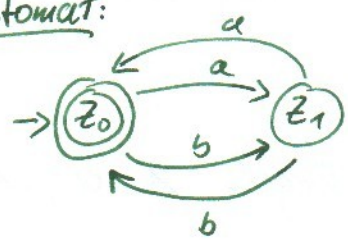
Automat für: $\cdot L = \emptyset$ Dann $E = \emptyset$ z.B. → (z0) ↻ Σ

$\cdot L = \{\varepsilon\}$ → (z0) —$a \in \Sigma$→ (z1) ↻ $a \in \Sigma$

$(\varepsilon \in L \Leftrightarrow z_0 \in E)$

10.04.2018

Beispielautomat:

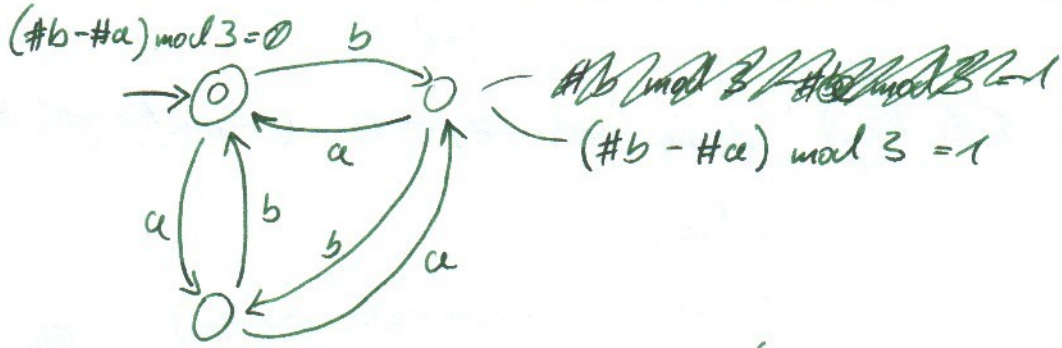


$T(M) = ?$

ε, aa, ab,
bb, ba, ...

aabb
aabbbb
aabbab

⇒ alle Worte über
$\{a, b\}$ mit gerader Länge



(#b - #a) mod 3 = 2

$\left(\begin{array}{l} (-1) \bmod 3 = 2 \\ (-2) \bmod 3 = 1 \end{array} \right)$

$$T(M) = \{ w \in \{a,b\}^* \mid (\#b - \#a \text{ in } w) \bmod 3 = 0 \}$$

Satz: M endlicher Automat, dann $T(M)$ ist regulär.

Beweis:

Baue Regeln $A \to a$, $A \to aB$

$S = z_0$ Regeln:

-  $\delta(z, a) = z'$ $\rightsquigarrow z \to a z'$
-  $z' \in E$ $\delta(z, a) = z'$ $\rightsquigarrow z \to a$

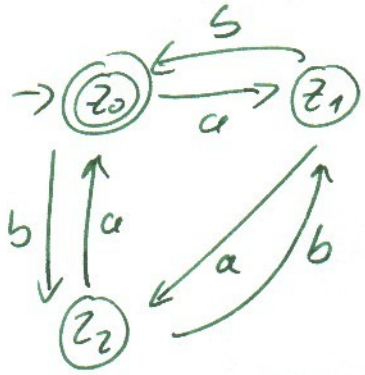
Falls  , ~~dann auch $z_0 \to \varepsilon$~~ (ε-Sonderregel braucht)

$S' \to \varepsilon$

$S' \to \ldots$ (rechte Seite von z_0)

∎

Bsp.:



$S' \to \varepsilon$

$S' \to a z_1 \mid b z_2$

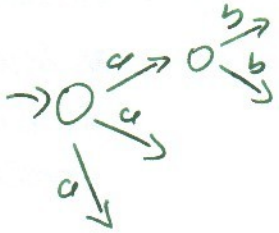
$z_0 \to a z_1 \mid b z_2$

$z_1 \to a z_2 \mid b z_0 \mid b$

$z_2 \to a z_0 \mid b z_1 \mid a$

$(\#a + \#b) \bmod 3 = 0$

Nichtdeterministischer Automat



- Q Menge der Zust.
- $S \subseteq Q$ Menge d. Startzustände
- $E \subseteq Q$ Menge d. Endzustände
- $\delta(z, a) \to \mathcal{P}(Q)$

  ↳ Potenzmenge von Q

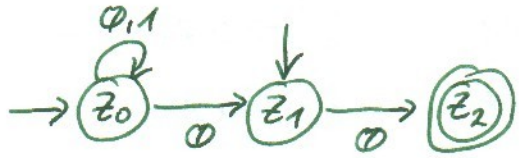
- $T(M)$: $a_1 \dots a_n \in T(M)$

  $\Leftrightarrow$

  Es gibt eine Rechnung der Art

Bsp.: $L = \{ w \in \{0,1\}^* \mid w \text{ endet auf } 00 \text{ oder } w = 0 \}$

nichtdet. Automat dazu:

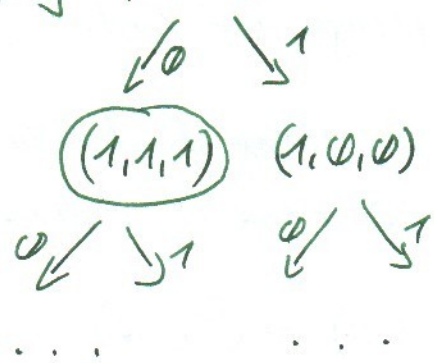


Wie nichtdet. Automat als Algorithmus?

↳ Merken uns alle Zustände, in denen der Automat sein kann. (etwa Bitvektor)

↳ $(0,1,1) \hat{=}$ Können im Nichtdet. Automat z_1 oder z_2 erreichen.

Aufzug: $(1,1,0)$

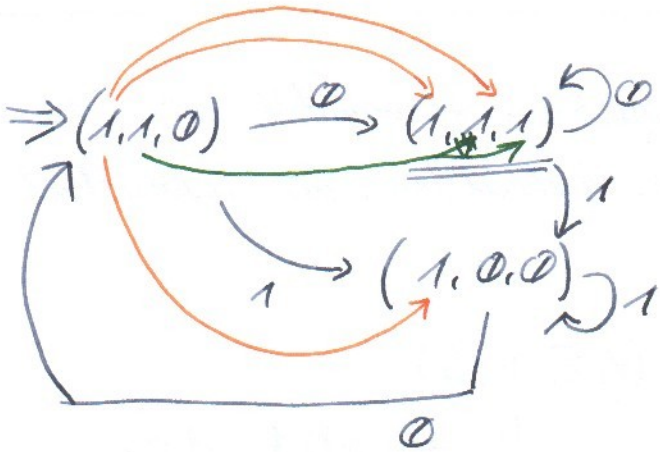


. . . . . .

16.01.2018

Konstruktion des DEA zum NEA aus dem Beispiel.

Startzustand des det. Automaten: (1,1,0)

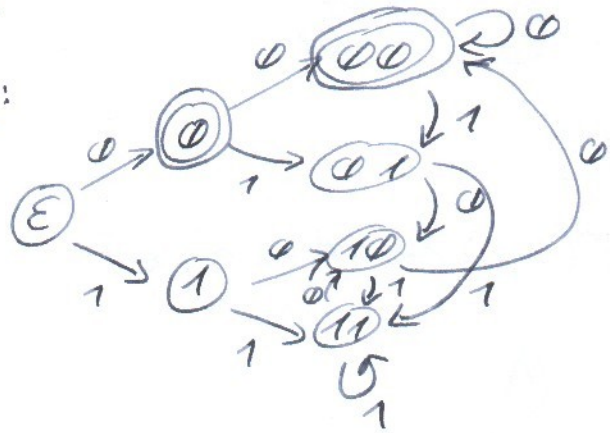


Endzustand: (1,1,1).

(andere Teilmengen der Zustände sind nicht erreichbar!)

- Konstruktion eines deterministischen Automaten von Hand:

z.B.:



allgemein: n det. n Zustände ⇒ det. 2^n Zustände!

⇒ Potenzmengenkonstruktion!

Satz: (Rabin, Scott)

Wird L von einem nichtdet. Automaten erkannt, dann auch von einem deterministischen.

Beweis: Potenzmengenkonstruktion ∎

Satz: Zu jeder regulären Grammatik gibt es einen nichtdet. Automaten, der äquivalent ist.

Beweis: Gegeben: $G = (V, \Sigma, P, S)$

$\hookrightarrow A \to aB$ & $A \to a$

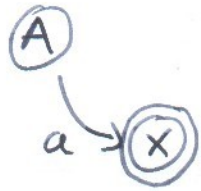
Automat $M = (Z, \Sigma, \delta, S', E)$

- $Z = V \cup \{X\}$ ‖ - $E = \{X\}$

$B \in \delta(A, a) \Leftrightarrow (A \to aB) \in P$

~~$S' = \{S\}$~~

$X \in \delta(A, a) \Leftrightarrow (A \to a) \in P$



- $S' = \{S\}$

bis hierher: ε wird nie erkannt.

- falls $\varepsilon \in L$: dann $E = \{S, X\}$
  - da dann S nicht auf rechten Regelseiten auftritt, hat S auch keine eingehenden Kanten.

16.04.
2018

Damit ist folgendes gezeigt:

→ DEA → reguläre Grammatik → NEA ┐

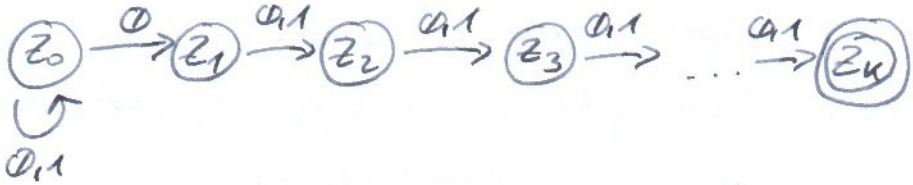
Das heißt reguläre Sprachen und NEA/DEA sind äquivalent.

Satz: (Man braucht ~~eventuell~~ im Allgemeinen exponentiell viele Zustände beim Übergang nichtdet. → det.!)

Sei $L_k = \{ x \in \{0,1\}^* \mid$ Das k-t letzte Zeichen von x ist 0$\}$.

Es gibt nichtdet Automaten mit k+1 Zuständen. Jeder deterministische Automat hat $\geq 2^k$ Zustände.

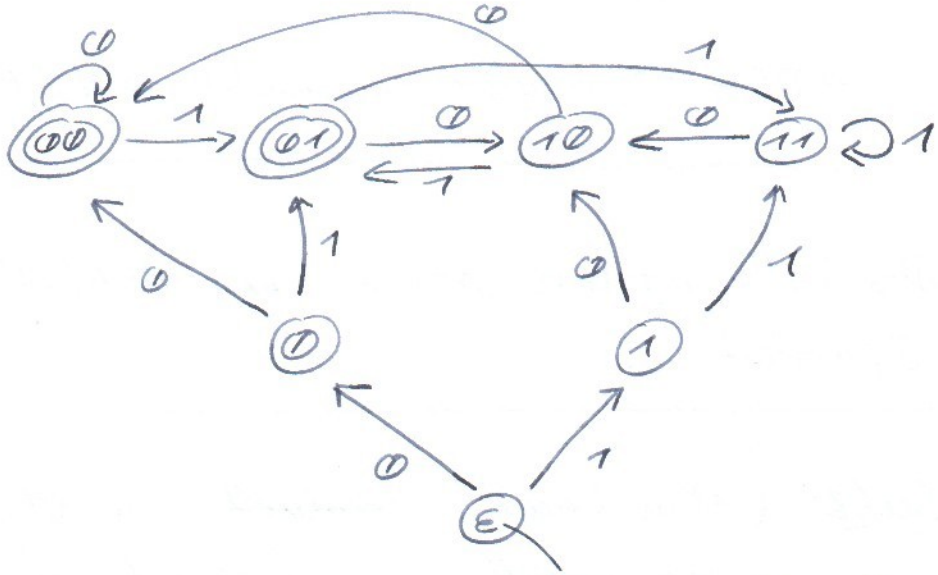
Beweis:



⇒ nichtdet. Automat

- Wie bekommt man einen det. Automat dafür?

Bsp.: det. Automat durch merken der letzten K Zeichen. K=2.



(zum merken der letzten ≤2 gelesenen Zeichen.)

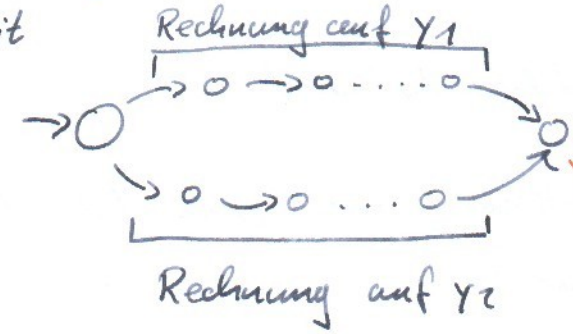
· weiter im Beweis.

Annahme: Haben det. Automaten ~~mit~~ M mit $\leq 2^K$ Zuständen.

Zeigen $T(M) \neq L_K$.

- Betrachten alle Worte $y \in \{0,1\}^K$. Das sind genau 2^K viele.

Dann gibt es $y_1 \neq y_2$ (jeweils $\in \{0,1\}^K$) mit



(Da $\leq 2^K$ Zustände)

gleicher Zustand.

16.04.2018

Angenommen (als Bsp.)

$y_1 = \ldots\ldots b\,a$

$y_2 = \ldots\ldots b'a$

unterscheiden sich im vorletzten Zeichen.
etwa $b = 0$, $b' = 1$

Verlängern (um $k-2$ Zeichen)

$y_1' = \ldots\ldots ba\overbrace{0\ldots0}^{k-2}$ $\Rightarrow y_1 \in L$,

$y_2' = \ldots\ldots b'a\underbrace{0\ldots0}_{k-2 \text{ Stück}}$ $y_2 \notin L$

Am Ende der gleiche Zustand (da det. A.), also wird L_k nicht erkannt!

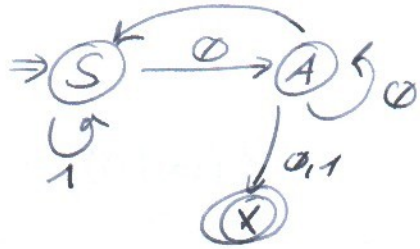
$\Rightarrow$ Allgemeiner Fall geht genauso. ∎

Abschließendes Beispiel: Reguläre Grammatik für L_2.

$S \to 0A \mid 1S$

$A \to 0 \mid 1 \mid 0A \mid 1S$

NEA dazu:



Reguläre Ausdrücke

L_2: Vorne was beliebiges, vorletztes Zeichen 0, Letztes Zeichen egal.

$(0|1)^* 0 (0|1)$

L_3: $(0|1)^* 0 (0|1)(0|1)$

Syntax der regulären Ausdrücke über Σ:

- ϕ ist reg. Ausdruck (Leere Menge)
- ε ist reg. Ausdruck (leeres Wort)
- a ist reg. Ausdruck für alle $a \in \Sigma$
- Ist α reg. Ausdruck, dann auch $(\alpha)^*$
- Sind α, β reg. Ausdrücke, dann auch $(\alpha|\beta)$, $\alpha.\beta$

Semantik: $L(\phi) = \phi$, $L(\varepsilon) = \{\varepsilon\}$, $L(a) = \{a\}$

$$L((\alpha)^*) = (L(\alpha))^* = \{ w_1 w_2 \dots w_k \mid k \geq 0, w_j \in L(\alpha) \text{ für } 1 \leq j \leq k \}$$

$$L((\alpha|\beta)) = L(\alpha) \cup L(\beta)$$

$$L(\alpha.\beta) = \{ w_1 w_2 \mid w_1 \in L(\alpha), w_2 \in L(\beta) \}$$

17.04.2018

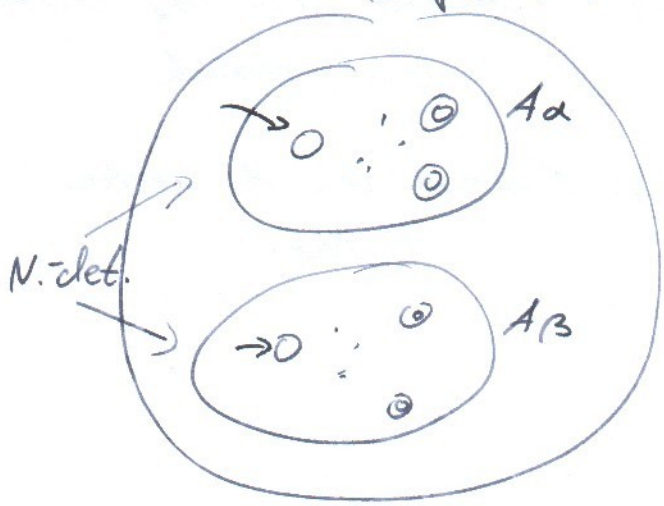
Satz (Kleene):

Sprachen, die durch reguläre Ausdrücke beschreibbar sind, stimmen mit den regulären Sprachen überein.

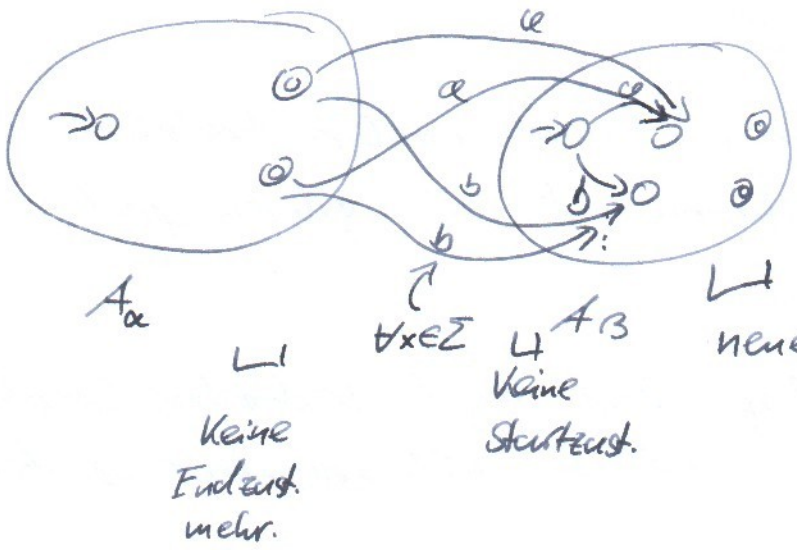
Beweis: Zeige: Durch reg. A. beschreibbare Sprachen $\Leftrightarrow$ von NFA erkennbare Sprachen

„$\Rightarrow$" reg A. $\rightarrow$ NFA:

- ϕ: → ○ Kein Endzustand
- ε: → ◎ Start = Endzustand, sonst keine Transitionen
- a: → ○ —a→ ◎
- $(\alpha | \beta)$ Haben det. Automaten ~~für~~ A_α und A_β für $L(\alpha)$ und $L(\beta)$



• $\alpha . \beta$, A_α , A_β gegeben



Wird w erkannt, dann hat w die Form $u.v$ mit $u \in L(\alpha)$, $v \in L(\beta)$ und wird auch erkannt.

Geht so, falls $\varepsilon \notin L(\alpha)$, $\varepsilon \notin L(\beta)$.

$\varepsilon \in L(\alpha)$, $\varepsilon \notin L(\beta)$: ✓

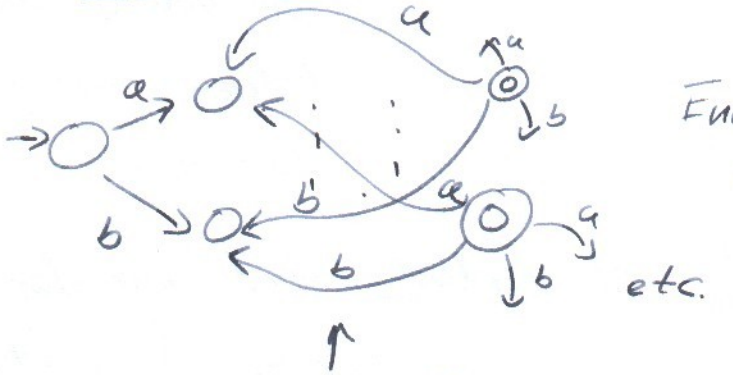
$\varepsilon \notin L(\alpha)$, $\varepsilon \in L(\beta)$: Endzust. von A_α bleibe Endzustände

$\varepsilon \in L(\alpha)$, $\varepsilon \in L(\beta)$: Endzust. von A_α bleibe Endzust.

17.04.
2018

• ~~(α)~~ $(\alpha)^*$ Haben A_α.

A_α:



Endzust. bleiben!

etc.

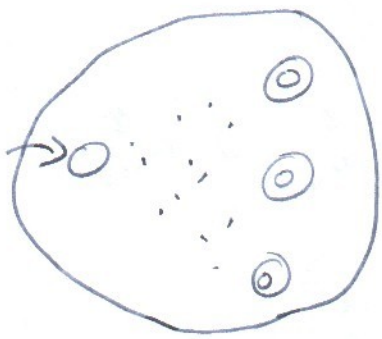
neue Übergänge

w wird akzeptiert $\Leftrightarrow$ $w = w_1 \dots w_n$,
~~dabei~~ wobei $w_i \in L(\alpha)$,
$n \geq 1$

falls ~~Startzust~~ $\varepsilon \in \alpha$ passt das so,
sonst noch zusätzlichen Startzust.
f. „ε"
$\rightarrow$◎ dazu.

~~■~~

„$\Leftarrow$" gegeben: DFA $\Rightarrow$ Konstruktion eines reg. A. dazu.



Sei $Z = \{1, \dots, n\}$

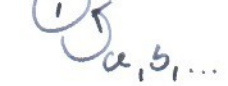
Startzustand ist 1.

im Automaten:

$R^{\emptyset}_{i,j}$ = Sprache der Worte, die von (i) nach (j) führen ohne Zwischenzustand.

$R^{\emptyset}_{i,j}$ ist endlich und von der Art $\{a, b, c, \dots\}$, $i \neq j$

$R^{\emptyset}_{i,i}$ enthält ε und eventuell noch weitere Zeichen

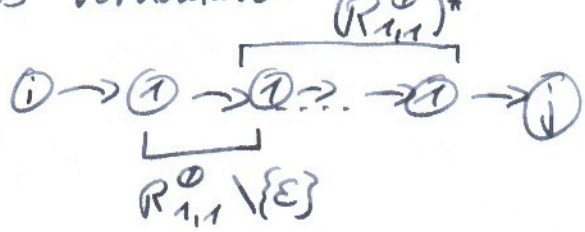
(z.B. 

)

⇒ Dafür haben wir reguläre Ausdrücke!

jetzt Mengen vergrößern / zusammenbauen.

