

Alphabet

- Leeres Wort: λ

- Alphabet $\Sigma = \{0, 1, b, a\}$

- Σ^* (=Menge aller Wörter über Alphabet)

- Kontraktion $\text{kon}(x, y) = xy$

- $(abcd)^R = (dcba)$

- Kanoni sche Ordnung

• Ein Wort w steht vor Wort w' , wenn es kürzer ist

• Oder bei gleicher Länge lexikographisch vor w'

Wort

- Zusammenstellung von Buchstaben

- $w = \underbrace{\dots}_\text{prefix} \underbrace{o}_\text{Teilwort} \underbrace{\dots}_\text{suffix} \underbrace{\dots}_T$

- Suffix / prefix sind auch Teilwörter

- Teilwörter: $w = ab \mid \{ab, a, b, \lambda\}$

- Echte Teilwörter / suffix / prefix:

• ohne leeres Wort
• ohne ganzes Wort

Sprache

- $L \subseteq \Sigma^*$

- Für jede Sprache muss Alphabet angegeben werden

- Sprache rekursiv \Leftrightarrow algorithmisch mit Turingmaschine entscheidbar, ob Wort in Sprache

- Sprache regulär \Leftrightarrow mit endlichen Automat lösbar,
ob Wort in Sprache ist

Kolmogorow-Komplexität

- Nur für Alphabet $\Sigma_{\text{bin}} = \{0, 1\}$
- Minimale binäre Länge Pascal-Programm, welches Wert ausgibt
- $K(x) \leq |x| + c$ / Zahlen: $K(n) = K(\text{bin}(n))$ → statischer Teil, der für jedes n gleich bleibt
- Bei Kolmogorov-Komplexität ist Speicherplatz egal, es zählt einfach, wie lange der nicht-statische-Syntax ist
- Beispiel: $w = 0^n 1^n$

- 2 Teile: Pascal Programm → Laufzeitanalyse

- begin


```

k = n
for(i = 1 to k)
    print("0")
end
for(i = 1 to k)
    print("0")
end
end
```

- $K(w) \leq \lceil \log_2(n+1) \rceil + c$

- c sei der statische Teil des Programms
- Je nach Länge ändert sich nur das $k = n$

- Wieso braucht Zahl n , $\log(n+1)$ in binär?

- Die Zahl n ist die $(n+1)$ -te Zahl, da es 0 auch noch gibt

- Beispiel: $w = 0^{n^{(n^5)}}$

- begin


```

k = n
j = k * k * k * k * k
for(int i = 1 to j)
    q = i * k
    for(int i = 1 to q)
        print("0")
    end
end
end
```

- $K(w_n) = \lceil \log_2(n+1) \rceil + c$

- $(w_n) = n^{(n^5)}$ / $\log_w(-)$

$$\Leftrightarrow \log_w((w_n)) = n^5 \quad \sqrt[5]{-}$$

$$\Leftrightarrow n = \sqrt[5]{\log_w((w_n))}$$

- $K((w_n)) = \lceil \log_2(n+1) \rceil + c$

$$= \lceil \log_2(\sqrt[5]{\log_w((w_n))} + 1) \rceil + c$$

Beweis: Für alle n existiert ein Wort x_n der Länge n mit der Eigenschaft $K(x_n) \geq n = |x_n|$

- Ein Wort, das nicht komprimierbar ist, heißt zufällig
 - Es gibt 2^n binäre Strings für Zahlen der Länge n , aber es gibt nur 2^{n-1} Strings, die kleiner sind als $|x_n|$
 - $n \in \mathbb{N}$ ist zufällig: $k(\text{bin}(n)) > \lceil \log_2(n+1) \rceil - 1$

Primzahlsatz

$$-\ln(n) - \frac{3}{2} < \frac{n}{\text{prim}(n)} < \ln(n) - \frac{1}{2} \quad n \geq 67$$

Endlicher Automat

- Ein EA benötigt nur Speicherplatz für das Programm selbst, nicht für Input/Varianten

$$-M = (\underbrace{\Sigma, Q, q_0}_{\text{Alphabet, Zustände, Anfangs- zustand}}, \underbrace{\delta}_{\text{Übergangs- funktion}}, \underbrace{F}_{\text{Akzept.- Zustände}}) \quad (\text{Jes Tupel})$$