$R^{1}_{i,j}$ = Menge der Worte von (i) nach (j), wo als Zwischenzustand nur (1) oder nichts vorkommt.

z.B. 

$$R^{1}_{i,j} = R^{\emptyset}_{i,1} \cdot (R^{\emptyset}_{1,1})^* \cdot R^{\emptyset}_{1,j} \cup R^{\emptyset}_{i,j}$$

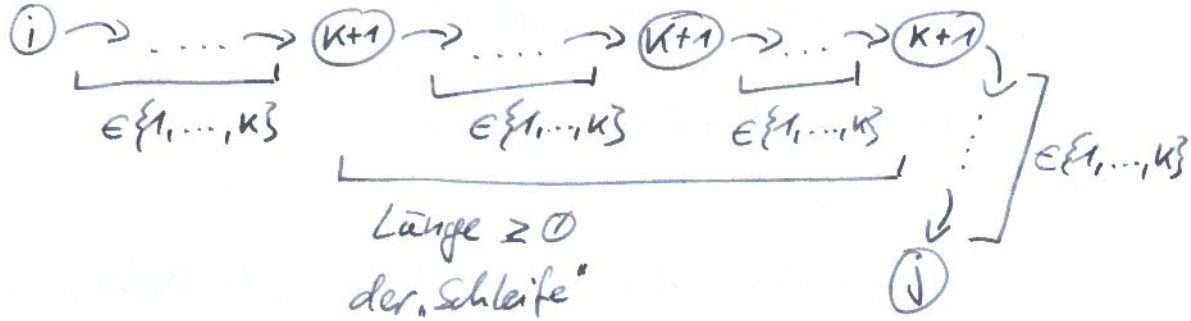
(gilt für alle i, j)

⇒ Haben reg. ~~Ausdrücke~~ Ausdruck für $R^{1}_{i,i}$.

17.04.
2018

$R_{i,j}^{k+1}$ = Worte von (i) nach (j) mit Zwischen-Zuständen ~~in~~ $\in \{1, \dots, k+1\}$.

$(k+1 \leq n)$ $\quad 0 \leq k \leq n-1$, i, j beliebig.



$$R_{i,j}^{k+1} = R_{i,j}^{k} \cup R_{i,k+1}^{k} \cdot \left(R_{k+1,k+1}^{k}\right)^{*} \cdot R_{k+1,j}^{k}$$

$$L(M) = \bigcup_{j \in E} R_{1,j}^{n}$$

Sprache des Automaten

■

Größe der regulären Ausdrücke?

$k=0$, dann $O(1) \leq |\Sigma|$

$k=1$ $\quad 4 \cdot |\Sigma|$

$k=2$ $\quad 4 \cdot 4 \cdot |\Sigma|$

$\vdots$

$k=n \rightsquigarrow \geq 4^n \cdot |\Sigma|$

Vergrößerung bei Übergang zu anderer Beschreibungsform:

NFA → DFA: exponentiell

DFA → reg. A: exponentiell

Typ 3 Gr. → NFA: linear

NFA → Typ 3 Gr.: linear

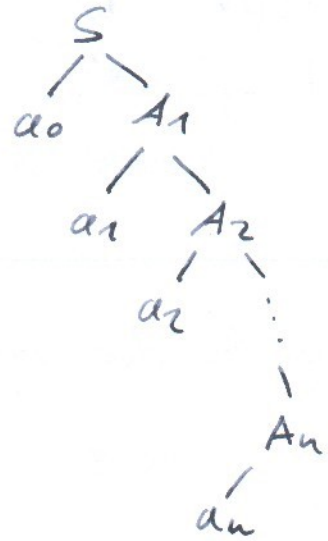
- Grammatiken $A \to aB$, $A \to a$ (Linkslinear)
  Automaten, reguläre Ausdrücke

Frage: Was ist mit $A \to Ba$, $A \to a$ „rechtslinear"?

Regeln: $S \to a_0 A_1$

$A_1 \to a_1 A_2$

⋮



$a_0 a_1 \dots a_n$

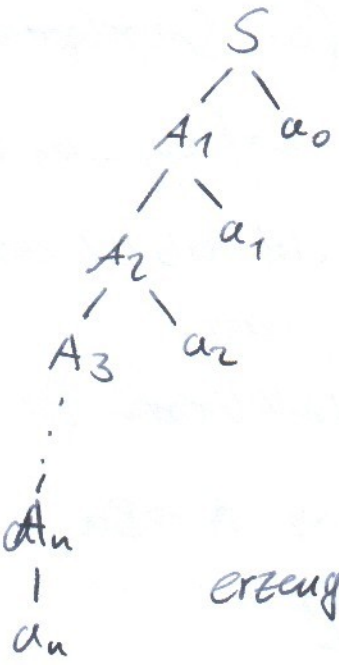
⇒ rechte Seiten umdrehen

$S \to A_1 a_0$

$A_1 \to A_2 a_1$

$A_2 \to A_3 a_2$

$\vdots$

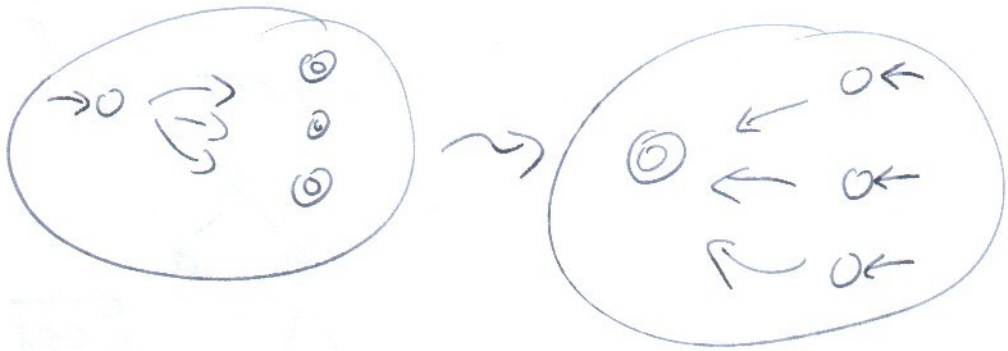


erzeugt $a_n \dots a_2 a_1 a_0$

Spiegelwort zu oben.

Haben L regulär. Dann Spiegelsprache auch regulär.

Automat L für M



Leeres Wort?

23.04. 2018 Gegeben Linkslineare Grammatik für L, dann rechtslineare Grammatik für Spiegelsprache $\overleftarrow{L}$

$$\overleftarrow{L} = \{a_n \dots a_1 \mid a_1 \dots a_n \in L\}$$

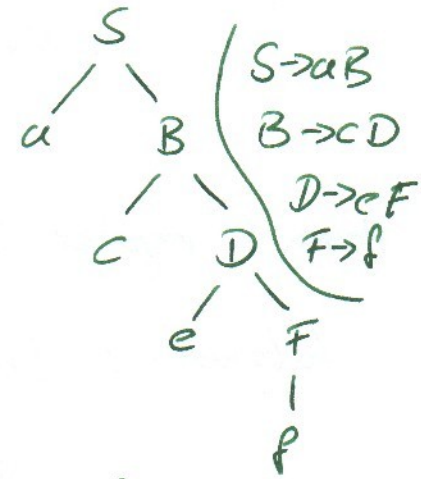
. Ist $ab \in L \Leftrightarrow ba \in \overrightarrow{L}$

usw.

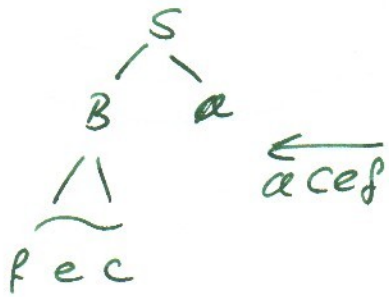
Transformation der Linkslinearen für L

$$A \to aB \quad \rightsquigarrow \quad A \to Ba$$

Rechtslinear für $\overrightarrow{L}$



a,cef

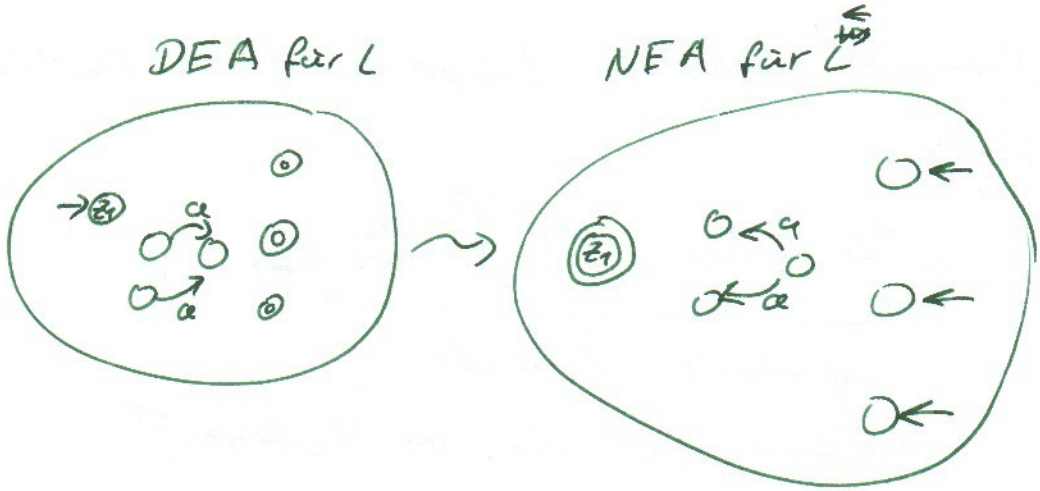


23.04.2018

Starten bei L Typ3.

Dann $\overleftarrow{L}$ auch Typ3.

DEA für L „umdrehen", zu NEA.



Dann für Spiegelsprache $\overleftarrow{L}$ Typ 3 Grammatik.

Dann ist für $L = \overleftarrow{L}$ rechtslineare Gr. auch

$A \to aB \rightsquigarrow A \to Ba$ gegeben.

Wie kann man sehen, dass eine Sprache nicht regulär ist?

Das heißt, dass es dafür keine Typ3-Grammatik / keinen DFA/NFA gibt!

⇝ Pumping Lemma

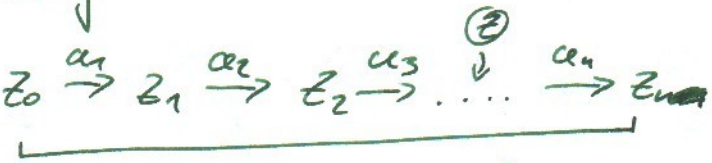
Pumping Lemma

Bsp.: $L = \{a^m b^m \mid m \geq 0\}$

DEA beliebig über $\Sigma = \{a, b\}$
Zustände n

Betrachte Wort der Länge n. $(a_1 \ldots a_n \in L)$

Rechnung:



irgendwelche Zustände,
mind. einer doppelt, da $n+1$ Positionen!

Betrachten jetzt $w \in L$ der Länge $\geq n$

$\underbrace{a \ldots a}_{m \text{ St.}} \underbrace{b \ldots b}_{m \text{ St.}}$ $\quad m \geq \frac{n}{2}$

~~je nach~~ je nachdem, wo der wiederholte Zustand liegt, werden auch Worte der Art

- $\underbrace{a \ldots a}_{\#a > \#b} \underbrace{ab \ldots b}$

oder $a \ldots a \underbrace{a \ldots b}\, \underbrace{a \ldots b}\, b \ldots b$

falsche Struktur

- $\underbrace{a \ldots a}_{\#a < \#b} \underbrace{b \ldots b}$

$\Rightarrow$ Ein Automat, der die Sprache erkennt, kann nicht existieren!

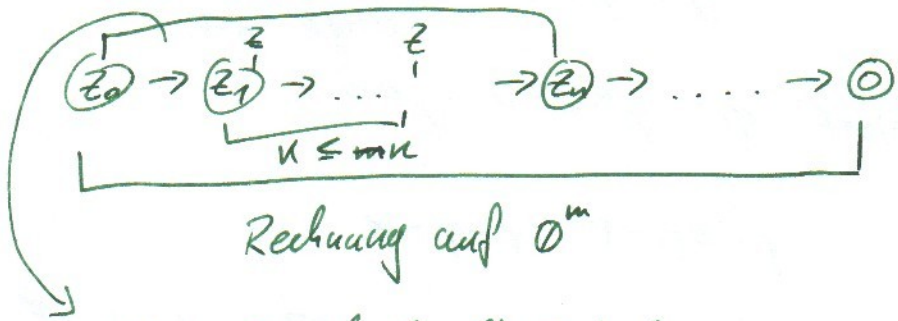
23. Okt. 2018

anderes Beispiel:

$L = \{ 0^m \mid m = 0, 2, 4, 9, \ldots ;\ m \text{ Quadratzahl} \}$

ist nicht regulär!

· DEA mit n Zuständen. Sei $m = n^2 > n$, 0^m wird erkannt.



in diesem Bereich doppelter Zustand.

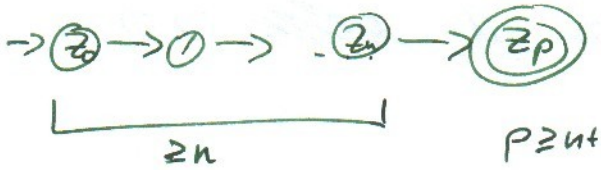
also wird auch 0^{m+k} für ein $1 \leq k \leq n$ und ebenso 0^{m-k}, 0^{m+2k}, ... erkannt.

$m+k$ ist keine Quadratzahl, denn

$n^2 < n^2 + k < (n+1)^2 = n^2 + 2k + 1$.

$L = \{ 0^p \mid p \text{ Primzahl} \}$

DEA, n Zust., $p \geq n+2$ Primzahl $0^p \in L$



also auf den ersten n Zeichen haben wir eine Schleife der Länge $\leq n \leq p-2$

also:

$$\emptyset^p = \underbrace{\emptyset^q \emptyset^k}_{q+k \leq n} \emptyset^{p-q-k}$$

Schleife auf $\emptyset^k$ $\quad k \geq 1$

~~$\emptyset^p \emptyset^{p-q-k}$~~ Länge ~~$p-k$~~

$\emptyset^q \emptyset^{p-q-k}$ Länge $p-k$

$\vdots$

Wort $\emptyset^q \cdot \emptyset^{k(p-k)} \cdot \emptyset^{p-q-k}$

$$q + kp - k^2 + p - q - k$$

$$= k(p-k-1) + p \quad \text{?? nachrechnen}$$

$\rightsquigarrow$ keine Primzahl.

Pumping Lemma:

Ist $L \subseteq \Sigma^*$, L regulär. Dann existiert ein $n \in \mathbb{N}$, so dass für alle $x \in L$ mit $|x| \geq n$, ~~so~~ es gibt Teilworte u, v, w von x, $x = uvw$ mit

(1) $|v| \geq 1$

(2) $|uv| \leq n$

(3) $uv^i w \in L$ für alle ~~$i \geq 0$~~ $i \geq 0$.

23.04.2018

Minimalautomaten

Definition (Endsprache)

Ist $L \subseteq \Sigma^*$, $w \in \Sigma^*$, dann ist die Endsprache von w bezüglich L:

$$L_w = \{v \in \Sigma^* \mid wv \in L\}$$

Bsp: $L = \{x \in \{0,1\}^* \mid x = y00 \text{ für ein } y \in \{0,1\}^*\}$

$L_\varepsilon = L$

$L_0 = \{0\} \cup L$

$L_1 = L$

$L_{00} = \{\varepsilon\} \cup L_0$

$L_{v0} = L_0$ (v endet mit 1)

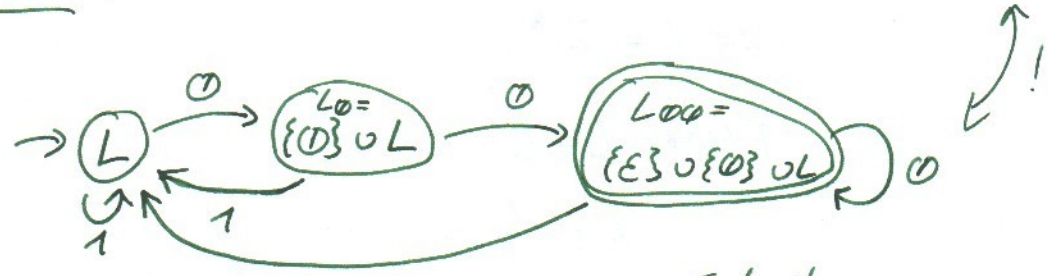
~~$L_{v0} = L_{00}$~~ $L_{v0} = L_{00}$ (v endet mit 0)

$L_{v1} = L$

$\Rightarrow$ 3 verschiedene Endsprachen!

($w \in L \Leftrightarrow \varepsilon \in L_w$)

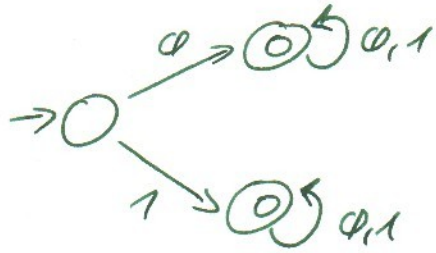
DEA?!



Endzust, wegen $\varepsilon \in L_{00}$!

Satz: L Typ 3 $\Rightarrow$ # verschiedener Endsprachen $\leq$ # Zustände eines DEAs für L

Bsp:



$L_\varepsilon = 0\,(0|1)^* \mid 1\,(0|1)^* = (0|1)(0|1)^*$

$L_0 = L_1 = (0|1)^*$

$\rightsquigarrow$ nur 2 versch. Endsprachen

Idee: Zustände mit gleichen Endsprachen zusammenfassen.

Satz: L Typ 3 $\Leftrightarrow$ endlich viele verschiedene Endsprachen.

Beweis: „$\Rightarrow$" sieht man direkt am DEA.

„$\Leftarrow$" Haben endlich viele versch. Endsprachen zu $L \in \Sigma^*$.

Etwa $L = L_\varepsilon$

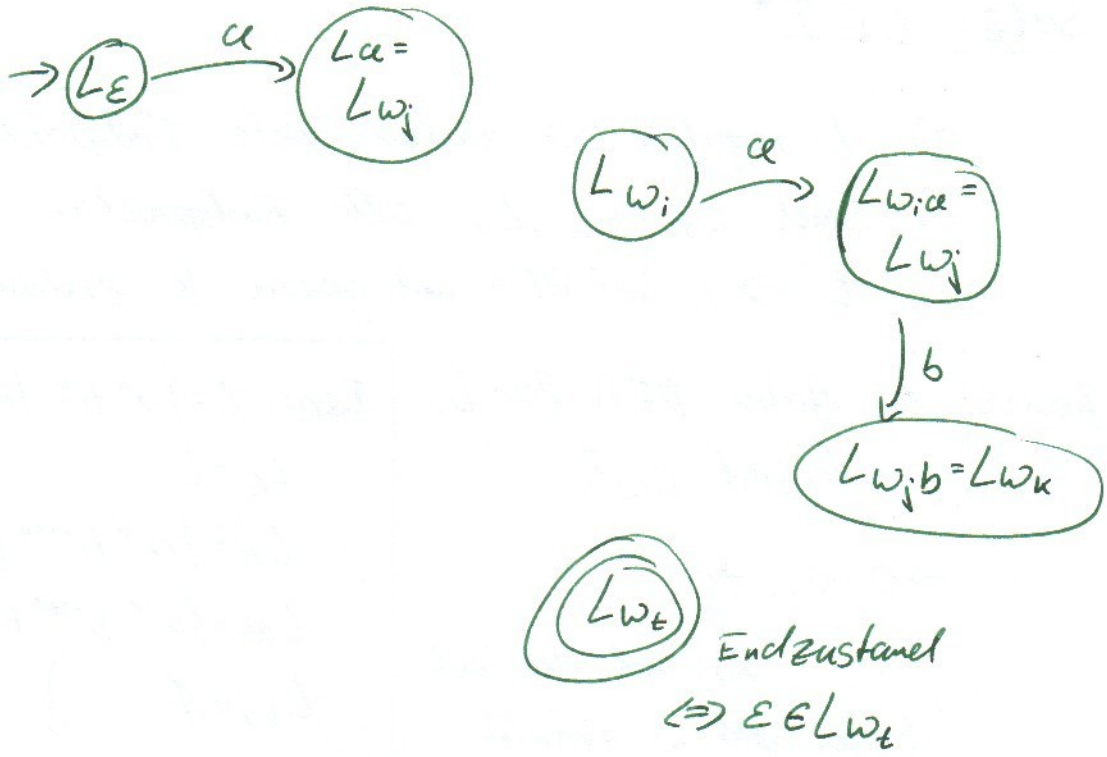
$L_1 = L_{w_1}$ ...

$L_n = L_{w_n}$ sind alle Endsprachen, alle verschieden.

Baue DEA.

$\Downarrow$

23.04.2018



24.04.2018

Beobachtung:

$$L_w = L_{w'} \Rightarrow L_{wa} = L_{w'a}.$$

$$\{x \mid wax \in L\} = \{ax \mid w(ax) \in L\}$$

$$= \{ax \mid w'(ax) \in L\}$$

$$= \{x \mid (w'a)x \in L\}$$

$$= \{x \mid ax \in L_w\}$$

$$= \{x \mid ax \in L_{w'}\}$$

nochmal:

Satz: $L \subseteq \Sigma^*$

a) L regulär $\Rightarrow$ endlich viele Endsprachen

b) Sind $L_1, L_2, \ldots, L_k$ alle Endsprachen von $L \Rightarrow L$ hat DEA mit genau k Zuständen.

Beweis: a) Haben DEA für L.

Was ist L_ω?

$\rightarrow 0 \rightarrow \ldots \rightarrow 0$

Zustand, der nach Lesen von ω erreicht wird.

L_ω = Sprache, die von diesem Zustand aus akzeptiert wird.

$\Rightarrow$ #Endsprachen von $L \leq$ #Zustände des DEA für L

Bsp.: $L = \{a^m b^m \mid m \geq 0\}$

$L_\varepsilon = L$

$L_a = \{a^m b^{m+1} \mid m \geq 0\}$

$L_{aa} = \{a^m b^{m+2} \mid m \geq 0\}$

$L_{ba} = \emptyset$ $\downarrow$

usw. ∞ viele

b) Konstruktion eines DEA aus den Endsprachen.

- Zustände ①, ②, …, ⓚ
- Endzustände: $\varepsilon \in L_i \Leftrightarrow$ ⓘ
- Startzustand: $L_\varepsilon = L$ (etwa $L_1 \rightarrow$ ①)
- Überführungsfunktion:

δ: ⓙ $\xrightarrow{a \in \Sigma}$ ⓛ

24.04.
2018

$\delta(j, a) = l \Leftrightarrow$ Ist $L_j = L_w$ für ein w
$\Rightarrow L_l = L_{wa}$

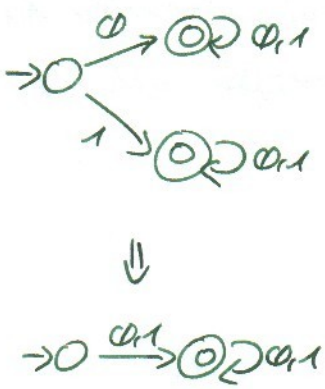
(Wichtig: $L_w = L_{w'} \Rightarrow L_{wa} = L_{w'a}$!)

Korrektheit: Vom Zustand j aus wird genau die Sprache L_j erkannt. Für alle j. ▮

(Induktion über die Wortlänge.)

Minimierung der Zustände eines DEA für L
(bis auf #Zustände = # Endsprachen)

Bsp:



$L_\varepsilon = L$

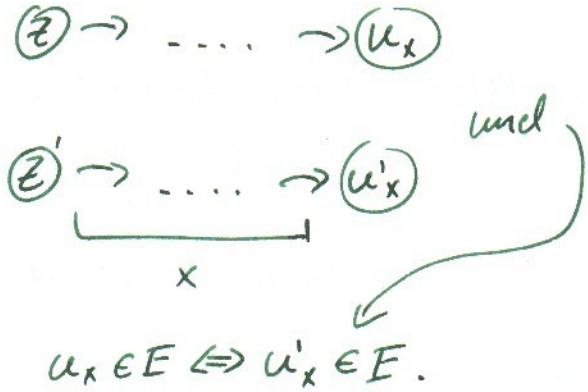
$L_0 = L_1 = L_v = \{0,1\}^*$
$v \neq \varepsilon$

Idee: Verschmelzen der Zustände, von denen aus die gleiche Sprache erkannt wird.