\rightarrow 

$$-\delta: Q \times \Sigma \rightarrow Q$$

- Berechnungsschritt

- Wir gehen einen Schritt weiter

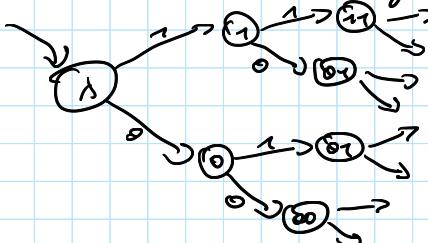
$$\cdot (q, ax) \xrightarrow{m} (q', x) \xrightarrow{m} (m) \xrightarrow{m} (m)$$

$$\cdot \underbrace{(q, \alpha x)}_{\text{Startkonzentration}} \xrightarrow{\frac{1}{m}} \underbrace{(q', \lambda)}_{\text{Endkonzentration}} \quad (\text{Ganz viele Schritte} \rightarrow \text{Wiederholung})$$

- Transitions & Label k

$$\begin{array}{c|c|c} \overline{0} & \overline{8} & \overline{1} \\ \hline 90 & 9x & 9y \\ \hline 91 & 9w & 9z \end{array}$$

- Speichern von Eingabe



Produktautomat

- Schaut sich mehrere Kriterien an
 - Wir bauen grossen Automat, der beides berücksichtigt,
 - oder wir bauen je Kriterium ein Automat und bilden Schnitt
 - $L_1 \cup L_2$:
-
- $q_{01} \xrightarrow{0} q_{11}, q_{01} \xrightarrow{1} q_{12}$ $q_{02} \xrightarrow{0} q_{21}, q_{02} \xrightarrow{1} q_{22}$
- -----
- $q_{11} \quad \vdots \quad q_{12}$ $q_{21} \quad \vdots \quad q_{22}$

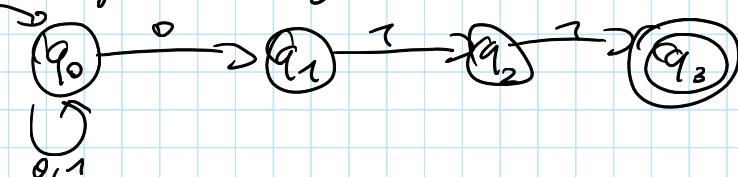
$$- L_1 \setminus L_2 \stackrel{\Delta}{=} L_1 \cap \overline{L_2} \quad \text{Akzeptierende Zustände werden vertauscht}$$

↳ Ich will Kriterium 1 ohne Kriterium 2

$$- \text{Klasse: formale Beschreibung eines Zustandes } KI[q_{\text{accept}}] = L$$

Nichtdeterministischer endlicher Automat

- Deterministisch: Festes Wort, immer gleicher Output
- Unterschied: Ich darf Menge von Zuständen haben, bei einem Konfigurationschritt
 - Das Verhalten eines Wortes ist nicht immer gleich
 - Algorithmus kann stecken bleiben, wenn kein Pfeil gibt
- Ein Wort ist in der Sprache, wenn es mindestens eine Möglichkeit gibt, in einem akzeptierenden Zustand zu landen



$$- \underline{\text{NEA }} M = (\Sigma, Q, \delta, q_0, F)$$

$$\cdot \delta: Q \times \Sigma \rightarrow \{q_x, q_y, -\}$$

$$\cdot \delta(q_x, w) \rightarrow \{q_y\} \quad (\text{NEA bleibt stehen} \rightarrow \text{nicht akzeptiert})$$

$$\cdot \text{Potenzmengenkonstruktion: Konvertiere NEA} \rightarrow \text{EA } \langle \{q_0, q_1, \dots\} \rangle$$

Nichtregularitätsbeweis

- immer am Anfang: „Wir nehmen an, Sprache L sei regulär.“

1. Lemma 3-3. / Pigeon-Hole

1. Da wir nur $|Q|$ verschiedene Zustände haben, und wir mehr als $|Q|$ Wörter konstruieren können $\sigma^0, \sigma^1, \dots, \sigma^{|Q|}, \sigma^{|Q+1|}$, muss für mind. zwei Wörter gelten: $\underbrace{\delta(q_0, \sigma^i)}_{\text{Länden im gleichen Zustand}} = \delta(q_0, \sigma^j)$
2. Da wir im gleichen Zustand landen, weiß der FA nicht mehr, welches Prefix wir gehabt haben
3. Wir wählen $z = 1^i$, und es muss gelten, dass entweder beide Wörter in der Sprache sind oder nicht
4. Das führt aber zu einem Widerspruch, da gilt:

- $0^i 1^i \in L$
- $0^i 1^i \notin L \quad \nrightarrow \text{Nicht regulär}$

2. Pumping Lemma

- $w = yxz$

(I) $|y|x| \leq n_0$

(II) $|x| \geq 1$

(III) yx^kz entweder $\in L$ oder $yx^kz \notin L$

- Bei diesem Beweis wählen wir nur das Wort, die Zerlegung muss immer einen Widerspruch ergeben

1. $w = 0^{n_0} 1^{n_0}$

(I). $xy = 0^m \quad m < n_0$

(II). $x = 0^l \quad l \geq 1$

(III). $yx^kz = 0^{n_0} 1^{n_0} \in L$

$yx^kz = 0^{n_0} 0^l 1^{n_0} \in L \quad \nrightarrow \text{Nicht regulär}$

$$w = \underbrace{0^{m-l}}_y \underbrace{0^l}_x \underbrace{0^{n_0-m}}_z 1^{n_0}$$

3. Kolmogorov-Lemma

- Präfixsprache hat keine Kolmogorov-Bedingung

- Suffix muss konstant sein

1. Wir wählen als Präfixsprache $L_{D^n} = \{y \in \{0,1\}^* \mid xy \in L\}$

2. Nun wählen wir y so, dass ein Wort entsteht, das an erster, zweiter,... Stelle in L_{D^n} steht. der Einfachheit halber nehmen wir das erste Wort

3. $y = 1^n$ ist das erste Wort in L_{D^n}

4. Für y gilt: $K(y) \leq \lceil \log_2(n+1) \rceil + c$

$$= \lceil \log_2(1+n) \rceil + c$$

$$= c'$$

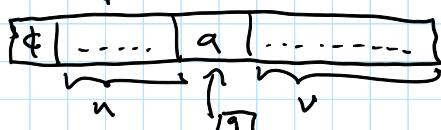
5. Wir können nun nur $2^{c'}$ verschiedene Wörter darstellen, aber in der Sprache gibt es unendlich viele Wörter, dies ist ein Widerspruch

Turing-Maschine

- Unterschied zum endlichen Automat: TM hat Speicher

- TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$

- Beschreibung: $\# u q_0 v \in \Sigma^* \cdot \Gamma^* \cdot Q \cdot \Gamma^*$



- Übergangsfunktion: $\delta(q_0, ax) \xrightarrow{\quad R \quad} (q_1, x, \underset{L}{v})$

- Akzeptierende Berechnung: Wenn ich in q_{accept} lande, ist TM fertig

- Unendliche Berechnung: Ich erreiche $q_{\text{acc}}/q_{\text{ref}}$ nie

- TM hält auf Wort x , wenn $q_{\text{acc}}/q_{\text{ref}}$ erreicht wird

- TM hält immer, wenn für alle Wörter $q_{\text{acc}}/q_{\text{ref}}$ erreicht wird

Mehrbandturingmaschinen

- Entspricht von-Neumann-Modell

$$-\delta: Q \times \Sigma \times \Gamma^k \rightarrow \underbrace{Q \times \Sigma_{L,R,N}^k}_{\text{Eingabeband}} \times \underbrace{(\Gamma \times \Sigma_{L,R,N})^k}_{\text{Arbeitsbänder}}$$

- Turingmaschine: 1 Band

- 1-Band Turingmaschine = 2 Bänder (Eingabe- + Arbeitsband)

Nichtdeterministische Turingmaschine

$$-\delta: Q \times \Gamma \rightarrow \text{Pot}(Q \times \Gamma \times \Sigma_{L,R,N})$$

- Um zu schauen, ob NTM Eingabe akzeptiert, geht nur Breitensuche, da Tiefensuche bei unendlich langen Berechnungsschritten nie aufhören würde

Berechenbarkeit

- $|A| \leq |B|$, wenn es injektive Funktion $f(a) = b$ gibt

- $|A| = |B|$, wenn es bijektive Funktion $f(a) = b$ gibt

↳ Beispiel: $|\mathbb{N}| = |\text{Ingeradell}$, da es bijektive Funktion $f(i) = 2i$ gibt