Problem: Wie sieht man das?

Wann können zwei Zustände z, z' verschmolzen werden?

Genau dann, wenn gilt: Für alle Worte $x \in \Sigma^*$ gilt:



$u_x \in E \Leftrightarrow u'_x \in E$.

- Wie lange x'e sind zu betrachten?

Sei n = #Zustände des zu minimierenden Automaten. Es reicht, um oben das zu prüfen, alle x mit $|x| \leq n^2$ zu überprüfen.

Sei $|x| \geq n^2$:

$z \to z_1 \to z_2 \dots \to u_x$

$z' \to z'_1 \to z'_2 \dots \to u'_x$

dann ein Paar von Zuständen zweimal! Dann rausschneiden, ~~und~~ d.h. ein Wort $\leq n^2$ führt zur selben Situation.

24.04.2018

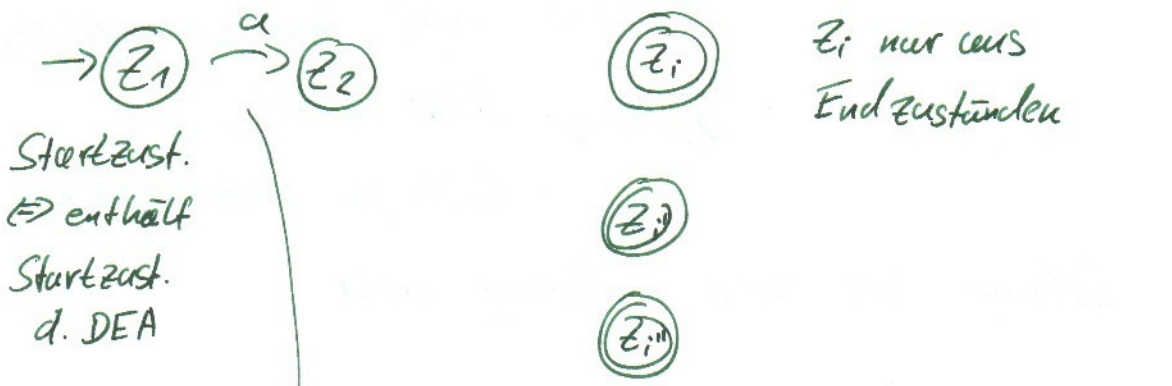
Algorithmus:

1) Stelle für jedes Zustandspaar (z, z') fest, ob die von z und z' erkannten Sprachen gleich sind.

2) Merke mir die Paare, wo das gleich ist. Dann fasse alle Zustände wo (z, z') gemerkt zusammen.

Haben Mengen von Zuständen

$$Z_1, Z_2, \ldots, Z_k \quad \text{disjunkt.}$$



$$\delta(z_1, a) = z_2 \Leftrightarrow \exists\, u \in z_1, v \in z_2 \text{ mit } \delta(u, a) = v \text{ im alten DEA.}$$

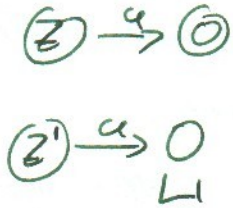
Zeit: $O(|\Sigma|^{n^2} \cdot n^2)$

Effizienterer Algorithmus, um zu sehen, ob von z, z' die gleiche Sprache erkannt wird.

- Suchen alle Paare (z, z') raus, die verschiedene Sprachen erkennen.

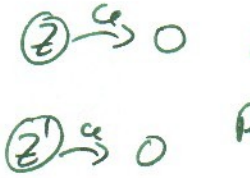
1. „Zeuge" der Länge 0, d.h. $\varepsilon \Leftrightarrow z \in E, z' \notin E$ markiert

2. „Zeuge der Länge 1, $a \Leftrightarrow$



auf markiertes Paar gestoßen

3. Für jedes Paar (z, z') das noch nicht markiert ist, teste ob



auf bereits markiertes Paar führt.

falls ja markiere (z, z')

Solange bis keine Änderung mehr.

Pro Runde: $O(|Z| \cdot n^2)$

Insgesamt: $O(n^4)$ (geht auch noch besser)

Haben jetzt alle Paare $\{z, z'\}$ von denen aus versch. Sprachen erkannt werden!

$\Rightarrow$ Zustände mit gleicher Sprache wie z: alle z' so dass $\{z, z'\}$ nicht markiert.

So weiter, bis alle Zustände hier behandelt sind.

(Disjunkte Mengen äquivalenter Zustände, Übergangsfunktion wie vorher!)

24.04.2018

Abschlußeigenschaften regulärer Sprachen

- Vereinigung nach ~~def~~ Def. RA
- Stern — " —
- Produkt — " —
- Komplement im DEA EZ und nicht-EZ vertauschen
- Schnitt → DEA auf Zustandspaaren oder De Morgan.

Entscheidbarkeit

Wortproblem: Linearzeit (DEA)

Leerheit: DEA ohne (erreichbaren) Endzustand

Endlichkeit: $\infty \Leftrightarrow \exists$ Wort mit Länge zwischen n und $2n$.

Schleife im DFA Graph.

Äquivalenz: $L_1 = L_2 \Leftrightarrow$ ~~L_2~~ auf Schnitt und Komplement ? zurückführen !

$$L_1 = L_2 \Leftrightarrow (L_1 \cap \bar{L_2}) \cup (L_2 \cap \bar{L_1}) = \phi$$

30. Okt. 2018

3. Kontextfreie Sprachen

(Grammatiken $A \to X$, $X \in (\Sigma \cup V)^* \setminus \{\varepsilon\}$, ε-Sonderregel.)

Satz: Lassen wir auch Regeln $A \to \varepsilon$, dann bleibt die Menge der Typ-2 Sprachen unverändert.

Beweis: **Idee:** Regel $B \to \dots A \dots$ bei $A \to \varepsilon$ wird ergänzt durch Regel

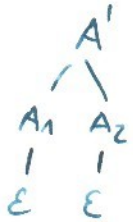
$B \to \dots \dots$ (↑ $A \to \varepsilon$)

(ε-Regeln direkt einbauen)

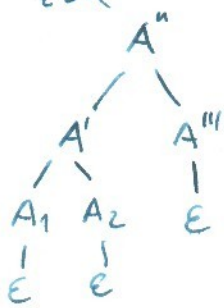
Problem: $B \to A$, $A \to \varepsilon$ gibt $B \to \varepsilon$
Wie endet das?

- Alle ~~Symbole~~ Symbole mit $A \Rightarrow \dots \Rightarrow \varepsilon$ bestimmen!

1. Alle Symbole A mit $A \to \varepsilon$ ist Regel.
2. Alle Symbole A' mit $A' \to$ „Satz von Symbolen aus (1)" ist Regel.
3. Alle Symbole, wo rechts Satz von Symbolen aus (1) oder (2) Regel ist.



z.B.



→ wiederholen solange bis Ende, nach ≤ |V| Schritten keine neuen mehr.

30.04.
2018

$\Rightarrow A, A', \dots, A'''$ alle Symbole, die zu ε führen.

In Regel $B \to \dots A \dots A' \dots A'' \dots$ ergänzen (alle Varianten)

$B \to \dots \varepsilon \dots A' \dots A'' \dots$

$B \to \dots A \dots \varepsilon \dots A'' \dots$

usw.

außer $S \to \varepsilon \Rightarrow$ hinterher $S' \to \varepsilon, S \to S'$. ∎

Kettenregeln sind $A \to B$.

<u>Satz</u>: Grammatik ohne Regeln $A \to \varepsilon$. Dann lassen sich alle Kettenregeln eliminieren. Ausnahme: ε-Sonderregel.

<u>Beweis</u>: <u>Idee</u>: $C \to \dots A_1 \dots A_2$, $A \to B$

zusätzlich

$C \to \dots B \dots A_2$ usw., dann Kettenregel weg.

<u>Problem</u>: $C \to A$, $A \to B$, ~~$C \to B$~~

$C \to B$ neue Kettenregel.

$D \to C$ $D \to A$ $D \to B$

Kreis!

$A \to D$

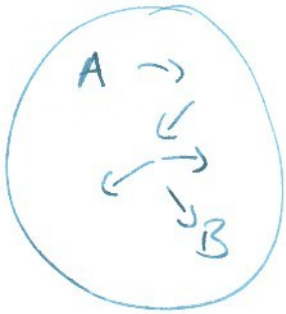
$A \to A$

$A \to B$

Wie endet das???

- Graph der Kettenregeln, gerichtet.

⇒ Starke Zusammenhangskomponenten



⇔ Es gibt Ableitungsschritte der Art

$A \Rightarrow \ldots \Rightarrow B \Rightarrow \ldots \Rightarrow A$

⇒ alle Symbole aus der Starken ZHK vereinigen.

⇒ z.B. A verwenden, alle rechte Seiten vereinigen.

, alle vorkommen von Symb. der Komp. durch A ersetzen.

⇒ Jetzt: Graph der Kettenregeln ist kreisfrei.

⇒ Topologisch sortierbar.

⇒ von hinten her einsetzen.

30.04.2018

Normalformen: Chomsky-Normalform (CNF)

$A \to BC$ $\quad B, C \in V$, ε-Sonderregel

$A \to a$

Satz: Falls ε nicht dabei, bekommen wir Regeln der Art $A \to BC$, $A \to a$.

Beweis: Kettenregeln weg, ε-Regeln weg.

Transformation: $A \to BCD$

$\Downarrow$

$A \to \not{A} CD$

A_{CD}

$A \to B A_{CD}$

$\hookrightarrow$ neues Symbol

$A \to BCa \Rightarrow A \to BCA_a, \Rightarrow \ldots$

$A_a \to a$

(für längere rechte Seiten dann analog.) ∎

CYK-Algorithmus, Wortproblem für Kontextfreie Sprachen

gegeben: L kontextfrei. Wortproblem: $w \in \Sigma^*$, $w \neq \varepsilon$.
$w \in L$?

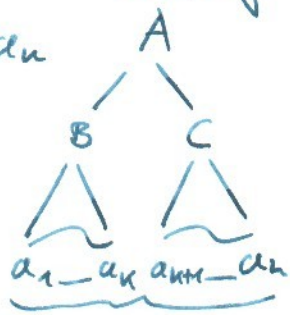
$n = |w|$, dann $O(n^3)$!
↙ wie geht das?

Haben Grammatik für L in CNF:

zuerst: Rekursiver Algorithmus.

```
Test(A, w)        A ist Nichtterminal, gibt es Ableitung A ⇒ ... ⇒ w.
if |w| = 1 then
    if Regel A → w then
        return wahr
    else
        return falsch
    end if
end if
// hier ist |w| ≥ 2, w = a1 a2 ... an
for jede Regel A → BC do
    for i = 1 to n-1 do
        if Test(B, a1 ... ai) ∧ Test(C, ai+1 ... an) then
            return wahr
        else
            return falsch
        end
    end
end
return falsch.
```

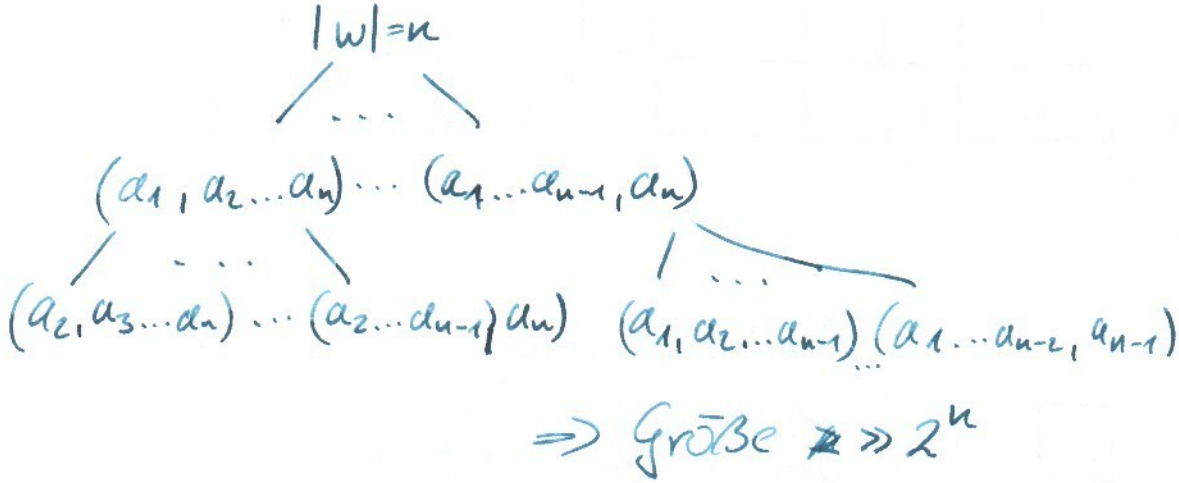
Ableitung:



30.04.2018

Laufzeit ? ⇒ Exponentiell

Aufrufbaum z.B.



⇒ Größe $\gg 2^n$

Aufrufe der Art

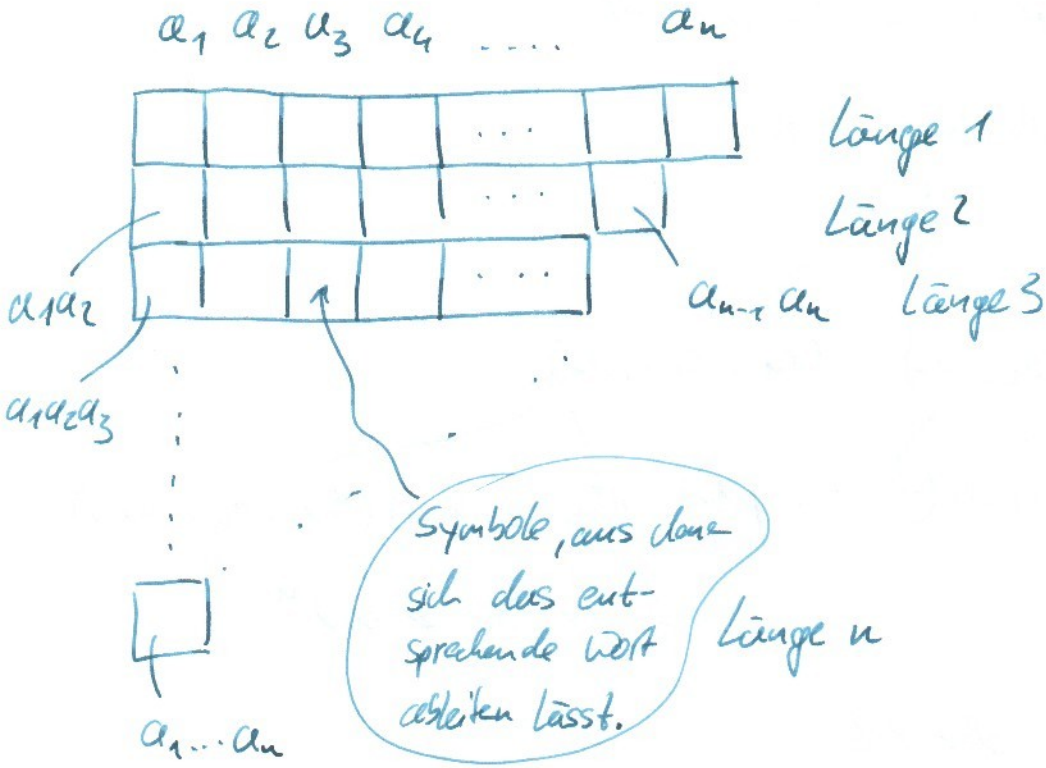
$Test(A, \underbrace{a_i \dots a_{i+k}}_{\text{zusammenhängende Teilstücke}})$ $k \geq 0$

$\binom{n}{2} + n$ verschiedene Aufrufe $O(n^2)$

$\frac{n(n+1)}{2}$

⇒ Dynamische Programmierung, Tabelle ausfüllen!

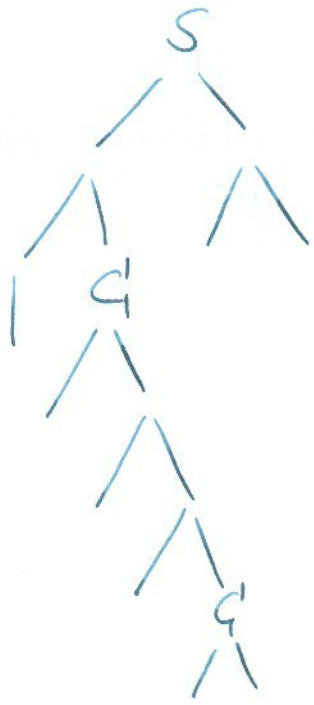
Tabelle:



$O(n^3)$

Pumping Lemma, L nicht Kontextfrei

Grammatik in CNF, Ableitungsbaum



Weg von der Länge ≥ # Nichtterminale

⇒ ein NT 2 mal auf dem Weg

⇒ Bedeutung auf Ebene der Ableitung, welche Worte auch ableitbar?

07.05.
2018

n = # Nichtterminale

Ableitungsbaum, 1 Nichtterminal 2x

Dann ist $uv^iwx^iy \in L$ für alle $i \geq 0$.

Satz: (Pumping Lemma für kontextfreie Sprachen)

Sei L kontextfrei. (D.h. haben Grammatik in $CNF_{\frac{3}{2}}$ mit n_a Nichtterminalen.)

Dann gibt gibt es eine Zahl $n \in \mathbb{N}$ ($n = 2^m$)

so dass für alle $z \in L$, $|z| \geq n$, gilt:

Wir können z aufteilen als $z = uvwxy$ mit folgenden Eigenschaften:

↓

- $|vwx| \leq n$ (im Ableitungsbaum von unten zum 1. doppelten NT gehen)
- $|vx| \geq 1$, d.h. $vx \neq \varepsilon$

  $(w \neq \varepsilon)$
- $uv^i wx^i y \in L$ für alle $i \geq 0$

Beweis: Sieht man am Ableitungsbaum bei Grammatik in Chomsky-Normalform. ∎

Beispiel: $L = \{a^m b^m c^m \mid m \geq 1\}$ nicht kontextfrei ☹

Sei n Zahl aus dem P.L. Betrachten

$$z = a^n b^n c^n \in L$$

Nach PL: Es gibt Zerlegung $z = uvwxy$ mit
$uv^i wx^i w \in L$ für alle $i \geq 0$.

So eine Zerlegung gibt es nicht, also nicht Typ 2.

07.05.
2018

$L = \{0^p \mid p \text{ Primzahl}\}$

n Pumping Lemma Zahl

$p \geq n$ Prim.

$0^p = u\,v\,w\,x\,y$ (unter $v\,w\,x$: $\leq n \leq p$)

$|vx| = L \geq 1$

$L \neq n$

$|uwy| = K \geq 1$, da $w \neq \varepsilon$

$K + h \cdot L$, $h \geq 0$ muß Primzahl sein.

für $K \neq 1$ geht $h = K$

$\Rightarrow K + K \cdot L = K(L+1)$ ist nicht Prim!

für $K = 1$ geht $h = (p+1)$

$\Rightarrow 1 + (p-1) \cdot L$

$= 1 + (p-1)^2$

$= p^2 - 2p + 2$

$= p(p-2) + 2$

(oder $\Rightarrow$ p größer machen, damit $K > 1$)

$K = 1 \Rightarrow u = y = \varepsilon$

$|w| = 1 \quad |vx| = p - 1$

$1 + (p-1) \cdot \underbrace{(p+1)}_{= n} = 1 + p^2 - 1$

$= p^2$

Abschlußeigenschaften:

(für kontextfreie Sprachen) L_1, L_2 Typ 2

Abgeschlossen unter:

$L_1 \cup L_2$ (Vereinigung)
$L_1 L_2$ (Produkt) sieht man an
L_1^* (Stern) Grammatik

nicht abgeschlossen unter:

$L_1 \cap L_2$ (Schnitt)
$\overline{L_1}$ (Komplement)

Beweis: (Schnitt)

$L_1 = \{a^m b^m c^k \mid m \geq 1, k \geq 1\}$

$L_2 = \{a^k b^m c^m \mid m \geq 1, k \geq 1\}$

Sind beide kontextfrei und

$L_1 \cap L_2 = \{a^m b^m c^m \mid m \geq 1\}$

ist nicht kontextfrei.

07.05.
2018

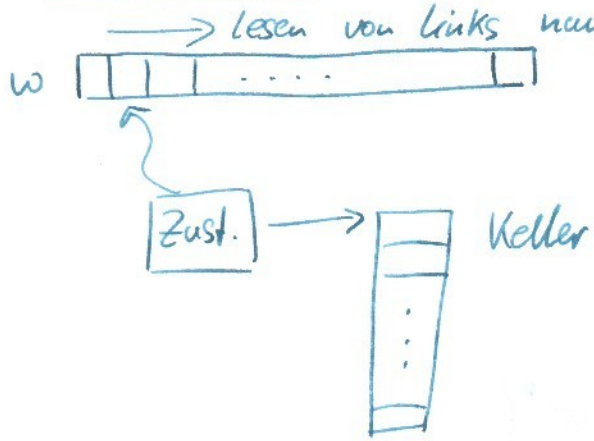
Komplement:

$$A \cap B = \neg(\neg(A \cap B)) = \neg(\neg A \cup \neg B)$$

Kann nicht gehen, da kein Abschluß unter Schnitt.

⇒ nicht kontextfrei

Kellerautomat



Z : Zustände

Σ, Γ : Eingabe- u. Kelleralphabet

$$\delta : (\Sigma \cup \{\varepsilon\}) \times \Gamma \to Pe(Z \times \Gamma^*)$$

$\# \in \Gamma$ Kellerstartsymbol

z_0 : Startzustand

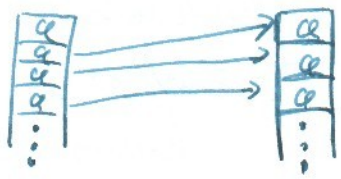
- Akzeptiert bei leerem Keller und Eingabe fertig gelesen.

Bsp.: Automat für $L = \{a^n b^n \mid n \geq 1\}$

- $(z_0, a, \#) \to (z_0, a)$

- $(z_0, a, a) \to (z_0, aa)$

- $(z_0, b, a) \to (z_1, \varepsilon)$



Bsp.: $L = \{w\overleftarrow{w} \mid w \in \{a,b\}^*\}$

08.05.
2018

Bsp.: Automat für $w\tilde{w}$

$S \to aSa \mid bSb \mid \varepsilon$

$\delta(z_0, a, \#) = (z_0, a) \quad \forall a \in \Sigma$

$\delta(z_0, a, b) = (z_0, ab) \quad a, b \in \Sigma \quad a = b$ mögl.

$\delta(z_0, a, a) = (z_1, \varepsilon)$

$\delta(z_1, a, a) = (z_1, \varepsilon)$

Kontextfreie Grammatik ⇝ Kellerautomat (nichtdet.)

(Linksableitung reicht)

$S \Rightarrow^* \underbrace{abc}_{\text{gelesen}} \underbrace{Bd}_{\text{im Keller}} \Rightarrow \dots$

$\# = S$

$A \to abcBG'Dabc$

Regel

$\Rightarrow \delta(z_0, \varepsilon, A)$ enthält

$(z_0, \underbrace{abcBG'Dabc}_{\text{rechte Regelseite}})$

$\delta(z_0, a, a) = (z_0, \varepsilon)$

$a \in \Sigma$

⇒ Beachte: nur z_0 als Zustand!

Bsp: $S \to aSa \mid bSb \mid cSc \mid \varepsilon$

$$\delta(z_0, \varepsilon, S) = \{(z_0, \varepsilon), (z_0, aSa), (z_0, bSb), (z_0, cSc)\}$$

$$\delta(z_0, a, a) = \{(z_0, \varepsilon)\}$$
$$\delta(z_0, b, b) = \{(z_0, \varepsilon)\}$$
$$\delta(z_0, c, c) = \{(z_0, \varepsilon)\}$$

Eingabe: abccba

$(z_0, abccba, S) \Rightarrow (z_0, abccba, aSa)$
$\Rightarrow (z_0, \#abccba, Sa)$
$\Rightarrow (z_0, bccba, bSba)$
$\Rightarrow (z_0, ccba, Sba)$
$\Rightarrow (z_0, ccba, cScba)$
$\Rightarrow (z_0, cba, Scba)$
$\Rightarrow (z_0, cba, \#cba)$
$\Rightarrow \dots$

08.05.
2018

Kellerautomat ⇝ Kontextfreie Grammatik

1. Nur 1 Zustand.

$(z_0, abcd, \#) \to \dots \to (z_0, cd, ABCDE)$

$\# \Rightarrow \dots \Rightarrow abABCDE$

↑ Startsymbol

$\delta(z_0, a, A)$ enthält $(z_0, BCDE)$

dann Regel $A \to aBCDE$

$(a \in \Sigma \cup \{\varepsilon\}$

- Was ist bei mehr Zuständen? Bekommt man den mehr? → Nein!

Problem: Automat mit mehreren Zuständen in einen Zustand vereinfachen!

(z, w, X) Idee: Zustand im Kellerkopf mitspeichern (Alphabetvergrößerung)

$(z, X) \in \Gamma'$ z.B.

$\delta_A(z, a, A) \ni (z', BD)$

$\delta_N(z_0, a, [z, A]) \ni (z_0^*, [z', B]D)$

aber: Beim Löschen vom Keller geht der Zustand verloren!

$\delta_A(z', \varepsilon, B) \ni (z'', \varepsilon)$

$\delta_N(z_0, \varepsilon, [z', B]) \ni (z_0, \varepsilon)$

~

Alt: $(z_m, a\ldots, A\ldots) \to (z', \ldots, BD) \to (z'', \ldots, D)$

Neu: $(z_0, a\ldots, [z, A]\ldots) \to (z_0, \ldots, [z', B]D) \to (z_0, \ldots, D)$

Wo ist z''?

Idee: Merken des Zustands nach dem Löschen auch im Symbol des Kellers.

Neues Kelleralphabet.

$\delta_A(z, a, A) \ni (z', BC)$

$\delta_N(z_0, a, [z, A, z_1^*]) \ni (z_0, [z', B, z_2][z_2, C, z_1])$

(beliebig)

für alle $z_1, z_2 \in \Gamma_{alt}$

$\delta_A(z', \varepsilon, B) \ni (z'', \varepsilon)$

$\delta_N(z_0, [z', B, z'']) \ni (z_0, \varepsilon)$

14.05.
2018

Für das Kellerstartsymbol

$\#_A$ altes Kellerstartsym., z_0 Startzust.

$\#_N$ neues Kellerstartsymbol

$$\delta_N(z, \#_N, \varepsilon) = \{ z, (z_0, \#_A, z') \mid z' \in Z \}$$

Abschlußeigenschaften

Nicht abgeschlossen unter Schnitt und Komplement.

$\underbrace{a^n b^n c^n}_{\text{nicht kontextfrei!}} = \underbrace{a^n b^n c^k}_{\text{kontextfrei}} \cap \underbrace{a^k b^n c^n}_{\text{kontextfrei}}$ für k beliebig

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}} \quad \text{(nach De Morgan!)}$$

mit $L_1 = a^n b^n c^k$

$L_2 = a^k b^n c^n$

ist $L_1 \cap L_2 = a^n b^n c^n$

Welches der Komplemente ist nicht kontextfrei?

- **Komplement von $a^n b^n c^k$ ist kontextfrei**
  (für $n, k \geq 1$)

Wie sehen die Worte aus?

ε, $a^n b^n$ (da $k \geq 1$), $a^n b^m c^l$ $n \neq m$, l bel.
$a^m b^n c^l$ $n \neq m$, l beliebig
(a, b, c richtige Reihenfolge) c^k, $k \geq 1$ (da $n \geq 1$)

(Reihenfolge „falsch"):
... cb ... enthalten
... ca ...
... ba ...
... ac ...

Grammatik dazu:
T erzeugt $\{a,b,c\}^*$

$T \to \varepsilon \mid a \mid b \mid c \mid Ta \mid Tb \mid Tc$

$S \to TcbT \mid TcaT \mid TbaT \mid TacT$
$S \to R \mid U \mid G$

$R \to \varepsilon \mid aRb$
$G \to \varepsilon \mid Gc$
$U \to AaRG$
$A \to \varepsilon \mid aA$

} $a^m b^n c^l$ $m \neq n$

$m \leq n$ analog

14.05.
2018

$$a^n b^n c^n = a^n b^n c^k \cap a^k b^n c^n$$

$$= Kpl\left(Kpl\left(a^n b^n c^k \cap a^k b^n c^n \right)\right)$$

$$= Kpl.\Big(\underbrace{Kpl\,(a^n b^n c^k)}_{\text{Kontext-frei}} \cup \underbrace{Kpl\,(a^k b^n c^n)}_{\text{Kontextfrei}} \Big)$$

Kontextfrei

nicht. Kontextfrei !

$$Komplement\,(a^n b^n c^n) = Kpl(a^n b^n c^k) \cup Kpl(a^k b^n c^n)$$

ist Kontextfrei.

Deterministisch Kontextfreie Sprachen, Deterministischer Kellerautomat

$$|\delta(z, \varepsilon, A)| + |\delta(z, a, A)| \leq 1$$

$E \subseteq Z$ Endzustände

Wort wird erkannt $\Leftrightarrow$ Wort ist ganz gelesen, Automat im Endzustand, kann nicht mehr weitermachen.

Nicht det. Kontextfrei: $L = \{ w \overleftarrow{w} \mid w \in \Sigma^* \}$

det. Kontextfrei: $a^n b^n$

Abschluß: Komplement

- $a^n b^n c^k$ ist det. kontextfrei
- Komplement ist det. kontextfrei

$$a^n b^n c^n = Kpl\big(Kpl(a^n b^n c^k) \cup Kpl(a^k b^n c^n)\big)$$

$Kpl(a^n b^n c^k)$
$Kpl.(a^k b^n c^n)$ sind det. kontextfrei

Vereinigung ist kontextfrei aber nicht deterministisch kontextfrei

- Schnitt von 2 det. KF i.A. nicht kontextfrei

⇒ nicht unter ~~KO~~ Vereinigung, nicht unter Schnitt abgeschlossen.

Wortproblem in Linearzeit. (Wie?)

14.05.
2018

L (det.) Kontextfrei, R regulär

$L \cap R$ ist (det.) Kontextfrei

↳ DEA für R als zweite Komponente im Zustand mitführen

Leerheitsproblem

- Nichtterminale markieren, aus denen sich Terminal(wörter) ableiten lassen

Iterieren

Endlichkeitsproblem

N Pumping-Lemma-Zahl

Beobachtung: Endlich $\Leftrightarrow$ alle Worte $< N$

Unendlich $\Leftrightarrow$ Es gibt Wort $\geq N$

$\Leftrightarrow$ Es gibt ein Wort w

$N \leq |w| \leq 2N$

(mit Pumping Lemma)

Äquivalenzproblem

- Typ 3 Algorithmus: $L_1 = L_2$

$\Leftrightarrow L_1 \subseteq L_2$ und $L_2 \subseteq L_1$

$\Leftrightarrow (\overline{L_2} \cap L_1) = \phi$ und

$(\overline{L_1} \cap L_2) = \phi$

$A \Rightarrow B$
$\Leftrightarrow \neg A \wedge B$ ist immer falsch

- Det. Kontextfrei: haben Algorithmus (kompliziert)!

- allg. Kontextfrei: Kein Algorithmus! (nicht entscheidbar)

- L_1 det. Kontextfrei, L_2 regulär

geht, da $L_1 \cap L_2$ wie oben gezeigt wieder det. Kontextfrei

Abschluß unter Komplement,
Entscheidbarkeit d. Leerheitsproblems f.
det. Kontextfreie Sprachen.

15.05.
2018

4. Berechenbarkeit

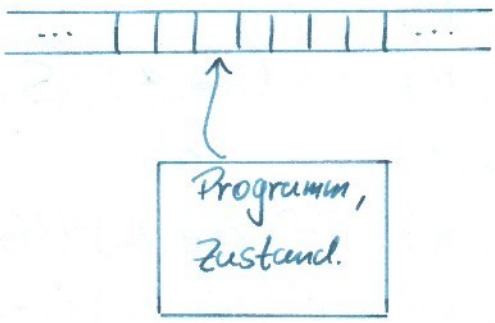
Eingabe: G_1, G_2 kontextfreie Grammatiken

Frage: Erzeugen G_1 und G_2 die gleiche Sprache?
(Ja oder Nein)

Es gilt: Für das Äquivalenzproblem gibt es
Keinen Algorithmus! (Beweis folgt)

Aber: Halb-lösbar für den Nein-Fall das Wort
suchen.
Falls Nein, findet es. Sonst Endlosschleife!

Turing Maschine (einfachster Automat, auf dem
jeder Algorithmus programmierbar ist)



Band zum Speichern

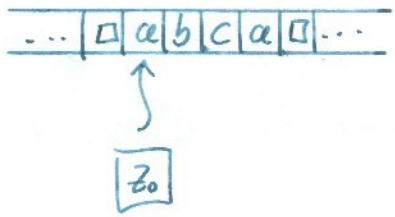
- Z Zustände
- Σ Eingabealphabet
- $\Gamma \supsetneq \Sigma$ Bandalphabet
- $z_0 \in Z$ Startzustand
- $\square \in \Gamma \setminus \Sigma$ Leerzeichen (Blank)
- $E \subseteq Z$ Endzustände
- δ Programm.

$$\delta : (Z \times \Gamma) \longrightarrow (Z \times \Gamma \times \{L, N, R\})$$

Kopfbewegung (links, nichts, rechts)

Bsp.: Turingmaschine, die $a^n b^n c^n$ $n \geq 1$ erkennt.

Am Anfang: Band (Kopf auf Anfang d. Eingabe, sonst nur Blanks auf Band)



$\delta(z_0, \square) = (S, \square, N)$ Zust. $S \rightarrow$ Senke

$\delta(z_0, b) =$ — " —

$\delta(z_0, c) =$ — " —

alle richtigen Zeichen „löschen"

$\delta(z_0, a) = (z_a, L_a, R)$

$\delta(z_a, a) = (z_a, a, R)$

$\delta(z_a, c) = (S, c, N)$

$\delta(z_a, b) = (z_b, L_b, R)$

$\delta(z_b, b) = (z_b, b, R)$

$\delta(z_b, a) = (S, a, N)$

$\delta(z_b, c) = (z_R, L_c, N)$

$\delta(z_R, x) = (z_R, x, L)$

$x \in \{L_c, L_b, a, b\}$

$\delta(z_R, L_a) = (z_0, L_a, R)$

$\delta(z_0, L_b) = (z_{Test}, L_b, R)$

$\delta(z_{Test}, x) = (S, x, N)$

a, b, c

$\delta(z_{Test}, L_b) = (z_{Test}, L_b, R)$

$\delta(z_{Test}, L_c) = (z_{Test}, L_c, R)$

$\delta(z_{Test}, \square) = (z_E, \square, N)$

(z_E: akzeptierender Endzustand.)

15.05.
2018

Konfiguration:

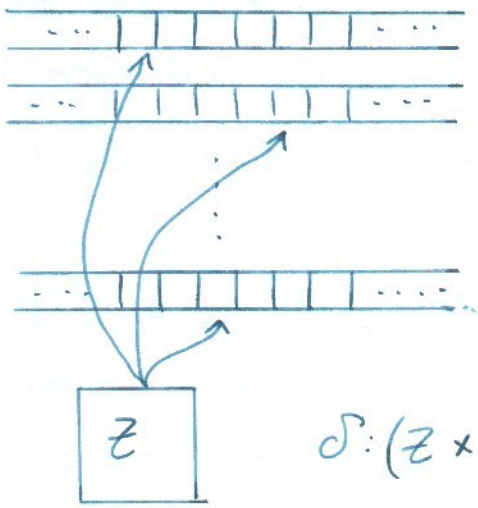
$\Gamma^* Z \Gamma^*$

- Erkennen mit Endzustand und Halten!

Zeit: # Rechenschritte

(f. Bsp.: $a^n b^n c^n$ Mind. n^2 $O(n^2)$)

- Erweiterungen:
  - Mehrband TM, mit $K \geq 1$ Bändern, K Konst.



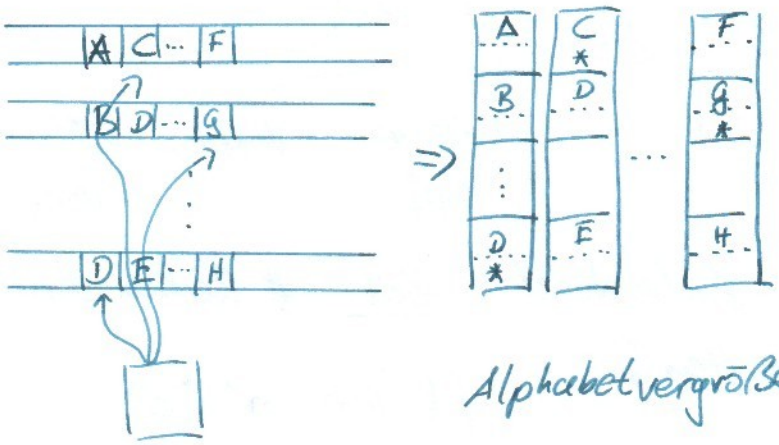
$\delta: (Z \times \Gamma^K) \to (Z \times \Gamma^K \times \{L, N, R\}^K)$

$a^n b^n c^n$ in $O(n)$ Anfang: Eingabe auf Band 1

1) von Links nach rechts b auf Band 2
2) —— " —— c auf Band 3
3) zurück auf Anfang der a, b, c
4) Abgleich auf den Bändern

Mehrband und Einband-TM sind äquivalent.
(bezogen auf Berechenbarkeit)

⇒ Simulation von K-Band TM auf Einband TM



Alphabetvergrößerung!

Simulation:
1) Gehe ganz nach links
2) Suche die *'e auf den Stücken, Merke Zeichen unter Stern
3) Simuliere Rechnung der K-Band-Maschine
   (neue Zeichen an richtige Stelle schreiben, Köpfe bewegen)
4) neue Simulation (gehe zu 1))

Bsp.: Endlosschleife mit TM

$\delta(z_0, a) = (z_0, a, R)$

$\delta(z_0, \square) = (z_0, \square, R)$

15.05.
2018

Funktionen auf $\mathbb{N}$. Einfache Programmiersprachen

LOOP: $x, y, \ldots$ Variablen

$0, 1, 2, 3$ Konstanten

LOOP x do
    Programm
end

} Schleife

$+, -$ Operationen ($-$ wird nicht negativ)

$y :=$ Ausdruck auf $+, -$ Zuweisung

Programm für Addition: $(f(x_1, x_2) := x_1 + x_2)$

x_0 Ausgabe
x_1, x_2 Eingabe

} Konvention

$x_0 := x_1 + x_2$

Multiplikation:

$x_0 := 0$
Loop x_2 do
    $x_0 := x_0 + x_1$
end

Schleifendurchläufe liegen von Betreten der Schleife fest ⇒ Keine! Endlosschleifen

Simulation von: if x=0 then A end

```
y=1
Loop x do y=0 end
Loop y do Übersetzung von A end
```

Subtraktion einer Konst. nur mit Add.

```
z=0
y=0
Loop x do
    z=z+y
    y=1
end
```

(z := x-1)

25.05.
2018

Simulation: if x=0 then A else B end

```
y=1
z=1
Loop x do
|   y=0
|   Loop z do
|   | B
|   end
|   z=0
end
Loop y do
| A
end
```

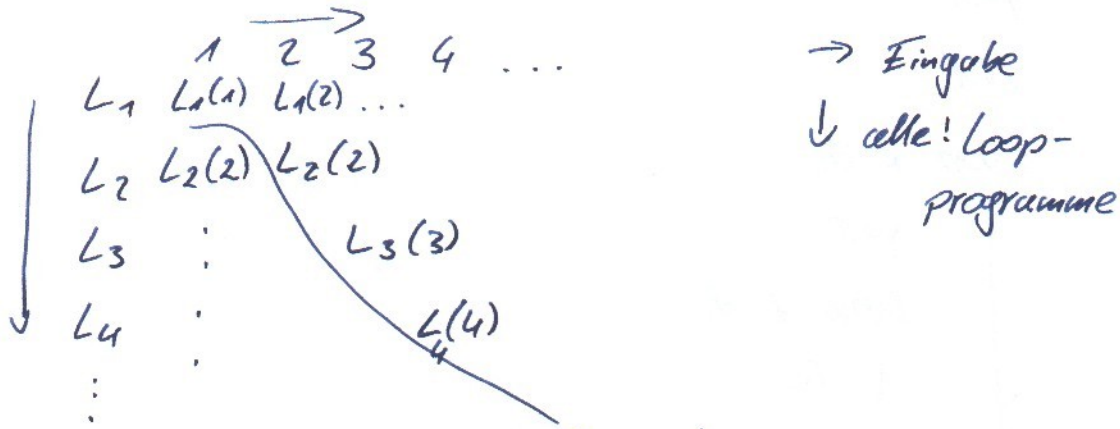
Eins abziehen:

```
y=0
z=0
Loop x do
|   Loop y do
|   |   z=z+1
|   end
|   y=1
end
```

$z \Rightarrow x-1$

Loop terminiert immer,

Funktion, die nicht Loop-Berechenbar ist?!



~~Folgender Algorithmus~~

Gibt es Algorithmus mit

Eingabe: $k \geq 1$

Ausgabe: $L_k(k)$?

Ja!

Hält immer!

Gibt es auch Loop-Programm dafür?

Das gibt es nicht!

Beweis: Angenommen, es gibt dafür ein Loop-Programm.

Dann ist dieses Programm Links irgendwo hingeschrieben. Nehmen wir an, es ist ~~L_k~~ L_ℓ

~~Dann ist $L_k(k)$ ein Wert.~~

Also gilt: $L_\ell(k) = L_k(k)$

28.05.
2018

Betrachte folgendes Programm:

Eingabe: $k \geq 1$
Ausgabe: $L_k(k)+1$

hat Algorithmus.

Gibt es dafür ein Loop-Programm?

Falls ja: Sei dies L_l.

Was ist $L_l(l)$?

$(L(l) \neq L(l)+1)$!

das müßte $L_l(l)+1$ sein!

(Widerspruch!)

WHILE-Programme

(ist wie LOOP, zusätzlich)

```
while x > 0 do
|   P
end
```

(x wird üblicherweise in P modifiziert)

```
Loop x do
|  ...
end
```

=>

```
y = x      (neue Var.)
while y > 0 do
|   ...
|   y = y - 1
end
```

Satz: While berechenbar ⇒ Turing-berechenbar.

Beweis: K-Band Turing-Maschine

→ pro Variable ein Band,
eventuell noch Hilfsbänder

→ binäre Simulation der Rechenschritte. ∎

GOTO-Programme

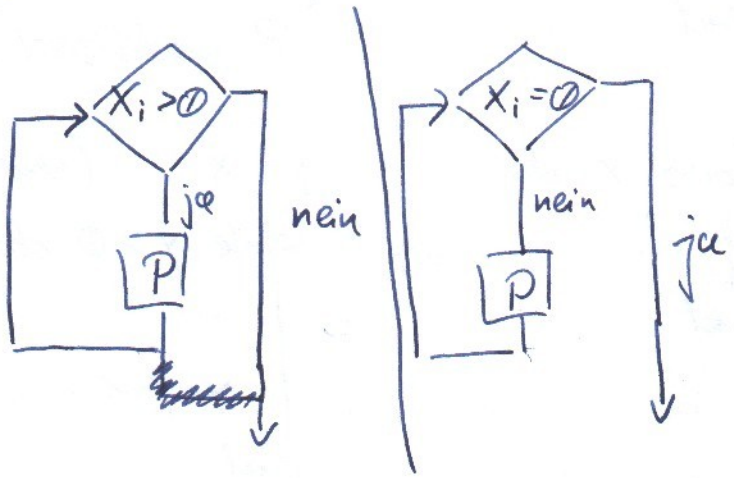
M_1: A_1
M_2: A_2
⋮

M_i, Marke
A_i, Anweisung:

- $x_i = x_i \pm C$
- goto M_i
- if $x_i = 0$ then goto M_j
- halt

Satz: WHILE-Berechenbar ⇒ GOTO-Berechenbar.

Beweis: Simulation der While-Schleife



28.05.
2018

Satz: GOTO-Berechenbar ⇒ WHILE-Berechenbar

Beweis:

M_0: halt
M_1: A_1
M_2: A_2
⋮
M_k: A_k

⇝

```
pc = 1
while pc > do
    if pc=1 then A1' end
    if pc=2 then A2' end
      ⋮
    if pc=k then Ak' end
end
```

A_i ist $x_i = x_i + C$

⇓

A_i' ist $x_i = x_i + C$
$pc = pc + 1$

usw.

∎

Folgerung: Eine While-Schleife reicht!
Sonst nur noch Loop-Schleifen gebraucht.

Satz: TM ~~berechenbar~~ berechenbar ⇒ Goto-Berechenbar.

Beweis: Γ Arbeitsalphabet der TM

Interpretieren Wort $(a_1 \dots a_n) \in \Gamma^*$ als

$$b = |\Gamma| + 1$$

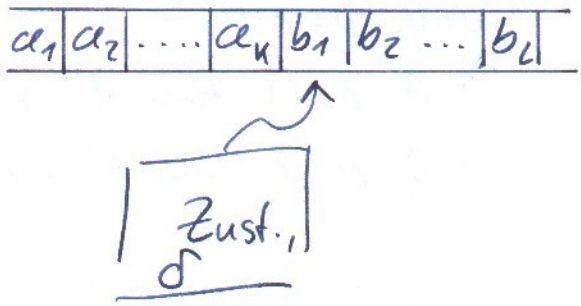
$$a_1 b^{n-1} + a_2 b^{n-2} + \dots + a_n b^0$$

$$\Gamma = \{c_1, \dots, c_{|\Gamma|}\} \rightsquigarrow c_1 = 1 \quad c_2 = 2, \dots, c_{|\Gamma|} = |\Gamma|$$

(@ nicht getroffen)

⇒ Jedes Wort aus T^* entspricht genau einer Zahl. (Noch $\emptyset = \varepsilon$.) ~~oder~~

Zustand der Turing-Maschine:



⇓

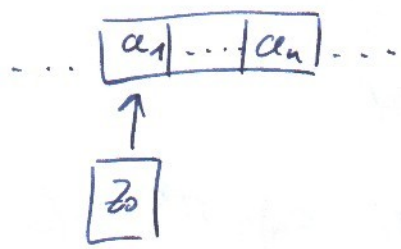
3 Zahlen:

$x = (a_1 \dots a_k)_b \in \mathbb{N}$ (Links von Kopf)

$y = (b_l \dots b_1)_b \in \mathbb{N}$ (Kopfpos. und rechts davon)

$z = l$ ↑ Zustand

Eingabe der TM:



GOTO-Programm

- hat in x_1 den Wert stehen $(a_1 - a_n)_b \in \mathbb{N}$
- Setzen am Anfang:

29.05.
2018

Nochmal neu!