- Menge A heißt abzählbar unendlich, falls $|A| = |\mathbb{N}|$

↳ Man kann Elemente der Menge nummerieren, z.B.
mit kanonischer Ordnung

- $\mathbb{N} \times \mathbb{N}$ ist abzählbar unendlich

	1	2	3	4	5
1	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)
2	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)
3	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)

- \mathbb{Z}^k ist abzählbar, wegen der kanonischen Ordnung

- \mathbb{Q}^+ ist abzählbar, da jede Zahl durch $\frac{p}{q}$ dargestellt werden kann

- $[0,1]$ ist nicht abzählbar, wegen der Diagonalisierungsmethode:

$f(x)$	$x \in [0,1]$
1	0, a ₁₁ a ₁₂ a ₁₃ a ₁₄ ...
2	0, a ₂₁ a ₂₂ a ₂₃ a ₂₄ ...
3	0, a ₃₁ a ₃₂ a ₃₃ a ₃₄ ...
...	...

- Mindestens eine Zahl kommt nicht vor

- Wir konstruieren diese: $c = 0, \overline{a_{11} a_{22} a_{33} \dots}$
↳ kommt noch nicht vor!

- Beweis: Es gibt nicht-rekursive Sprachen
 - Intuitiv: Es gibt unendlich Sprachen, aber nur endlich viele Turingmaschinen

• $| \text{Ked TM} | < | \text{Potenzmenge}((\Sigma_{\text{Bool}})^*)|$

Menge binär codierbaren Alle Sprachen
Turingmaschinen

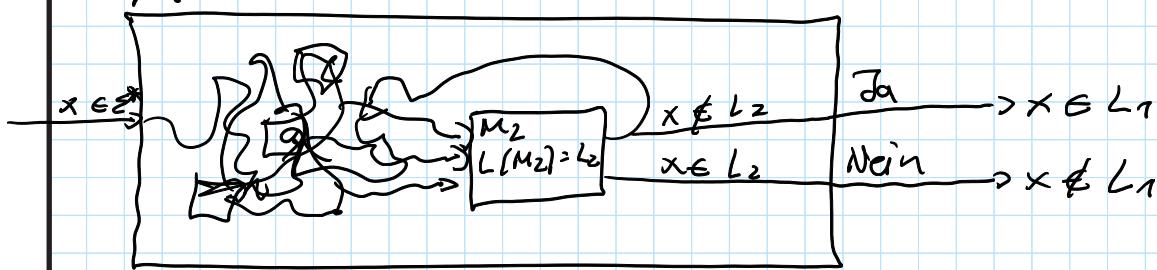
	w_1	w_2	w_3	\dots	Kanonische Ordnung aller Wörter über Σ_{Bool}^*
M_1	1	0	0	\dots	1: M_x akzeptiert w_y
M_2	0	1	1	\dots	
M_3	1	0	0	\dots	
\vdots	\vdots	\vdots	\vdots	\vdots	
All TM					

- Wir konstruieren $L_{\text{diag}} = 001\dots$, und $L_{\text{diag}} \notin L_{\text{rekursiv}}$, da es keine TM M_i gibt, welche die Sprache erkennt.

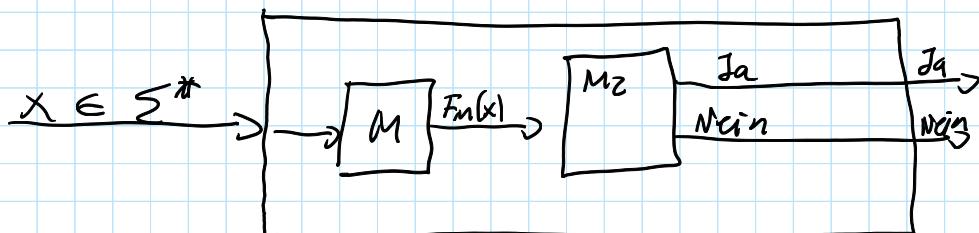
Reduktion

- Wir nehmen an, wir haben eine Maschine, die L_2 löst, und wir nutzen die Reduktion, um L_1 zu lösen
- „ L_1 ist nur höchstens so schwer wie L_2 “
- LRE : Es gibt TM, die Sprache erkennen kann, aber unendlich lange laufen darf
- L_R : Es gibt TM, die Sprache erkennen kann, und auf jedem Wort w hält, also nur endlich läuft
- $L_1 \leq_R L_2$: • L_1 ist rekursiv reduzierbar auf L_2