Satz: TM $\Rightarrow$ GOTO

Beweis: • Γ Arbeitsalphabet $\quad \overset{\in\Gamma}{a_1} \dots a_n \in \Gamma^* \setminus \{\varepsilon\}$

• Zuordnung $a \in \Gamma \mapsto$ Zahl zwischen 1 und $|\Gamma|$

$$(a_1 \dots a_n)_{\underbrace{|\Gamma|+1}_{=b}} = a_1 b^{n-1} + \dots + a_{n-1} b^1 + a_n b^0$$

$\in \{1, \dots, |\Gamma|\}$

• **TM berechnet:**

$$a \in \Sigma^* \mapsto TM(a) \in \Gamma^* \quad \text{(evtl. Endlosschleife)}$$

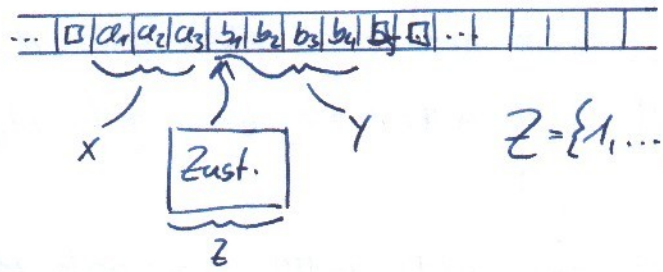
• **Goto-Programm berechnet:**

$$(a)_{|\Gamma|+1} \mapsto (TM(a))_{|\Gamma|+1}$$

TM wird durch Goto-Programm Schritt für Schritt simuliert.

$b = |\Gamma| + 1$

Zwischenzustand der TM:

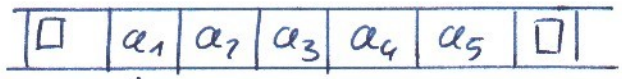


$Z = \{1, \dots, n\}$

$X = (a_1 a_2 a_3)_b \qquad Y = (b_5 b_4 b_3 b_2 b_1)_b$

$z = K$

Am Anfang:

TM: 

Goto:
$u = (a_1 a_2 a_3 a_4 a_5)_b$
(
Eingabe Goto-Programm.

1) Initialisierung von Goto:

(wollen haben: $y = (a_5 a_4 a_3 a_2 a_1)_b$)

$y = 0$

A: if $u = 0$ then goto SIM

$v =$ ~~$u \bmod b$~~ u MOD b

$u = u$ DIV b

$y = y \cdot b + v$

goto A

SIM: ... (kommt gleich)

$z = 0$ (Startzust.)

$x = (\square)_b$ (Blank links vom Kopf)

2) Simulation:

SIM: $a = y$ MOD b

if $z = 0$ AND $a = 1$ then goto M_{01}

if $z = 0$ AND $a = 2$ then goto M_{02}

⋮

if $z = n$ AND $a = |\Gamma|$ then goto $M_{n,|\Gamma|}$

29.05.
2018

$\delta(z, a) = (z', c, L)$

z: z, a: Zeichen zu j, z': l, c: j', L

$\Downarrow$

$M_{k,j}$: $z = l$

(für nicht definierte $M_{k,j}$ $\Rightarrow$ halten)

$y = y \text{ DIV } b$

$y = y \cdot b$

$y = y + j'$

$y = y \cdot b + (x \text{ MOD } b)$

$x = x \text{ DIV } b$

goto SIM

rest analog, noch Ausgabekonvention herstellen.

∎

Churchsche These: Jeder Algorithmus ist durch eine Turing Maschine zu machen!

(Auch: Simulation der RAM durch TM.)

Unentscheidbarkeit, Halteproblem

Σ Alphabet, Sprich $A \subseteq \Sigma^*$ ist entscheidbar

$\Leftrightarrow$ Es gibt einen Algorithmus für das Problem:

Eingabe: $w \in \Sigma^*$

Ausgabe: Ja, falls $w \in A$,
Nein sonst.

$\Leftrightarrow$ $\chi(w) = \begin{cases} 1 & w \in A \\ 0 & \text{sonst} \end{cases}$

Gibt es Sprachen, die nicht entscheidbar sind?

Spezielles Halteproblem:

Alle Turing-Maschinen, durch Programm beschrieben

$TM \mapsto w \in \{0,1\}^*$ zu $w \in \{0,1\}^*$ haben wir TM_w

(falls w kein Programm, dann irgendeine feste TM dafür nehmen)

$K \subseteq \{0,1\}^*$

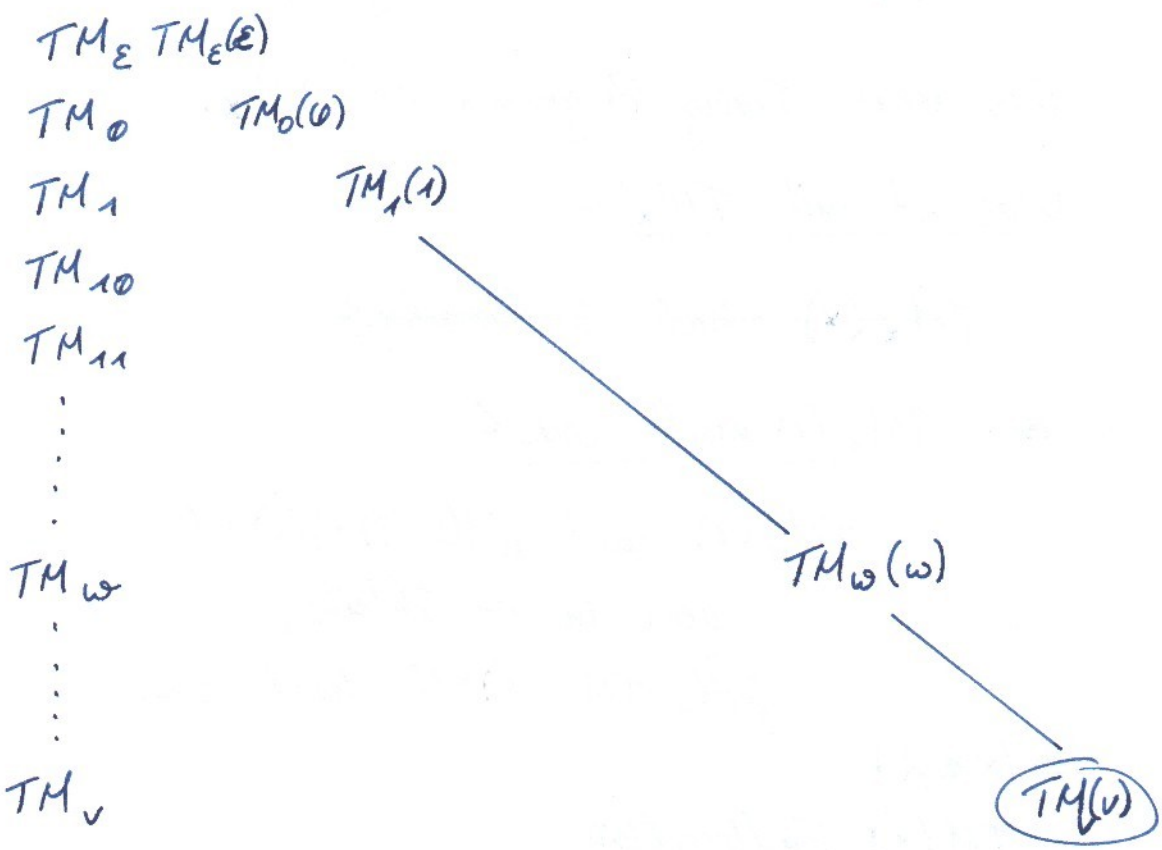
$K = \{ w \in \{0,1\}^* \mid TM_w \text{ mit Eingabe } w \text{ hält} \}$

29.05.
2018

Satz: K ist nicht entscheidbar! ▽

Beweis:

ε 0 ~~01~~ 10 11 ... ω ... v



- Diagonale maßgeblich für K, alle w wo auf der Diagonale ein Wert steht (also hält) sind in K.

- Angenommen haben Algorithmus für K, Entscheidung von K.

  Nach Church'scher These haben wir dann ~~#~~ TM_v für die Entscheidung von K. ($TM_v(v) = 1$, kein Widerspruch)

Ändere TM_v ab:

Falls TM_v eine 1 ausgibt, gehe in Schleife.

Falls TM_v eine 0 ausgibt, dann halten.

Das neue Turing-Programm ist TM_x.

Was ist mit TM_x?

~~$TM_x(x)$ hat Endlosschleife~~

$\Leftrightarrow$ $TM_x(x)$ macht was?

$TM_v(x)$ und falls $TM_v(x)=1$
gehe in ∞-Schleife,
falls $TM_v(x)=0$ halte an.

$\Leftrightarrow (x \notin K)$

$TM_x(x)$ Endlosschleife

$\Leftrightarrow TM_v(x)$ gibt 1 aus

$\Leftrightarrow TM_x(x)$ hält

↯

$\Leftrightarrow (x \in K)$

∎

04.06.2018

Weitere unentscheidbare Probleme

allgemeines Halteproblem

$H = \{ w \# v \mid TM_w \text{ mit Eingabe } v \text{ hält} \}$

Reduktion: $K \leq H$

$w \mapsto w \# w$

Algorithmus für H gibt direkt Algorithmus für K. Widerspruch!

Halteproblem auf leerem Band:

$H_\emptyset = \{ w \mid TM_w \text{ mit Eingabe } \varepsilon \text{ hält} \}$

Reduktion: $H \leq H_\emptyset$

$w \# v \mapsto w'$ so dass

$TM_{w'}(\varepsilon)$ hält

$\Leftrightarrow$

$TM_w(v)$ hält.

~~Aus $w \# v$ muss w' gebaut werden, so dass~~

$TM\ w'$ so:
1) Schreibe Wort v aufs Band. gehe in die Startsituation von TM_w
2) Lasse ~~v~~ laufen. $TM_w(v)$

K, H, H_0 sind semi-entscheidbar. Das heißt es gibt einen Algorithmus mit folgendem Verhalten:

Ausgabe 1 $\Leftrightarrow$ $w \in K$

Nicht halten $\Leftrightarrow$ $w \notin K$

Allgemeines Korrektheitsproblem:

Etwa bei Eingabe von 5 soll 5! herauskommen.

G = Menge aller Programme mit $5 \mapsto 5!$

oder z.B. $n \mapsto n^2$

G' = Menge aller Programme, die $n \mapsto n^2$ berechnen.

Suche Programm aus G'.

04.06.
2018

Satz (Rice):

Sei R = Menge aller Funktionen, die durch TM berechnet werden können.

Sei $S \subseteq R$ dann gilt: $S \neq \emptyset$, $S \neq R$

$C(S)$ = Menge aller Programme, deren berechnete Funktion in S liegt.

Dann gilt: $C(S)$ ist nicht entscheidbar!

$$C(S) = \{ w \mid TM\ w \text{ gehört zu } S \}$$

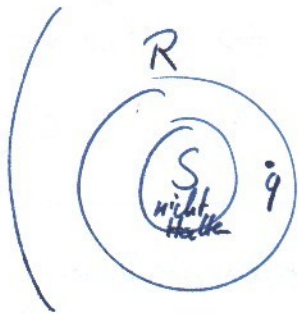
Beweis: ~~Sei~~

Es ist $w \mapsto$ Nicht halten. eine Funktion in R.

~~Ann Annahme~~

Annahme: Diese Funktion gehöre zu S.
Sei q eine Funktion, die nicht zu S gehört.

Sei Q die TM zu q.



Reduktion: $H_0 \leq C(S)$

$w \mapsto w'$ mit

$$w' \in C(S) \Leftrightarrow w \in H_0$$

Nehme w und baue neue TM w' so:

Bei Eingabe w'' macht TM w' folgendes.

1. führe TM w auf ε aus
2. kommt 1. zum Schluß (d.h. hält), dann Programm Q ausführen mit Eingabe w''.

Jetzt gilt:

$w \in H_0$

$\Leftrightarrow$ $TM_w(\varepsilon)$ hält

$\Leftrightarrow$

$TM_{w'}(w'') = Q(w'')$ für alle Eingaben w''.

$\Leftrightarrow$ $w' \notin C(S)$

und: $w \notin H_0$

$\Leftrightarrow$ $TM_w(\varepsilon)$ hält nicht

$\Leftrightarrow$ $TM_{w'}(w'')$ hält nicht

$\Leftrightarrow$ $w' \in C(S)$ ∎

Äquivalenzproblem: $\ddot{A} = \{ w\#v \mid TM_w$ und TM_v berechnen dieselbe Funktion$\}$

Mit Rice: $S = \{ v \mid TM_v$ berechnet feste Funktion, die zu TM gehört$\}$

05.06.2018

Postsches Korrespondenzproblem

Eingabe: $K = ((x_1, y_1), (x_2, y_2), \dots, (x_k, y_k))$

$x_i, y_i \in \Sigma^*$

Frage: Gibt es eine Folge $i_1, i_2, \dots, i_l$ aus $\{1, \dots, k\}$, so dass

$$x_{i_1} x_{i_2} \dots x_{i_l} = y_{i_1} y_{i_2} \dots y_{i_l} \quad ? \quad (l \text{ beliebig})$$

Einfaches Beispiel:

$$((11, 1), (1, 11*), (**, *))$$

Lösung:

	1 1	1	* *	x
	1	1 1 *	*	y
	1	2	3	
	$\|$	$\|$	$\|$	
	i_1	i_2	i_3	

- 1 Lösung $\Rightarrow \infty$ viele Lösungen (bestehende Lösung einfach wiederholen!)

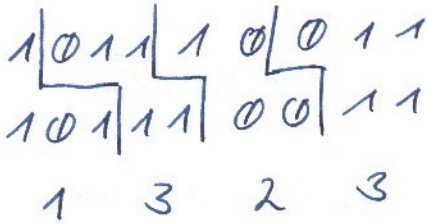
ohne Lösung: z.B. $((11, 1))$

Falls eine Lösung existiert, dann lässt sie sich mit Algorithmus (Suchprozess, Backtracking) finden.

$\Rightarrow$ semientscheidbar!

noch ein Bsp. (etwas Komplizierter):

$((1, 101), (10, 00), (011, 11))$



Lösung

Satz: Halteproblem # ist ~~reduzierbar auf ein PKP~~

* folgendermaßen auf das PKP reduzierbar:

Prog. ↗ x w ↙ Eingabe

$\text{Prog.}\ x\#w \mapsto ((x_1, y_1), \dots (x_k, y_k))$

wobei gilt $w\#v \in \# \Leftrightarrow$ PKP hat Lösung der Form $i_n = 1, \dots$.

(mod. PKP, Lösung muß mit 1. Paar beginnen!)

Idee: Rechnung/Programm der TM in $\mathcal{P}$(M)PKP übersetzen.

(Worte stehen für Konfiguration)

05.06.
2018

Beweis: Haben TM mit Γ, Z.
Alphabet für PKP ist $\Gamma \cup Z \cup \{\#\}$.

$x_1 = \#$ „Füllen voll" $\#$

$y_1 = \# z_0 w \#$ „Rechenschritt simulieren" $\#$

jetzt noch folgende Wortpaare:

- ~~for~~ ist $\delta(z,a) = (z', c, N)$, dann Wortpaar

$$x_i = za \quad y_i = z'c \equiv (za, z'c)$$

- Kopierregeln: (a,a) für alle $a \in \Gamma \cup \{\#\}$
- $\delta(z,a) = (z', c, R) \Rightarrow (za, cz')$ und $(za\#, cz' \square \#)$
- $\delta(z,a) = (z'c, L) \Rightarrow (bza, z'bc)$ für alle $b \in \Gamma$ $(\#za, \#z' \square c)$
- falls $\delta(z, \square) = (z', c, N) \Rightarrow (z\#, z'c\#)$ zusätzlich

Kann der rechte Rand sein!

$\delta(z, \square) = (z', c, L) \Rightarrow (bz\#, b z' z' bc\#)$ für $b \in \Gamma$ und $(\#z\#, \#z' \square c\#)$

$\delta(z, \square) = (z', c, R) \Rightarrow (z\#, cz'\#)$

Bei nicht halten ⇒ keine Lösung!

Bei Halten auch nicht! (Mit den Paaren bis hierher.)

Noch extra Paare zum Halten!

Haben einen definierten Endzustand, z_e wo immer schluß ist. Sonst nie!

(Jede TM kann so umgebaut werden!)

$(z_e a, z_e)$ und $(a z_e, z_e)$ für alle $a \in \Gamma$.

Am Schluß folgende Situation:

x: #

y: #z_e#

⇒ noch $(z_e \#\#, \#)$

∎

(Lösung ⇒ Halten

Halten ⇒ Rechnung, die hält ⇒ Lösung.)

05.06.2018

Eine weitere Reduktion zum allgemeinen PKP

$$K = \left((x_1, y_1), \ldots, (x_k, y_k)\right) \mapsto K' = \left((x'_1, y'_1), \ldots, (x'_{k'}, y'_{k'})\right)$$

K hat Lösung $\Leftrightarrow$ K' hat irgendeine Lösung mit $i_1 = 1$.

$x_1 \mapsto \#x_1\#$ $\quad x_2 \mapsto x_2\#$ $\quad \ldots$ $\quad x_k \mapsto x_k\#$

$y_1 \mapsto \#y_1$ $\quad y_2 \mapsto \#y_2$ $\quad \ldots$ $\quad y_k \mapsto \#y_2$

$x_1 = 101 \mapsto \#1\#0\#1\#$

$x_1 = 10 \mapsto \#1\#0$

↗ Analog, auch jeden Zeichen #

- auch noch für zwischendrin

$x_1 \to x_1\#$

$y_1 \to \#y_1$

- und fürs Ende

$(\#, \#\#)$

∎

11.06.
2018

Wiederholung: PKP unentscheidbar

TM w, Eingabe x, genau ein Endzustand z_e,

$\delta(z_e, a)$ immer undef.

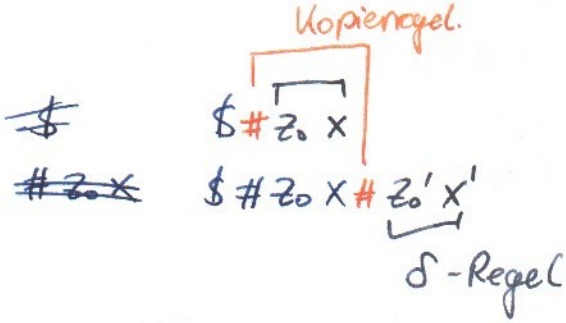
sonst immer def.

Reduktion auf PKP, das immer mit Wortpaar 1 anfängt.

Wortpaar 1: $\$\#$ / $\$\# z_0 x$ $\qquad (x_1, y_1)$ — $\$$ / $\$\# z_0 x$

Kopierregel: $\frac{a}{a}$ $\quad \forall a \in \Gamma \cup \{\#\}$

$\delta(z,a) = (z', c, N) \approx \frac{za}{z'c} \equiv (zc, z'c)$



Am Ende:

$\# abc z_e def \#$

$\# abc z_e def \# ab z_e def \#$ ... $\# z_e \# \#$ / $\# z_e \# \#$

$(az_e, z_c) \& (z_e a, z_e)$

für alle $a \in \Gamma \cup \{\#\}$

11.06.
2018

Folgerung: PKP über $\Sigma = \{0,1\}$ ist auch unentscheidbar.

Beweis: Haben PKP über $\Sigma = \{a_1, a_2, \dots, a_m\}$.

Schreibe a_i als $0\underbrace{1\dots1}_{i \text{ Einsen}} = 01^i$

0 Trennzeichen.

Im Gegensatz: (Übung)

$|\Sigma| = 1$, dann entscheidbar!

Äquivalenz von Typ-3 Grammatiken.

$$L_1 \subseteq L_2 \Leftrightarrow (\Sigma \setminus L_2) \cap L_1 = \emptyset$$

$$L_1 = L_2 \Leftrightarrow (\Sigma \setminus L_2) \cap L_1 = \emptyset$$

und $(\Sigma \setminus L_1) \cap L_2 = \emptyset$

bei Typ 2 geht das so nicht!

Da nicht unter Komplement abgeschlossen!

$L = \{a^n b^n c^n \mid n \geq 1\}$ nicht Typ-2. $\Sigma = \{a, b, c\}$

Aber: $\Sigma \setminus L$ ist Typ-2.

(Kontextfrei nicht unter Schnitt abgeschlossen:

$$a^n b^m c^m \cap a^m b^m c^n = a^n b^n c^n$$)

Satz: Unentscheidbar ist:

Eingabe: G_1, G_2 kontextfreie Grammatiken

Frage: Ist $L(G_1) \cap L(G_2) = \emptyset$?

(Komplement des Problems ist semi-entscheidbar.)

Beweis: Reduktion vom PKP.

$$K = ((x_1, y_1), (x_2, y_2), \dots, (x_k, y_k)) \quad x_i, y_i \in \{0,1\}^* \setminus \{\varepsilon\}$$

Eine Grammatik: Das, was das PKP aus den x'en ~~und y'en~~ und y'en erzeugen kann. (unabhängig voneinander)

$$A \to a_1 A x_1 \mid a_2 A x_2 \mid \dots \mid a_k A x_k \mid$$
$$a_1 x_1 \mid a_2 x_2 \mid \dots \mid a_k x_k$$

erzeugt z. B.:

$$A \Rightarrow \dots \Rightarrow a_1 a_2 a_3 x_3 x_2 x_1$$

$$B \to \overleftarrow{y_1} B a_1 \mid \dots \mid \overleftarrow{y_k} B a_k \mid \overleftarrow{y_1} a_1 \mid \dots \mid \overleftarrow{y_k} a_k$$

$\overleftarrow{abc} = cba$

$$S \to A \$ B$$

$\Rightarrow$ Grammatik G_1

11.06.2018

Bsp: PKP $K = ((10,1),(\overset{10}{\cancel{010}},\overset{010}{\cancel{10}}))$

$\left(\begin{array}{c|c} 10 & 10 \\ 1 & 010 \end{array}\right)$ Lösung

Ableitung:

$S \Rightarrow A\$B \Rightarrow a_1 A 10 \$ B$

$\Rightarrow a_1 a_2 \overset{10}{\cancel{010}} 10 \$ B$

$\Rightarrow a_1 a_2 \overset{10}{\cancel{010}} 10 \$ 1 B a_1$

$\Rightarrow a_1 a_2 \overset{10}{\cancel{010}} 10 \$ \overset{1010}{\cancel{1010}} a_2 a_1$

$S \Rightarrow A\$B \Rightarrow a_2 A 10 \$ B$

$\Rightarrow a_2 a_1 10 10 \$ B$

$\Rightarrow a_2 a_1 1010 \$ 010 B a_2$

$\Rightarrow a_2 a_1 1010 \$ 010 1 a_1 a_2$

Lösung Lösung

Lösung vom PKP:

wort $ Spiegelwort

Jetzt: G_2 : Test auf Spiegelwort.

$$S \to a_1 S a_1 \mid \dots \mid a_k S a_k \mid T$$

$$T \to 0T0 \mid 1T1 \mid \$$$

PKP hat Lösung $\Leftrightarrow L(G_1) \cap L(G_2) \neq \emptyset$

Folgerung: • $L(G_1) \cap L(G_2)$ ist unendlich? Ist nicht entscheidbar.