- Wenn $L_2 \in \text{L}_R$ ist, ist auch $L_1 \in \text{L}_R$



- $L_1 \leq_{EE} L_2$:
 - Ich darf nur Eingabe von L_1 zur Sprache von L_2 konvertieren, mehr nicht
 - Wenn $\underbrace{f_M(x)}_{\text{in } L_2 \text{ konvertiertes Wort}} \in L_2$, dann $x \in L_1$



- Jede EE-Reduktion ist auch eine R-Reduktion
- $\leq_{EE} \Rightarrow \leq_R$

Sprachen für Reduktionen

- $L_{\text{Diag}} = \{ w \in \Sigma^* \mid w = w_i \wedge M_i \text{ akzeptiert } w_i \text{ nicht} \}$
 - ↪ $L_{\text{Diag}} \notin L_{\text{RE}}$ wegen Diagonalisierungsbeweis
- $L_u = \{ \text{Kod}(M) \# w \mid M \text{ akzeptiert } w \}$
 - ↪ $L_u \in L_{\text{RE}}$ (läuft unendlich, wenn w auf M nicht hält)
- $L_h = \{ \text{Kod}(M) \# w \mid M \text{ hält auf } w \}$
 - ↪ $L_h \in L_{\text{RE}}$
 - $L_h^c \notin L_{\text{RE}}$, da L_h^c annimmt, wenn man gerade unendlich lange läuft, aber wenn eine Maschine unendlich lange läuft, sagt man, verweigert man das Wort

Theoretische Informatik - Endterm

Kapitel 5

- L_{empty} : Besitzt kein akzeptierenden Zustand

- L^c_{empty} :
 • Entweder keine korrekte Kodierung
 • Oder akzeptiert mind. ein Input

Satz von Rice

• Jedes nichttriviale semantische Entscheidungsproblem über eine Turingmaschine ist nicht-entscheidbar

• $L \notin \mathcal{L}_R$

• Die Sprache $L_{U,\lambda}$ ist ein semantisch nichttriviales Entscheidungsproblem, denn sie ist offenbar nicht leer, enthält nicht die Kodierungen aller Turingmaschinen und für zwei beliebige Turingmaschinen A und B , die die gleiche Sprache akzeptieren, gilt $\text{Kod}(A) \in L_{U,\lambda} \iff \text{Kod}(B) \in L_{U,\lambda}$. Nach dem Satz von Rice folgt somit, dass $L_{U,\lambda} \notin \mathcal{L}_R$.

Reduktionen

• $L_{H,\lambda}$: OS Maschine M auf Input λ haltest

• Input für Maschine ignorieren

• Reject Zustände zu Accept umwandeln $\rightarrow L_H$

• i-ter Zustand ist der akzeptierende Zustand

• i+1-te Zustand ist verwesender Zustand

• Wenn man zeigen will, dass Sprache L' nicht rekursiv ist, macht man Reduktion z.B. mit

$L_{H,\lambda}$, da diese nicht-rekursiv ist: $L_{H,\lambda} \leq_R L' \Rightarrow L' \notin \mathcal{L}_R$

~ Falls $L \in \mathcal{L}_{RE}$ und $L^c \in \mathcal{L}_{RE} \Rightarrow L \in \mathcal{L}_R$

\hookrightarrow Schriftweise von L / L^c ein Schild ausdrucken

(z.G. auf, um zu zeigen, dass $L^c \notin \mathcal{L}_{RE}$)

$\cdot L \in \mathcal{L}_{RE}$ 
 zu zeigen: $L \notin \mathcal{L}_R$

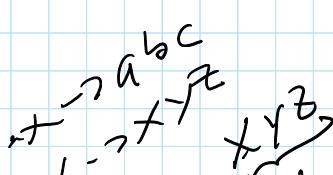
Kapitel 10

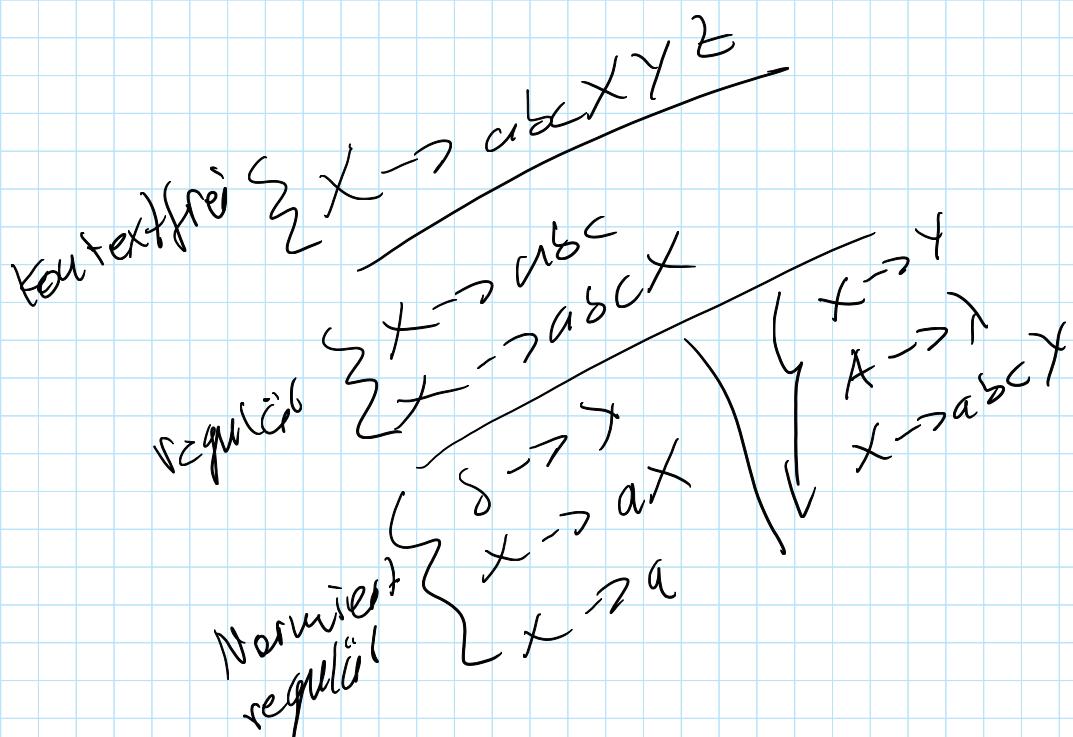
Kapitel 10

- Grammatik $G = \{ \Sigma_T, \Sigma_N, P, S \}$ → Ableitfugen
 abstrakt grammatisches Modell

- Typen der Grammatiken

- L_0 : Keine Anforderung
 - L_1 : Kontextsensitive-Sprache
 - Wörter dürfen nur längen werden
 - L_2 : Kontextfreie-Sprache
 - Kellerautomat
 - Es darf links nur ein Nichtterminal stehen
$$\hookrightarrow X \rightarrow abXy$$
 - L_3 : Reguläre-Sprache
 - Endlicher-Automat
 - $X \rightarrow abcX$
 - $X \rightarrow abc$


 (Kleene Star): X^*
 (Hilfszustand): ϵ



Niemals reguläre Sprache

1. $S \rightarrow \lambda$ (= Generierung leeres Wort)

2. $A \rightarrow a$

3. $B \rightarrow bC$

• Unerlaubte Regeln: 1. Kettenregel: $A \rightarrow C$

2. $A \rightarrow \lambda$ ($A \neq S$)

3. $B \rightarrow abcdC$

1. Kettenregel eliminieren

• $X \xrightarrow{f} Y / Y \rightarrow a / Y \xrightarrow{f} Z / Z \rightarrow b$

• $(X \rightarrow Y) : \text{Kett}_0(X) = \{X\}$

$\text{Kett}_1(X) = \{X, Y\}$

$\text{Kett}_2(X) = \{X, Y, Z\}$

$$\left. \begin{array}{l} \\ \\ \end{array} \right\} |\Sigma_n| - 1 = 2$$

• $(Y \rightarrow Z) : \text{Kett}_0(Y) = \{Y\}$

$\text{Kett}_1(Y) = \{Y, Z\}$

$\text{Kett}_2(Y) = \{Y, Z\}$

• $Y \rightarrow a / Z \rightarrow b / \underbrace{X \rightarrow a / X \rightarrow b / Y \rightarrow b}_{\text{Neu}}$

2. $A \rightarrow \lambda$ eliminieren

• $S \rightarrow X / X \rightarrow abY / X \rightarrow cdY / Y \xrightarrow{f} \lambda$