Folgerung: • $L(G_1) \cap L(G_2)$ ist Kontextfrei. Ist nicht entscheidbar.

Beweis: • Hat das PKP eine ~~Lösung~~ Lösung, dann ist der Schnitt nicht kontextfrei.

Haben eine Lösung; wie sieht $L(G_1) \cap L(G_2)$ aus?

Typische Worte haben die Form:

Indexfolge der x_i's \$

$a_{10} \dots a_5 x_5 \dots x_{10} \$ \overleftarrow{y}_{10} \dots \overleftarrow{y}_5 a_5 \dots a_{10}$

$\Rightarrow$ Mit Pumping-Lemma nicht kontextfrei!

11.06.2018

Folgerung: $L(G_1) \subseteq L(G_2)$ ist nicht entscheidbar.
$L(G_1) = L(G_2)$ ist nicht entscheidbar.

Beweis: Die beiden Grammatiken zum PKP K ~~sind~~ oben sind deterministisch kontextfrei.

$$(a_5 \dots a_1 x_1 \dots x_5 \$ \overleftarrow{y}_{10} \dots \overleftarrow{y}_7 a_7 \dots a_{10})$$

Also sind die Komplemente auch (deterministisch) kontextfrei!

Zu ~~f~~ den Grammatiken G_1, G_2 vom PKP

$\rightsquigarrow$ baue det. Kellerautomat

$\rightsquigarrow$ baue det. Kellerautomat f. Komplemente

$\rightsquigarrow$ baue Grammatiken f. Komplemente.

Sei $L_1 = L(G_1)$ G_1, G_2 Grammatiken vom PKP

$L_2 = L(G_2)$

G_1', G_2' Grammatiken f. Komplemente.

PKP hat keine Lösung $\Leftrightarrow L(G_1) \cap L(G_2) = \emptyset$

$\Leftrightarrow L(G_1) \subseteq L(G_2')$

$\Leftrightarrow L(G_1) \cup L(G_2') = L(G_2')$

$\Leftrightarrow L(G_3) = L(G_2')$, G_3 ist Gr. f. $G_1 \cup G_2'$

■

Damit ist sowohl das Enthaltensein als auch das Äquivalenzproblem nicht entscheidbar.

12.06.2018 Reduktion von $L(G_1) \subseteq L(G_2)$ auf Äquivalenzproblem:

$G_1, G_2 \mapsto G_3, G_2$

mit $L(G_3) = L(G_1) \cup L(G_2)$

jetzt: $L(G_1) \subseteq L(G_2) \Leftrightarrow L(G_3) = L(G_2)$

Teilmenge nicht entscheidbar

$\Rightarrow$ Äquivalenz nicht entscheidbar.

---

analoge Folgerungen f. det. Kontextfrei.

außer Äquivalenz!

12.06.
2018

Folgerung: Für kontextfreie Grammatiken ist unentscheidbar:

- G Mehrdeutig ?
- $\overline{L(G)}$ kontextfrei ?
- $L(G)$ regulär ?
- $L(G)$ det. kontextfrei ?

Beweis: G_1, G_2 Grammatiken aus PKP-Beweis.

G_3 ist Vereinigungsgrammatik aus G_1, G_2

dann:

PKP Lösbar $\Leftrightarrow$ G_3 mehrdeutig

G_1, G_2 nicht mehrdeutig.

falls PKP Lösung hat, dann kann die Lsg. aus G_1 oder aus G_2 abgeleitet werden. ∎

- **Komplement Kontextfrei:**

$$L(G_1') = \overline{L(G_1)}\,; \quad L(G_2') = \overline{L(G_2)}$$

G_4 ist Vereinigungsgr. von G_1' und G_2'

$$L(G_4) = L(G_1') \cup L(G_2')$$

$$= \overline{L(G_1)} \cup \overline{L(G_2)}$$

$$\overline{L(G_4)} = \overline{\overline{L(G_1)} \cup \overline{L(G_2)}} = L(G_1) \cap L(G_2)$$

$$\overline{L(G_4)} = L(G_1) \cap L(G_2)$$

$= \phi$ falls Keine Lösung von PKP

$=$ ... nicht kontextfrei sonst. ∎

$L(G_4) = \Sigma^*$ $\Rightarrow$ Keine Lösung

12.06.
2018

Komplexitätstheorie

$O(n^k)$ k Konstant, Polynomialzeit

Exponentiell: $O(2^n)$, 2^{n^2}, $2^{\varepsilon n}$ $\varepsilon > 0$, $n!$

Ziel: $2^{\varepsilon n}$ nicht $O(n^k)$

$2^n \geq n+1$ durch Induktion ($n \geq 0$)

$(2^x \geq x, x \in \mathbb{R})$

$\Rightarrow 2^{\varepsilon \cdot n} \geq \varepsilon \cdot n$ (durch Subst.)

$\Leftrightarrow (2^{\varepsilon \cdot n})^{1/c} \geq (\varepsilon n)^{1/c}$

$\Rightarrow 2^n \geq \varepsilon^{1/c} \cdot n^{1/c}$

$2^x \not\geq x$

$\Rightarrow 2^{\varepsilon x} \geq x^{\varepsilon}$ $x = \frac{y}{\varepsilon}$

$\Rightarrow 2^y \geq$

$$2^n \geq n$$

$$\Rightarrow 2^{(k+1)n} \geq n^{k+1}$$

$$\Rightarrow 2^m \geq \left(\frac{1}{k+1}\right)^{k+1} \cdot m^{k+1}$$

$$= \underbrace{\left(\frac{1}{k+1}\right)^{k+1} \cdot m}_{\geq 1 \text{ für } m \text{ groß genug}} \cdot m^k \geq m^k$$

Nicht polynomialzeit Probleme:

- TSP $n^2 \cdot 2^n$ mit dyn. Programmierung
- 3-färbbarkeit etwa $3^n \cdot p(n)$ ↳ Polynom

12.06.
2018

Erfüllbarkeitsproblem d. Aussagenlogik
(SAT-Problem)

$(x_1 \vee x_2 \vee \neg x_4 \vee x_5) \wedge \ldots$

(

0, 1

wahr/falsch

Entscheidbar in 2^n,

n Variablen

NP (nichtdeterministisch polynomial)

enthält Probleme der folgenden Form

$L \in NP \Leftrightarrow$ • Für $w \in L$ gibt es ein Wort v polynomial lang, so dass es einen poly. Algorithmus gibt, der auf Eingabe (w, v) ja ausgibt

• Für $w \in L$ gibt es kein Wort, so dass dieser poly. Alg. ja ausgibt.

Bsp.: $L = SAT$ = Menge erfüllbarer Formeln

v = erfüllende Belegung

Polynomialzeitalgorithmus = „Einsetzen" von v und „Ausrechnen" der Formel.

L = Menge der 3-färbbaren Graphen

V = 3-Färbung der Eingabe

Algo: Teste die Färbung

18.06.2018

NP ~> Menge von Sprachen

$L \subseteq \Sigma^*$ Sprache,

$L \in NP$

$\Leftrightarrow$ Es gibt einen Polynomialzeitalgorithmus

$P(w,u)$ so dass gilt: Für alle $w \in \Sigma^*$

$w \in L \Leftrightarrow$ es gibt ein u

$|u| \leq O(|w|^k)$ für ein k Konstant.

und $P(w,u) = 1$. (akzeptieren)

Probleme in NP:

Travelling Salesman Problem: $u \hat{=}$ Rundreise

Erfüllbarkeitsproblem: $u \hat{=}$ Belegung

Graphfärbung: $u \hat{=}$ Färbung

$P \hat{=}$ überprüfen der Lösung!

18.06.2018

- Probleme in NP sind entscheidbar!

↳ Exponentialzeit. Größe der Lösung ist durch Länge der Eingabe beschränkt (nach Definition).

⇝ Alle Lösungskandidaten absuchen!

$|u| \leq O(|w|^k)$

↳ $|\Sigma|^{O(|w|^k)}$ Laufzeitschranke.

Könnte $L \in NP$ nicht auch in Polynomialzeit entschieden werden?

⇝ Vermutung: Nein! Nicht bewiesen!

NP steht für nichtdeterministische Polynomialzeit

(P steht für Polynomialzeit)

auf Turing Maschine

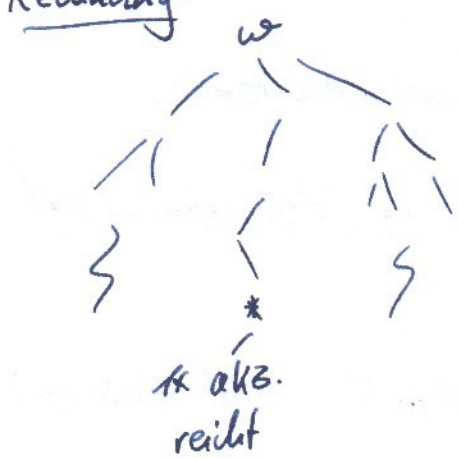
nichtdet. TM: $\delta(q, A) = \{(q_1, A_1, L), (q_2, A_2, \dots), \dots, (q_n, A_n, \dots)\}$ ← Auswahl

Ist M nichtdet. TM, dann ist $L \subseteq \Sigma^*$ die akzeptierte Sprache

$\Leftrightarrow$ Für $w \in \Sigma^*$ gilt:

$w \in L \Leftrightarrow$ Es gibt Rechnung mit Eingabe w, die in einen akzeptierenden Endzustand führt.

Rechnung:



SAT durch nichtdet. TM akzeptiert?

↳ Ja!

1) nichtdet. 0-1-Folge erzeugen

2) überprüfen, ob diese Folge eine erfüllende Belegung ist.

SAT als Sprache:

SAT = Menge der Formeln, die wahr gemacht werden können.

(UNSAT = Menge der Formeln, die widersprüchlich sind

UNSAT $\notin$ NP (Vermutung, nicht bewiesen))

18.06.
2018

NTM: Am Anfang

1. Fahren ans Ende der Eingabe. q_1
2. $\delta(q_1, \square) = \{(q_1, 1, R), (q_1, 0, R), (q_2, \square, N)\}$
3. In q_2, setze die 0-1-Folge in die Formel ein.

($L \in NP \Rightarrow L$ auf nichtdet. TM akzeptiert.)

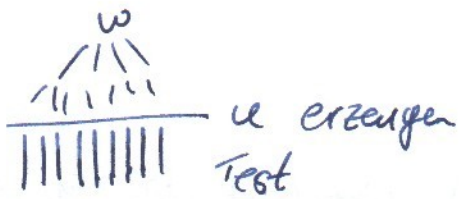
Laufzeit einer n-det. TM auf Eingabe w.

= #Schritte der kürzesten akzeptierenden Rechnung für $w \in L$ (d.h. wenn w akz. wird)

und undef. wenn w nicht akzeptiert.

Satz: $L \in NP$ (Def. mit $P(w,u)$)

$\Rightarrow$ L in polynomialer Zeit durch nichtdet. TM akzeptiert.

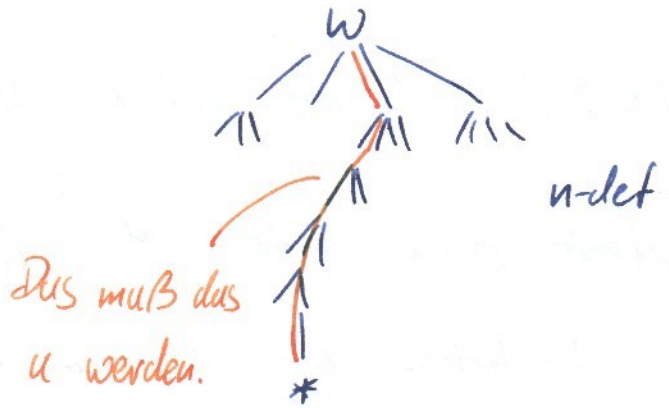


Satz: L durch n-det. TM in Polynomialzeit akzeptiert

$\Rightarrow L \in NP$ (Def. mit $P(w,u)$)

Beweis: So sieht die TM w akzeptiert aus:

$m = |w|$ Länge $p(u)$



Suche Poly.-alg. $P(w,u)$

Zu w gehört das u:

$\delta(q_0, \triangleright A) = \{ \dots \}$

$(\delta(q_0, A), \delta(q_1, B, R)),$
$\dots$

$p(u)$ viele

gewählte Rechenschritte der TM

$P(w,u)$ simuliert die in u angegebene Rechnung der TM (~~auf~~ w)

(mit überprüfen)

■

15.06.2018

Satz von Cook

SAT = Menge der erfüllbaren Formeln

hat folgende Eigenschaften:

- SAT $\in$ NP
- Für alle $L \in NP$ gilt:

  Für w Eingabe zu ~~von~~ L lässt sich in Polynomialzeit F_w Eingabe zu SAT hinschreiben, so dass gilt:

  $$w \in L \Leftrightarrow F_w \in SAT.$$

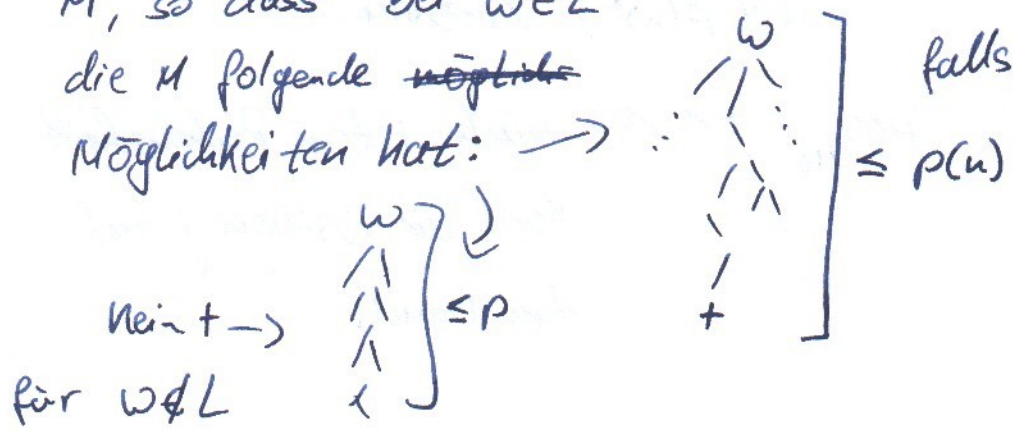
SAT ist NP-hart!

$\rightsquigarrow$ SAT NP-vollständig.

Beweis: · SAT $\in$ NP ist klar ✓

· Sei $L \in NP$ beliebig!

Also haben wir Polynom $p(n)$ und n-det. TM M, so dass bei $w \in L$ die M folgende ~~möglichen~~ Möglichkeiten hat:



M hat Zustände $z_1, \dots, z_k$ (Zu w bauen wir F_w)
$|w| = n$.

Betrachte TM auf $p(n)$ Schritten.

Aussagenlogische ~~Wert~~ Variablen:

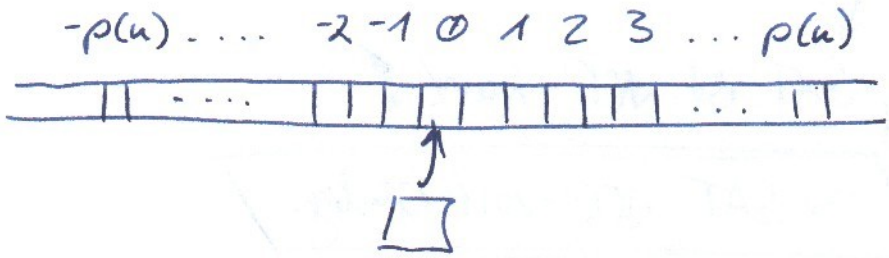
$Zust_{0,1}, \dots, Zust_{0,k}$ (Zustände)

$\vdots \qquad \vdots$

$Zust_{p(n),1}, \dots, Zust_{p(n),k}$ $O(p(n))$ Var.

Idee: $Zust_{i,j} = 1 \Leftrightarrow$ nach i-tem Schritt ist der Zustand j.

(Kopfpositionen)



$Pos_{0,-p(n)}, \dots, Pos_{0,0}, \dots, Pos_{0,p(n)}$

$\vdots$

$Pos_{p(n),-p(n)}, \dots \qquad \dots, Pos_{p(n),p(n)}$

$O(p(n)^2)$ Variablen.

$Pos_{i,j} = 1 \Leftrightarrow$ nach i-tem Rechenschritt Kopf an Position j auf dem Band.

18.06.
2018

$\Gamma = \{a_1, \dots, a_L\}$

$band_{0,-p(n),1}, \dots \qquad band_{p(n),p(n),L}$

$band_{i,j,k} = 1 \Leftrightarrow$ Nach i an Pos j das a_k

$$O(p(n)^2 \cdot L) = O(p(n)^2)$$

~

<u>Für den Anfang</u>: Alle Variablen mit erstem Index 0 setzen. (z1 Anfangszustand)

$Zust_{0,1} \wedge pos_{0,0} \wedge band_{0,0,a_{i_1}} \wedge$
$band_{0,1,a_{i_2}} \wedge$
$\vdots$
$band_{0,n-1,a_{i_n}} \wedge$
$band_{0,n,\square} \wedge \dots$
$band_{0,p(n),\square} \wedge$
$band_{0,-1,\square} \wedge \dots \wedge band_{0,-p(n),\square}$

($w = a_{i_1} a_{i_2} \dots a_{i_n}$ (i_1 etc. $\in (1,\dots,L)$)

($i = 0$ Eingabekonfiguration z1 Startzustand)

19.06.
2018

Angenommen

$$\delta(z_1, a) = \{(z_2, b, L)\}, \quad a_{i_1} = a$$

Nach 1. Stritt

$\wedge$

$\vdots$

$\wedge$

$\rightsquigarrow$ Vor. sollen gemäß der Rechnung richtig stehen!

Nach $p(n)$. Schritt

δ von oben wird zu: (für $i=0$)

$$(z_{0,1} \wedge pos_{0,0} \wedge band_{0,0,i_1}) \rightarrow (z_{1,2} \wedge pos_{1,-1} \wedge band_{0,0,b})$$

Tatsächlich für alle i:

$$\bigwedge_{i=0}^{p(n)-1} \bigwedge_{j=-p(n)+1}^{p(n)} \left((z_{i,1} \wedge pos_{i,j} \wedge band_{i,j,i_1}) \rightarrow (z_{i+1,2} \wedge pos_{i,j-1} \wedge band_{i,j,b})\right)$$

19.06.
2018

Nichtdeterminismus in der TM:

noch ein Bsp.:

$$\delta(z_1, a) = \{(z_2, b, N), (z_3, c, R)\}$$

dafür:

$$\bigwedge_i \bigwedge_j \left(z_{i,1} \wedge pos_{i,j} \wedge band_{i,j,i_1}\right) \rightarrow \Big(\left(z_{i+1,2} \wedge pos_{i+1,j} \wedge band_{i+1,j,b}\right) \vee \left(z_{i+1,3} \wedge pos_{i+1,j+1} \wedge band_{i+1,j,c}\right)\Big)$$

Alles zusammenunden für jedes $\delta(z, a)$.

Am Ende: Test auf erreichten Endzustand (akzeptierend)

$$\wedge\ zust_{p(n),e}$$

(nach ~~im~~ Schritt $p(n)$ im Zustand z_e)

$\delta(z,a)$ undef $\rightsquigarrow$ dann alles so lassen

(insbesondere Zustand bleibt gleich)

jetzt noch: ~~wo~~ wo der Kopf nicht ist, bleibt alles gleich

$$\bigwedge_{i=0}^{p(n)-1} \bigwedge_{j=-p(n)}^{p(n)} \Big((\neg pos_{i,j} \wedge band_{i,j,a}) \rightarrow band_{i+1,j,a} \Big)$$

für alle $a \in \Gamma$

weiter dabei: Formeln, die folgendes sagen:

$$\bigwedge_{i=0}^{p(n)} \Big((\text{genau } \underline{\text{ein}}\ z_{i,j} \text{ wahr}) \wedge (\text{genau ein } pos_{i,j} \text{ wahr}) \wedge \bigwedge_{j=-p(n)}^{p(n)} (\text{genau ein } band_{i,j,a} \text{ wahr}) \Big)$$

~~für alle Paare (i,j)~~

Allgemein: $x_1, \dots, x_n$ Variablen, genau eine soll wahr sein!

$$(x_1 \vee \dots \vee x_n) \wedge (\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge \dots \wedge (\neg x_1 \vee \neg x_n) \wedge (\neg x_2 \vee \neg x_3) \wedge \dots \wedge (\neg x_2 \vee \neg x_n) \wedge \dots \wedge (\neg x_{n-1} \vee \neg x_n)$$

(Länge $O(n^2)$)

19.06.
2018

Größe der Gesamtformel:

$$O(p(n)^3)$$

Variable $O(p(n)^2)$

∎

Reduktionen: Weitere NP-vollständige Probleme

Satz: 3-KNF ist NP-vollständig

(
$(x_1 \vee x_2 \vee x_3)$
≤3 Lit. pro Klausel

(↳ NP-hart und $\in NP$)

Beweis: $F \mapsto$ 3-SAT-Formel

(
beliebige Formel.

Formel als Syntaxbaum schreiben.
Beweis durch Beispiel

$$\underbrace{(A \wedge B)}_{N_6} \vee \Big(\underbrace{\underbrace{(\neg C)}_{N_4} \rightarrow \underbrace{(D \vee \neg A)}_{N_2}}_{N_3} \Big)$$

N_1

N_5

$$(N_6 \Leftrightarrow (A \wedge B)) \wedge (N_1 \Leftrightarrow \neg A)$$

$$\wedge (N_2 \Leftrightarrow (D \vee N_1))$$

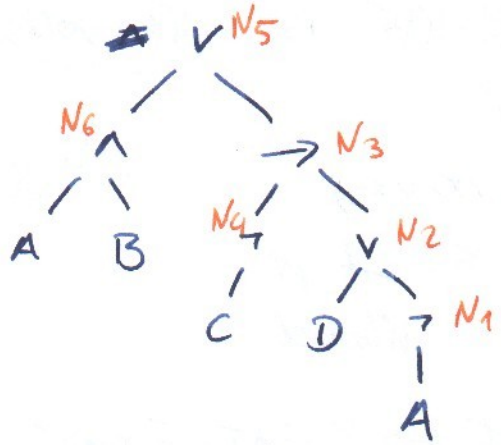
$$\wedge (N_4 \Leftrightarrow \neg C)$$

$$\wedge (N_3 \Leftrightarrow (N_4 \rightarrow N_2))$$

$$\wedge (N_5 \Leftrightarrow (N_6 \vee N_3))$$

$$\wedge \underline{\underline{N_5}}$$

$\Rightarrow$ Lineare Größe in F.



<u>Aber</u>: 2-KNF geht in Polynomialzeit!

19.06
2018

K-Clique in einem Graphen:

K Knoten mit allen $\binom{K}{2}$ Kanten.