• $S \rightarrow X / X \rightarrow abY / X \rightarrow cdY / \overbrace{X \rightarrow ab / X \rightarrow cd}$

3. $X \rightarrow abcY$ eliminieren

• $X \rightarrow abcY$

• $X \rightarrow aH_1$

$H_1 \rightarrow bH_2$

$H_2 \rightarrow cY$

- Chomsky-Normalform: $\cdot A \rightarrow BC$

$\cdot A \rightarrow a$

- Greibach-Normalform: $A \rightarrow aXYZ$

- Pumping-Lemma für kontextfreie-Sprachen

1. Angenommen L sei kontextfrei

2. Dann existiert eine nur von L abhängige Konstante n_L

3. Dass für alle Wörter $z \in L$: $|z| \geq n_L$

4. Es gibt eine Zerlegung $z = uvwx$ und dabei gilt

(i) $|vx| \geq 1$

(ii) $|vwx| \leq n_L$

(iii) $\{uv^iwx^iy \mid i \in \mathbb{N}\} \subseteq L$

- Beispiel: $L = \{a^n b^{n^3} \mid n \geq 1\}$ | $\overbrace{\quad + \quad}^{\text{man darf nur rauspumpen}} = \overbrace{\quad}^{\text{num(v)} \neq \text{num(w)} \neq \text{num(x)}}$
 $\cdot z = a^{n_L} b^{n_L^{n_L}}$ | $1^{n_L} 0^{n_L} \neq 0 \neq 1^{n_L} 0^{n_L}$

- Wir unterscheiden nun mehrere Fälle:

1: $vx = a^i$: b wird dann nicht vergrößert

2: $vx = b^i$: a wird dann nicht vergrößert

3: $vx = a^i b^j$: Gibt nicht, dass $i = j^3$

\hookrightarrow In jedem Fall erhalten wir einen Widerspruch

- Nichtdeterministischer Kellerautomat

- Zwischending zwischen Turingmaschine \Leftrightarrow Endlicher Automat

- Kellerautomat ist endlicher Automat mit Keller/Stack

- Also eine Art Turingmaschine mit mehreren qaccept/qreject

- $M = (Q_0, \Sigma, \Gamma, \delta, q_0, Z_0)$

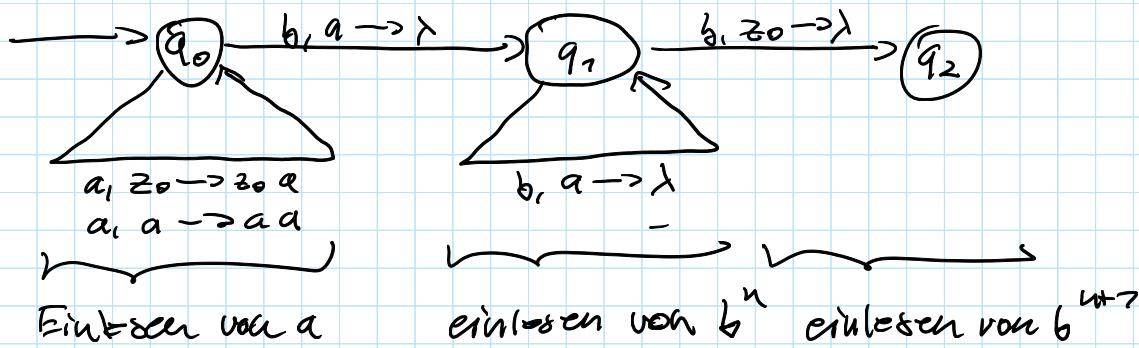
$\xrightarrow{\text{Zustände}}$ $\xrightarrow{\text{Input}}$ $\xrightarrow{\text{Keller}}$ $\xrightarrow{\text{Aufgang}}$

$\cdot \delta: Q_0 \rightarrow Q_0 \times (\Sigma \text{ oder } \lambda) \times \Gamma$

$\xrightarrow{\text{Alter Zustand}}$ $\xrightarrow{\text{neuer Zustand}}$ $\xrightarrow{\text{Input nehmen, oder nicht}}$ $\xrightarrow{\text{Keller modifizieren}}$

- Akzeptierung eines Wortes: • Entweder (q) q
• oder wenn Kellerausgang wurde

• Beispiel: $L = \{a^n b^{n+1} \mid n \in \mathbb{N}\} \setminus \{\epsilon\}$



- Umwandlung: Greibach-Normalform \rightarrow Kellerautomat.