Eingabe: Graph

$K = \frac{n}{2}, \frac{n}{4}$ (n Knotenzahl) (K abhängig von n)

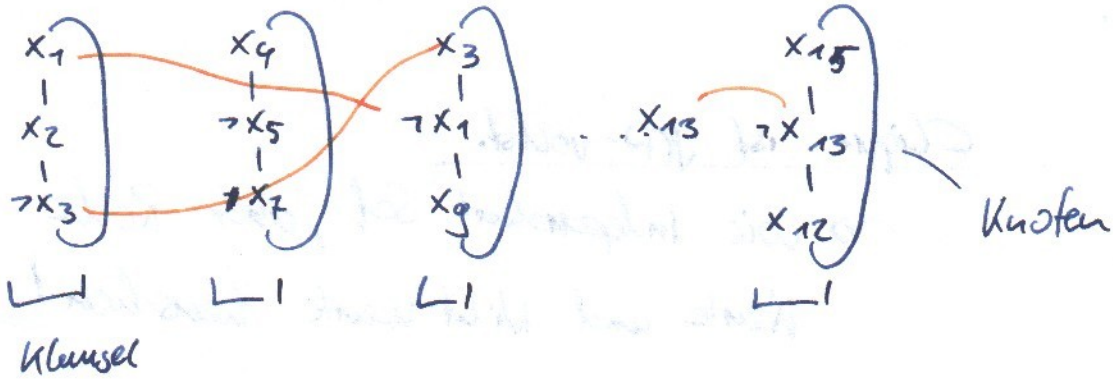
Frage: Hat Graph Clique von K (oder mehr) Knoten?

Satz: Clique ist NP-vollständig.

Unabhängige Menge: wie Clique ~~mit~~ nur ohne jede Kante.

Beweis: unabh. Menge $\in$ NP $\rightarrow$ klar.

Reduktion von 3-SAT



Erfüllbarkeit bei KNF $\hat{=}$ „widerspruchsfreier Weg" durch die Klauseln

(x und ¬x kommen nicht gemeinsam vor)

⇒ Kante zwischen allen Knoten, die x bzw. ¬x repräsentieren

⇒ In den Klauseln alle Kanten!

F erfüllbar ⇒ Unabhängige Menge der Größe m

F nicht erfüllbar ⇒ Keine unabh. Menge der Größe m.

(Keine 2 Knoten aus 1 Klausel möglich! Kein widerspruchsfreier Weg, da unerfüllbar)

Clique ist NP-vollst.

⇝ wie Independent Set, aber Rolle von Kante und Nicht-Kante tauschen!

25.06.
2018

Knotenüberdeckung (Vertex Cover)

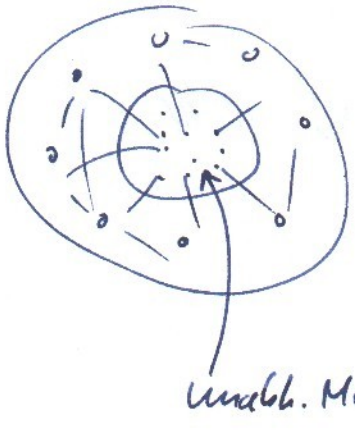
Eingabe: $G = (V, E)$, k

Frage: Gibt es $V' \subseteq V$, $|V'| \leq k$ so dass jede Kante von G hat einen Knoten in V'.

$\rightsquigarrow$ Ist NP-vollständig.

Beweis: · Ist in NP. ✓

· ~~Klique~~ $\mapsto$ Knotenüberdeckung
Unabh. Menge



Ist $W \subseteq V$ unabhängig
$\Leftrightarrow V \setminus W$ ist ~~ein~~ Knoten-überdeckung

(G, k für unabh. Menge) $\mapsto$ (G, $n-k$ für Knotenüberdeckung)

■

Subset Sum

Eingabe: $a_1, \dots, a_k \in \mathbb{N}$, $b \in \mathbb{N}$

Frage: Gibt es $\{i_1, \dots, i_m\} \subseteq \{1, \dots, m\}$, m versch. Zahlen mit

$$a_{i_1} + a_{i_2} + \dots + a_{i_m} = b.$$

Ist NP-vollständig. Von 3-SAT ausgehend übersetzen.

$F = (x_1 \vee x_2 \vee \neg x_3) \wedge \dots$ n Variablen, m Klauseln

haben Zahlen:

$b, \quad v_1, \dots, v_n \quad v_1', \dots, v_n' \in \mathbb{N}$

Etwa: v_1 ist so gebaut:

~~$v_1 = \text{Ziffer}_1 \dots \text{Ziffer}$~~

~~$v_1 = z_F$~~

$v_1 = \underbrace{z_{i_1} \dots z_{i_m}}_{\text{m Stück}} \underbrace{z_{i_{m+1}} \dots z_{i_{m+n}}}_{\text{n Stück}}$

$v_1 = 1\,1\,0 \dots 0 \quad 1\,0 \dots 0$

( ) (x_1

x_1 in Klausel 1 und 2

25.06.
2018

$F = (x_1 \vee x_2 \vee x_3) \wedge \ldots$

$v_1 = 1 \ldots . | 1 0 \ldots 0$ $\quad (x_1 \vee x_1 \vee \ldots)$

$v_2 = 1 \ldots . | 0 1 0 \ldots 0$ $\quad \Downarrow$

$v_1 = 2 \ldots$???

braucht man das?

v_i' analog

Zielwert b: $\underbrace{4\,4 \ldots 4}_{m}\,\underbrace{3\,3 \ldots 3}_{n}\,\underbrace{1\,1 \ldots 1}_{n}$

dazu noch dabei: (für jede Klausel)

$c_1 = 1 0 \ldots 0 | 0 \ldots 0$ $\quad d_1 = 2 0 \ldots 0 | 0 \ldots 0$

$c_2 = 0 1 \ldots 0 | 0 \ldots 0$

$\vdots$ $\quad \vdots$

$c_m = \underbrace{0 \ldots 1}_{m} | \underbrace{0 \ldots 0}_{n}$ $\quad d_m = \underbrace{0 \ldots 2}_{m} | \underbrace{0 \ldots 0}_{n}$

Ein Beispiel:

$$(x_1 \vee \neg x_3 \vee x_5) \wedge (\neg x_1 \vee x_5 \vee x_4) \wedge (\neg x_2 \vee \neg x_2 \vee x_5)$$

$v_1 = 1000|100000$ $\quad$ $v_1' = 0100|100000$

$v_2 = 0000|010000$ $\quad$ $v_2' = 0002|010000$

$v_3 = 0000|000100$ $\quad$ $v_3' = 1000|000100$

$v_4 = 0100|000010$ $\quad$ $v_4' = 0000|000010$

$v_5 = 111|000001$ $\quad$ $v_5' = 0000|000001$

$b = 444|11111$

$c_i = \ldots$

$d_i = \ldots$

25.06.2018

Partition

Eingabe: $a_1, \dots, a_n$ ~~Eingabe~~

Gibt es $\{i_1, \dots, i_L\} \subseteq \{1, \dots, n\}$ alle verschieden

$$\overbrace{a_{i_1} + a_{i_2} + \dots + a_{i_L}}^{P_1} = \overbrace{a_{j_1} + \dots + a_{j_K}}^{P_2}$$

$$\{j_1, \dots, j_K\} = \{1, \dots, n\} \setminus \{i_1, \dots, i_L\}$$

SubsetSum $(a_1, \dots, a_n, b)$ $\mapsto$ Partition $(a_1, \dots, a_n, b+1, M-b+1)$

$M = a_1 + \dots + a_n$

ohne +1 immer lösbar!

26.06.2018

Partition mit

$a_1 + \dots + a_n = M$

und Eingabe

$(a_1, \dots, a_n, b, M-b)$

hat Lösung

$P_1 = (a_1, \dots, a_n)$, $|P_1| = M$

$P_2 = (b, M-b)$, $|P_2| = M$

mit der Eingabe $(a_1, \dots, a_n, b+1, M-b+1)$

geht das so einfach nicht mehr.

$(b+1)$ und $(M-b+1)$ können nicht beide gewählt werden, denn

$$(b+1)+(M-b+1) = M+2 > a_1+\ldots+a_n$$

also: ~~o.B.d. sei $(b+1) \in P_1$ und~~

sei $(M-b+1) \in P_1$ und $(b+1) \in P_2$

dann: gibt es a_i mit $\sum_{i \in I} a_i = b$, dann

diese in P_1:

$$P_1 = \{(M-b+1)\} \cup \{a_i \mid i \in I\}$$

$$|P_1| = M-b+1+b = M+1$$

und den Rest in P_2

$$P_2 = \{(b+1)\} \cup \{a_i \mid i \notin I\}$$

$$|P_2| = b+1 + \underbrace{M-b}_{\text{da } \sum a_i = M \text{ und } \sum_I a_i = b} = M+1$$

d.h. Lösung f. Subsetsum $\Leftrightarrow$ Lösung f. Partition ∎

26.06.
2018

Bin Packing

K Behälter, Größe b

$a_1, \dots, a_n \quad a_i \leq b$

Verteilen der a_i auf Behälter, so dass kein Behälter überläuft.

Algorithmus (Versuch)

- Der größe nach in den jeweils leersten Behälter tun.

Bsp.: $K=2$, $b=6$

$a_1 = a_2 = 3 \qquad a_3 = a_4 = a_5 = 2$

wird nicht gefunden!

So einfach funktioniert das nicht!

---

von Partition aus reduzieren

Part. $\longmapsto$ Bins: $K=2$, $b = M/2$

$a_1, \dots, a_n \qquad (a_1, \dots, a_n)$

Hamiltonkreis im gerichteten Graph

Hamilton Kreis = Kreis, der bei n Knoten genau die Länge n hat.

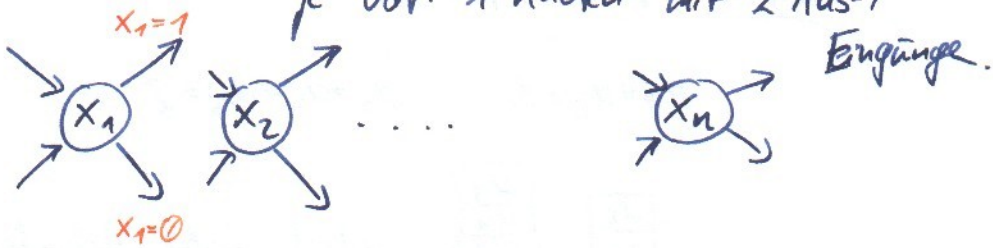
⇒ auch NP-vollst.
(zuerst im ger. Graph)

- Reduktion von 3-SAT aus (Weg durch die Klauseln, der keinen Widerspruch hat)

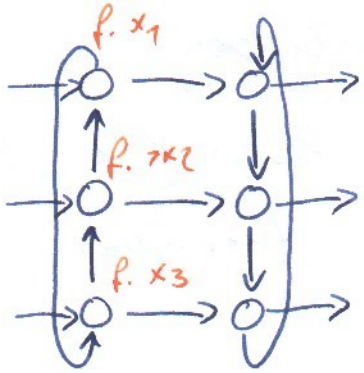
Variablen $x_1, \dots, x_n$
Klauseln $G_1, \dots, G_m$

↓

Graph:

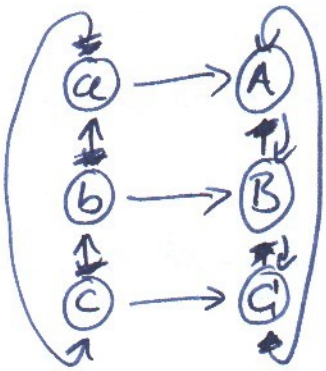


Für jede Klausel: $(x_1 \vee \neg x_2 \vee x_3)$ Teilgraph mit 6 Knoten



~~26~~.06.2018

Zusammenhang des Kleensel-Teilgraph zum Hamiltonkreis:



Möglichkeit:

1) → a → A →
später
→ b → B → B
später
→ c → C' →

2) → a → c → b → B → C' → A →

3) → a → c → C' → A → ;
→ b → B →

4) → b → a → A → B → ;
→ c → C' →

5) → c → b → B → C' → ;
→ a → A →

...

Beobachtung: 1) Wo man reingeht, muß man auch rausgehen (a rein ⇒ A raus, usw.)

2) Möglichkeit da durchzugehen:

1. $\rightarrow \underline{\underline{a}} \rightarrow A \rightarrow \ldots \rightarrow \underline{\underline{b}} \rightarrow B \rightarrow \ldots \rightarrow \underline{\underline{c}} \rightarrow C' \rightarrow \ldots$

2. $\rightarrow \underline{a} \rightarrow c \rightarrow \cancel{\#}b \rightarrow B \rightarrow C' \rightarrow A \rightarrow \ldots$

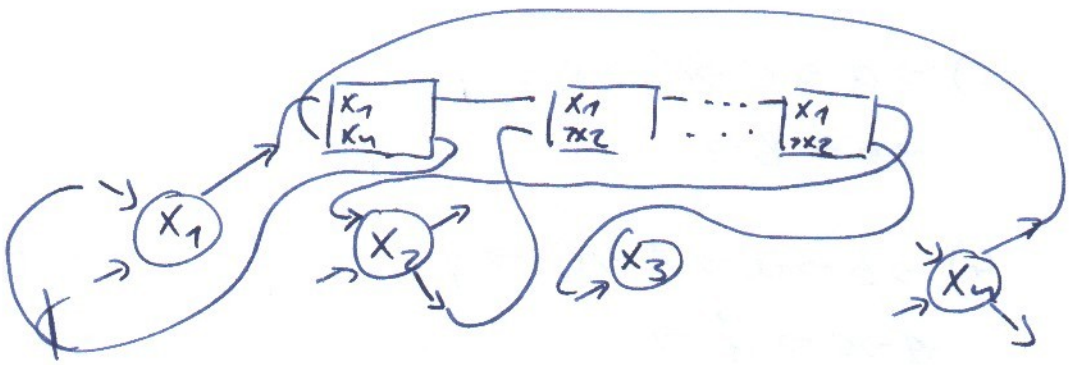
   Ebenso b, c als „erstes"

3) $\rightarrow \underline{\underline{a}} \rightarrow A \rightarrow \ldots \rightarrow \underline{\underline{c}} \rightarrow b \rightarrow B \rightarrow C \rightarrow \ldots$

$\rightarrow \underline{\underline{a}} \rightarrow c \rightarrow C' \rightarrow A \rightarrow \ldots \rightarrow \underline{\underline{b}} \rightarrow B \rightarrow \ldots$

$\underline{\underline{x}}$ sind wahr, rest mit Symmetrie

Klauselgraphen mit Variablen verknüpfen.

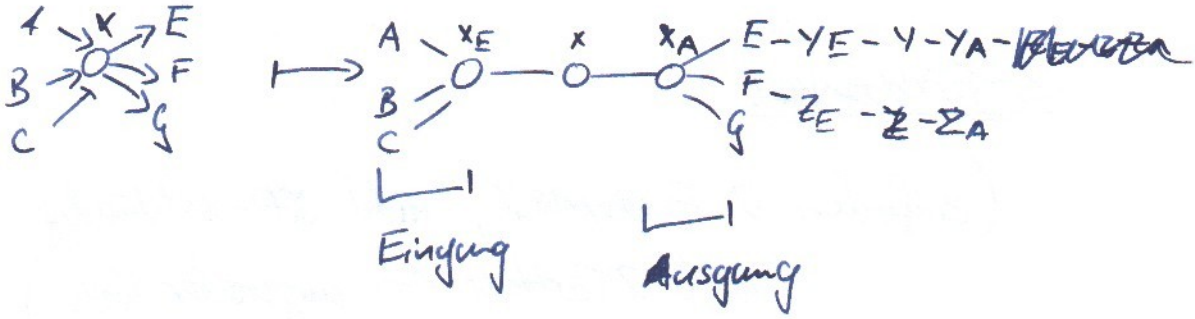


Von Var. zu ~~passenden~~ passenden Eingang der Klausel, dann durch die Klausel, dann zur nächsten Variable, zum Schluß wieder zu x_1 zurück.

Erfüllbarkeit $\Leftrightarrow$ Hamiltonkreis existiert

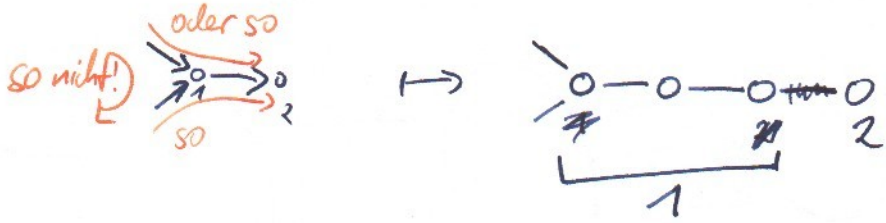
26.06.2018

Ungerichteter Hamiltonkreis



02.07.2018

gerichteter Hamiltonkreis ⇝ unger. Hamiltonkreis



Kürzester Weg bei neg. Kanten

Hamiltonkreis → ~~Kw. von 1 zu~~ Es gibt einen Knoten

T ~~einen Knoten = (n-1)~~ x so dass

Kw von 1 ~~und~~ nach

x Länge (n-1) hat.

Suche KW's von

1 zu allen Knoten

⇝ Länge zu x Länge (n-1)

+ Kante x → 1

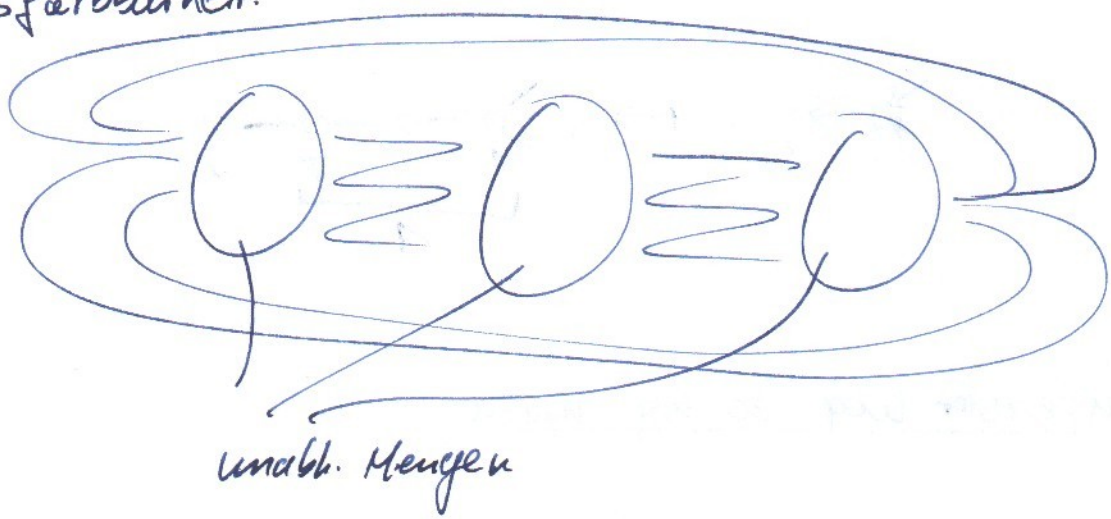
Längster einfacher Weg

geht genauso, Kantengewicht überall 1 !

3-Färbbarkeit

(einfacher: 2-Färbbarkeit; nicht NP-vollständig.
nicht 2Färbbar $\Leftrightarrow$ ungerader Kreis)

3färbbarkeit:

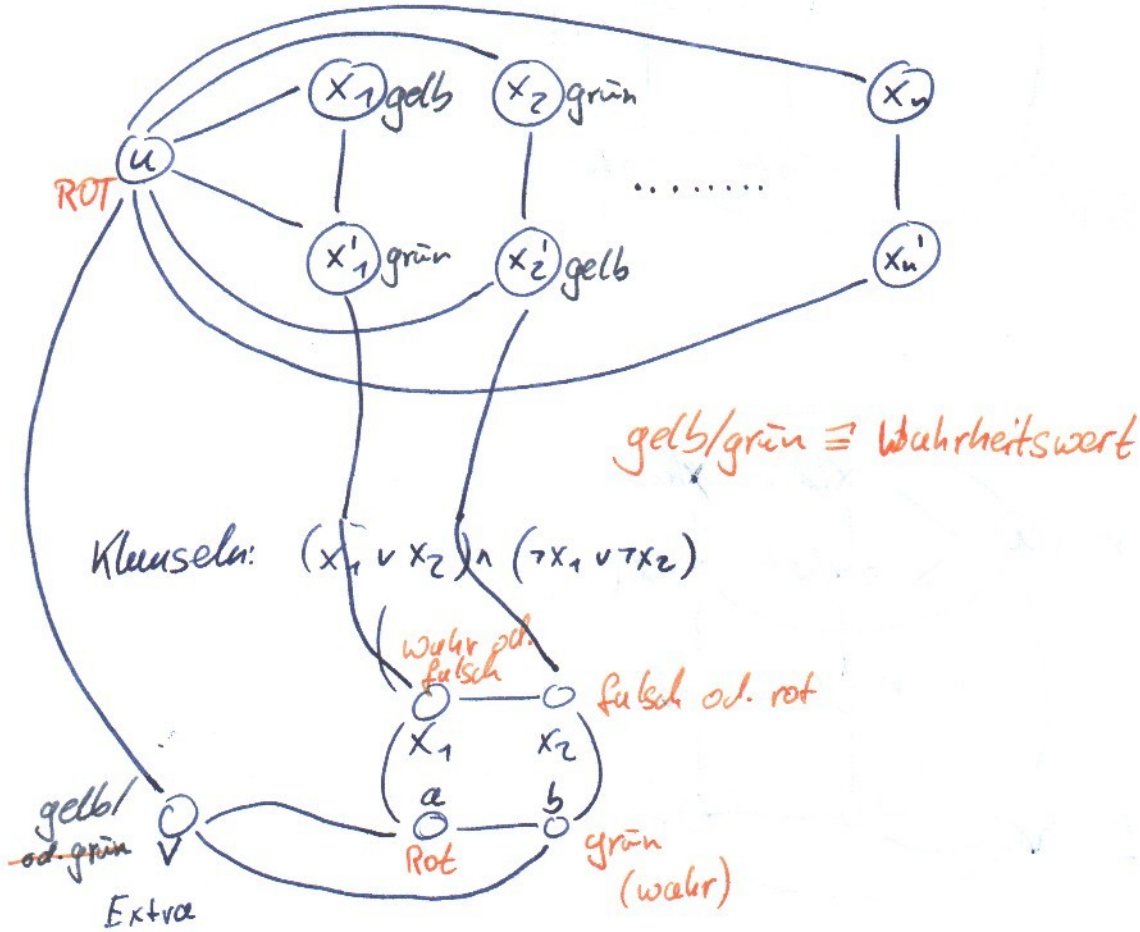


3-SAT $\leq_p$ 3-Färbbarkeit

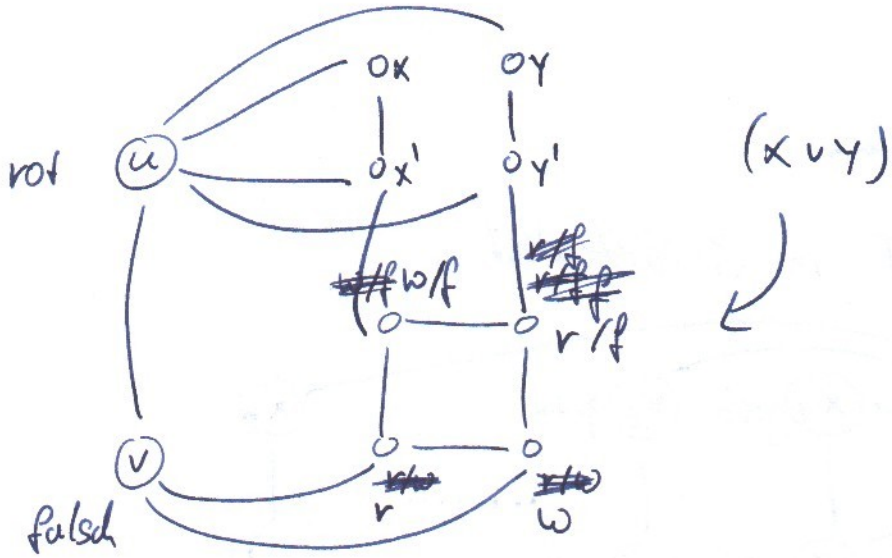
02.07.
2018

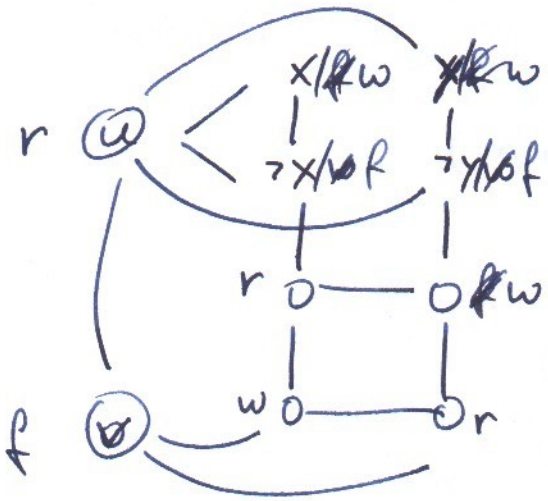
Zum Verstehen

2SAT in 3-Färbbarkeit



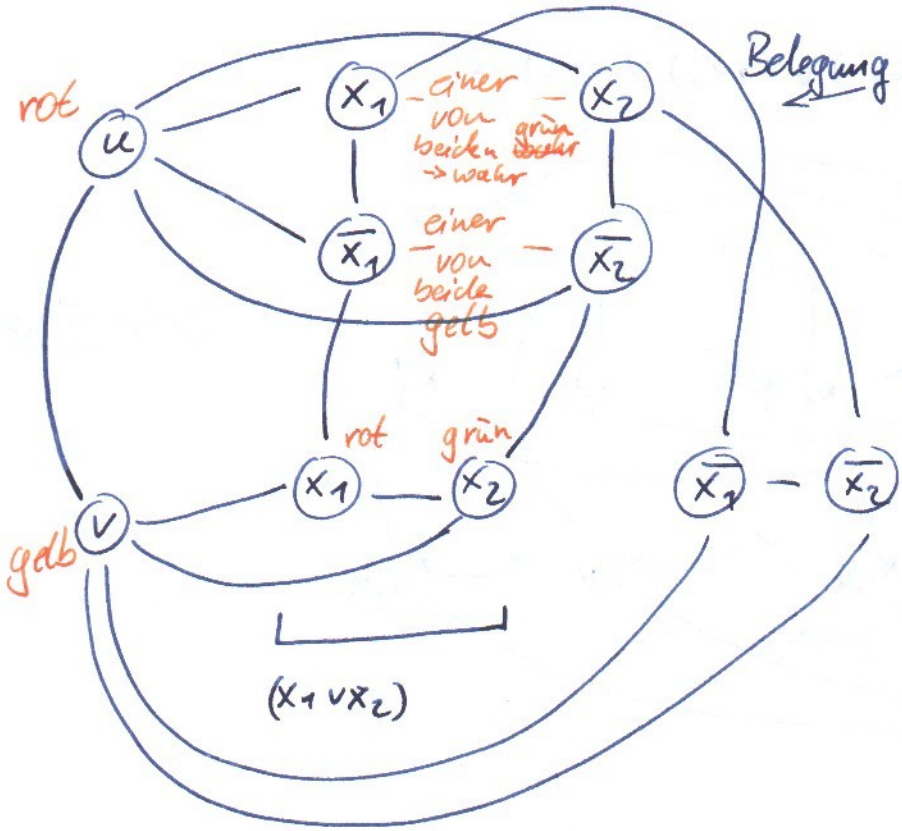
Farbe hier
steht für
„falsch"


x
y
x'
y'
w/f
r/f
r
w
(x ∨ y)


r
f
r
w

02.07.
2018

$(X_1 \vee X_2) \wedge (\neg X_1 \vee \neg X_2)$



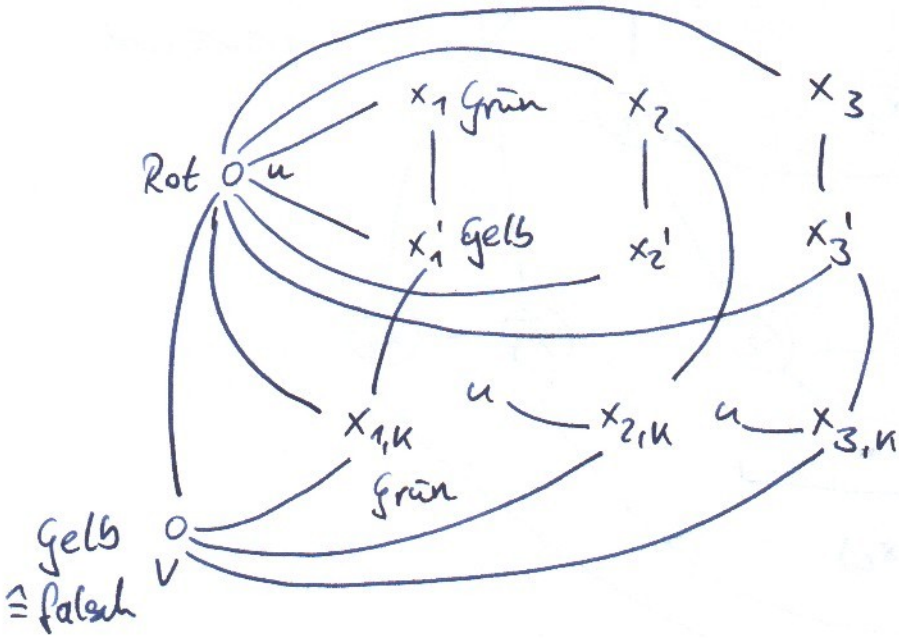
rot ≙ kein WW
gelb ≙ falsch
grün ≙ wahr

Färbungen: u v $V.X_1$ $V.\overline{X_1}$ $v.x_2 V.\overline{x_2}$ $U_1 x_1$ $U_1 x_2$
rot gelb ... ...

03.07.
2018

1-SAT

$(x_1) \vee (\neg x_2) \wedge (x_3) \wedge \ldots$



⇒ Kommt a und ¬a vor, dann unerfüllbar und auch keine gültige Färbung möglich!

x_i' steht für $\neg x_i$

⇝ Knoten aus der Klausel immer an gegenteil anschließen!

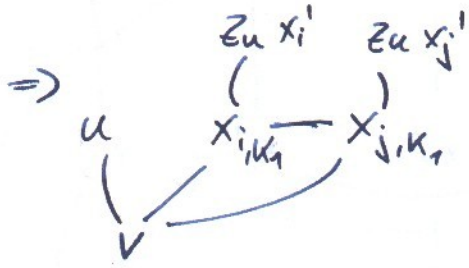
also: x in Klausel → Kante zu x'

⇝ Färbung der x_i in den oberen Dreiecken entspricht Belegung!

03.07.
2018

2-SAT

Klausel: $(x_i \vee x_j)$



Färbung: u rot
v gelb (falsch)

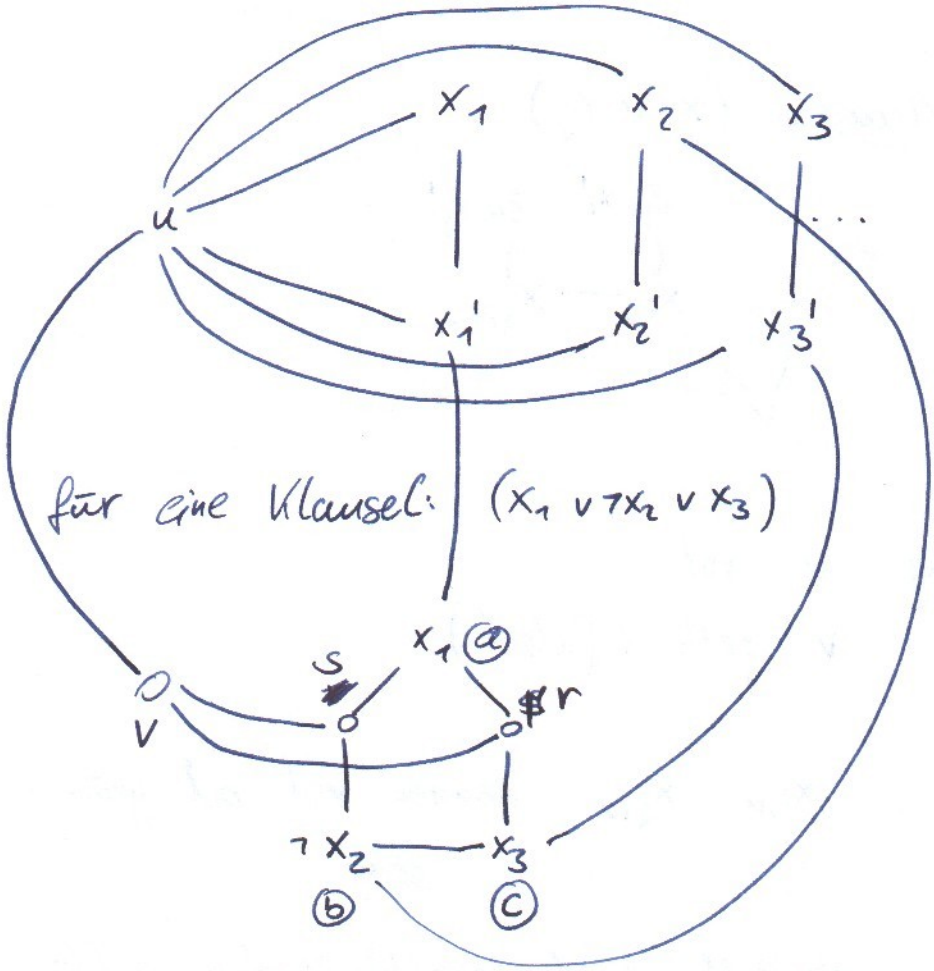
$x_{i,u}$ $x_{j,u}$ können rot od. grün sein

$\Rightarrow$ falls beide Variablenknoten gelb, dann Klausel falsch!
und keine Färbung möglich!

$\Rightarrow$ zu jeder Belegung, die erfüllt, gibt es mind. eine Färbung.

$\Rightarrow$ aus gültiger Färbung ergibt sich eine erfüllende Belegung.

3-SAT (darum geht es ja eigentlich!)



u rot

v gelb (falsch)

r s x_1 $\neg x_2$ x_3 Färbung im Häuschen

~~rot/grün~~ ~~rot/grün~~

~~r r wie x_1 wie x_2 wie x_3~~
~~oben oben oben~~

oben fest

x_1 x_2 x_3

~~r~~ gr. gr. r ← gr ge gr

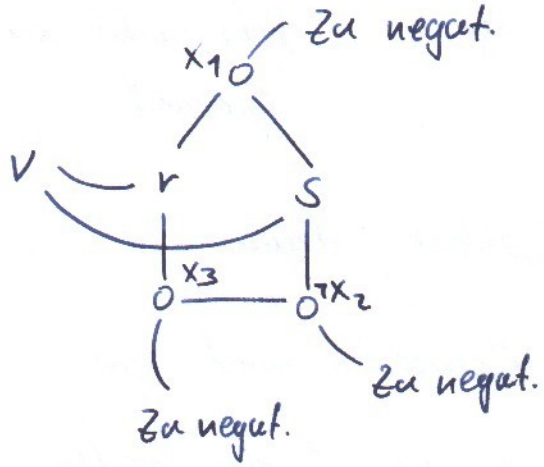
03.07.
2018

nochmal neu

$(x_1 \vee \neg x_2 \vee x_3)$

u rot

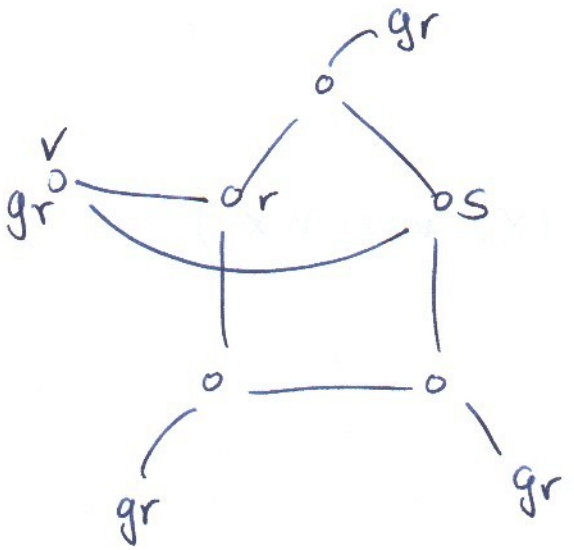
v grün (wahr)



x_1	x_2	x_3	x_1	x_3	$\neg x_2$	r	s
gr	ge	gr	gr	rot	gr.	ge	rot
gr	ge	ge	gr	rot	gr.	ge	ge
gr	gr	ge	gr.	rot	ge.	~~rot~~ ge.	rot
ge	gr	gr	ge	gr.	ge	rot	rot
ge	ge	gr	ge	gr.	ge	rot	rot

falls erfüllende Belegung ~~Klausel~~ Klausel wahr → 3-färbung mögl.

Alle 3 ~~Nacht~~ Nachbarn grün → dann darf das nicht färbbar sein!



nur noch gelb/rot übrig

⇒ umgeb. Kreis geht nicht zu Färben!

⇝ falls ~~3farbig~~ 3färbbar, denn

für jede Klausel mind. ein Nachbar hat die Farbe gelb

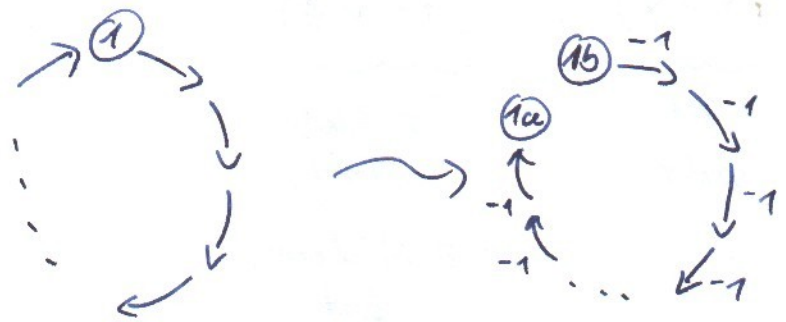
⇒ erfüllende Belegung ergibt sich aus den grünen Variablenknoten.

⇝ alle 8 Möglichkeiten der Färbung der Nachbarn aufzählen/untersuchen!

03.07.
2018

Reduktion:

Hamilton-Kreis ↦ K.W. (einfach) mit neg. Kanten



HC ⟺ KW mit Länge(-n)

Nicht NP-vollständig:

2-SAT geht in Polynomialzeit

allgemein: Backtracking, Davis-Putnam

DP(F):

1. F enthält (x) und (¬x), dann unerfüllbar!
2. F ist ohne Klausel, dann erfüllbar!
3. wähle Variable x von F zum Ausprobieren
4. $F_1 := F_{|x=1}$; ~~DP(~~
   if DP(F_1) = erfüllbar then ret. erfüllbar
   else
   $F_0 := F_{|x=0}$
   return DP(F_0)
   end.

Wie sieht das bei 2-SAT aus?

Das Setzen, etwa $x=1$

$F = \ldots (x \vee y) \ldots (\neg x \vee z) \ldots$

$(x \vee y)$: wird wahr

$(\neg x \vee z)$: z bleibt übrig, z ist dann ~~fest~~ bestimmt auf $z=1$

Danach setze weiter $z=1$ (einer Klausel).

$F = (z \vee ..) \ldots (\neg z \vee u)$

$\downarrow$

$u=1$

usw. solange einer-Klauseln auftreten.

Wie hört das auf?

1. Fall: Keine Einer mehr, d.h. nur noch Zweier

⇒ Es reicht ohne Backtracking weiterzumachen

(F ist erfüllbar ⇔ Rest ist erfüllbar)

(Rest ist unabhängig von x)

2. Fall: $(v) \wedge (\neg v)$

⇒ Setzung auf $x=1$ am Anfang bringt nichts.

⇒ es muß $x=0$ gesetzt werden!

⇒ geht dann in Polynomialzeit $O(n^2 \cdot |F|)$

03.07.
2018

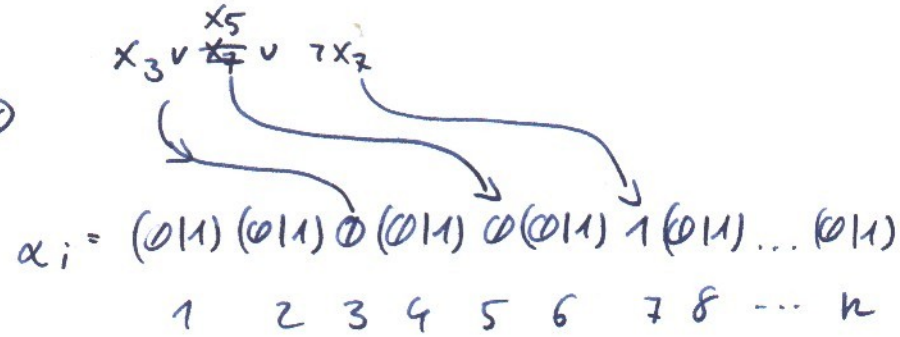
Äquivalenz regulärer Ausdrücke

NP-hart

Reduktion $3\text{-SAT} \leq_p$ Äq. Reg. A.

3SAT

$x_1, \dots, x_n \rightsquigarrow$



$\alpha = \alpha_1 | \alpha_2 | \dots | \alpha_m$

Formel unerfüllbar $\Leftrightarrow$ Sprache von α ist $\{0,1\}^n$

Belegung erfüllt Klausel nicht $\Leftrightarrow$ Belegung ist in der Sprache der Klausel (α_i)

$\beta = (0|1)^n$

erfüllbar $\Leftrightarrow$ $\alpha \neq \beta$

Laufzeit beim Übergang

~~RAM → TM~~

TM → RAM

Das ist einfach. TM in $O(p(n))$, dann auch RAM in $O(p(n))$

↝ Simulation der TM genau wie beim Übergang WHILE-Programme → TM.

Übergang RAM → TM

~~Geht z.B. auf~~

Geht auf Mehrband TM, braucht eine Reihe von Bändern:

~~B: für B~~

P: für Programm
B: für Befehlszähler
M: für Speicher
A: für Akkumulator / Rechnung

} Aufwand!

Zählen der Laufzeit

- bisher: uniformes Kostenmaß, jeder Befehl zählt gleich

Problem: Bei sehr großen Zahlen müssen viele Bits bearbeitet werden, beim Übergang zur Turing-Maschine.

Addition: 2 Zahlen mit je n Bits zu addieren dauert auf der TM $O(n^2)$ Schritte
~~(sogar $O(n^2)$ wenn man sich nicht so geschickt anstellt)~~

Zwischen RAM und TM liegt hier ein Faktor von n^2
das ist noch kein Problem für die Polynomialzeit aber selber nicht besonders realistisch!

wie bekommen wir auf der RAM viele Bits (große Zahlen)?

■

Eine Addition erzeugt immer nur höchstens ein neues Bit:

haben wir 2 Zahlen x, y mit m und n Bits, dann ist

$x \leq 2^m$ und $y \leq 2^n$ und für $m \geq n$

gilt $x+y \leq 2^m + 2^n \leq 2 \cdot 2^m \leq 2^{m+1}$

also eine Zahl mit $m+1$ Bit.

$\rightsquigarrow$ mit $p(n)$ Additionen können wir also höchstens $p(n)$ viele Bits erzeugen.

Bei Multiplikationen sieht das nicht mehr so günstig aus!

Multiplikation von 2 n-Bit Zahlen auf der TM sollte in $O(n^3)$ gehen. Dabei ~~entstehen~~ entstehen 2n viele Bits.

Multiplikation:

Haben wir x, y zwei Zahlen mit jeweils m und n Bits, dann

$$x \leq 2^m \quad , y \leq 2^n$$

$$x \cdot y \leq 2^m \cdot 2^n = 2^{m+n}$$

Also haben wir jetzt $m+n$ viele Bits! Schlimmstenfalls ist $m=n$ und in jedem Schritt verdoppelt sich die Anzahl der zu betrachtenden Bits.

mit $p(n)$ ~~Operationen~~ Multiplikationen können wir also

$2^{p(n)}$ viele Bits erzeugen.

↝ beim Übergang zur TM haben wir dann

$O(p(n) \cdot 2^{p(n)})$ Schritte.

Das ist nicht mehr Polynomiell.

Deshalb: Logarithmisches Kostenmaß:

Operationen der Ram werden mit den verwendeten Bits gezählt:

Eingabegröße: # Bits d. Eingabe

z.B. Eingabe n Zahlen, die größte ist

$$2^k,$$

dann Eingabegröße

$$O(n \cdot \log 2^k) = \underline{O(n \cdot k) \text{ Bits}}$$

n, k beliebig

eine Add. auf der RAM zählt dann nicht mehr mit $O(1)$ sondern mit $O(\log x)$, wo bei x ~~die~~ ~~in dem~~ größere der beiden ~~Oper~~ Operanden ist, also mit der # der verarbeiteten Bits.

für die Mult. ebenso. & Beachte. Es reicht, zur Erhaltung der Poly-zeit, die # der Bits zu zählen. Der tatsächliche Aufwand einer ~~reen~~ realen RAM wäre auch hier etwa & $O((\log x)^2)$.

Eine Rechnung der Ram, die logarithmisch mit $O(p(n))$ vielen Schritten auskommt, arbeitet auch nur auf $O(p(n))$ vielen Bits.

Da sich jede Einzeloperation der RAM auf einer TM mit polynomiellem Aufwand simulieren lässt, bleibt es bei ~~Polynomiell~~ Polynomialzeit.

Simulation der RAM

- RAM hat Zeit (logarithmisch) $t(n)$
- TM hat für jede Programmzeile ein Unterprogramm
- Eingabe auf Band 1
- Band 2 enthält Speicher der RAM

=> Schrittweise Simulation der RAM

- b → Zustand
- Verknüpfung der Unterprogramme entsprechend
- alle Informationen von Band 2 holen und Rechenschritt simulieren, dann Band 2 aktualisieren geht in Polynomialzeit in der Länge des benutzten Bandes, Bandlänge ist durch $O(t(n))$ wegen log. Kosten der RAM beschränkt also ebenso Zeit $O(q(t(n)))$ für die TM

Mehrband zu Einband geht auch mit polynomiellem Zusatzaufwand, denn etwa

$$\mathcal{O}\left((g(t(n)+n))^2\right)$$