$$\cdot M = (\Sigma^*, \Sigma_T, \Sigma_0, \delta, q_0, \delta)$$

$$\cdot \delta(q_0, a, A) = \{(q_0, \alpha^R) \mid A \rightarrow a\alpha \in P\}$$

↳ für alle $a \in \Sigma_T$ und $A \in \Sigma_0$

$$\cdot \text{Beispiel: } S \rightarrow aAB : \delta(q_0, a, S) = \{L(q_0, BA)\}$$

$$\cdot A \rightarrow aAB : \delta(q_0, a, A) = \{L(q_0, BA)\}$$

$$\cdot A \rightarrow aB : \delta(q_0, a, A) = \{L(q_0, B)\}$$

$$\cdot B \rightarrow bB : \delta(q_0, b, B) = \{L(q_0, B)\}$$

$$\cdot B \rightarrow \lambda : \delta(q_0, \lambda, B) = \{L(q_0, \lambda)\}$$

Kapitel 6

- Es ist möglich Mehrband-TM \rightarrow Einband-TM zu konvertieren

- P = {L(B) | B eine MTM mit $\text{Time}_B(n) \in O(n^c)$ für konstantes c}

- NP = {L(B) | B eine NMTM mit $\text{Time}_B(n) \in O(n^c)$ für konstantes c}

- VP: Wenn ich mithilfe eines Polynomzeitverifizierers } Prüfbar
und einem Beweis (z.B. Belegung SAT) in
polynomialer Zeit feststellen kann, ob Eingabe }
in Sprache ist

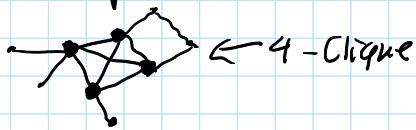
- $NP = VP$: Es ist gleich schwer Beweis für NP zu testen, oder NT wel?
- $NP \subseteq VP$: - Ich habe Wort x welches in der Sprache ist
 - Dann schau ich mir einfach den Entscheidungsbaukasten an, welche Zweige x nimmt, um in $accept$ zu landen.
 - Dies wähle ich als Zertifikat $c = 101111\ldots$
- $NP \supseteq VP$: - Ich nehme das Wort x und wähle nichtdeterministisch das Zertifikat c

SAT

- x kodiert eine Formel in KNF und ist erfüllbar
- Konjunktive Normalform: $(\underbrace{x \vee y \vee z}_{\text{LITERAL}}) \wedge \underbrace{(A \vee B \vee C)}_{\substack{\text{und} \\ \text{Negation}}} \wedge \dots$ oder
 $\underbrace{\neg A}_{\text{Klausel}}$

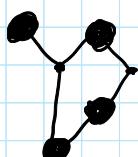
Clique

- Graph G ist ein Graph der eine k -clique enthält
- Vollständiger Teilgraph, wo jeder Knoten mindestens k -Kanten besitzt



Vertex-Cover

- Ein Vertex-Cover mit höchstens k -Knoten, welche alle Kanten im Graph "berühren"



- Vierfach-SAT: KNF-Formel, welche vier unterschiedliche Belegungen hat, sodass Formel wahr ist
- 3SAT: KNF-Formel, welche in jeder Klausel genau 3 Literale hat

Polynomzeitreduktion

- L_1 ist polynomiell reduzierbar auf L_2 : $L_1 \leq_p L_2$, falls ein Algorithmus existiert, der die Eingabe x zur Eingabe $A(x)$ für L_2 in polynomialer Zeit umwandelt und danach gilt $x \in L_1 \Leftrightarrow A(x) \in L_2$
- Ähnlich wie Ende-zu-Ende-Reduktion, einfach mit Polynomzeitkriterium
- NP-schwer: Wenn L NP-schwer ist, kann ich alle anderen Probleme darauf reduzieren
 $L' \leq_p L$
- NP-vollständig: Wenn L NP-schwer ist, und L in NP liegt und ein Entscheidungsproblem darstellt (z.B. SAT)
- Um zu zeigen, dass ein NP-schweres Problem NP-vollständig ist, kann ich sagen, dass $VP = NP$ gilt, und einen Verifizierer angeben
- Man zeigt nur, wie Umwandlungsfunktion $A(x)$ aufgebaunt ist