Formale Beschreibung einer Turingmaschine

Eine **Turingmaschine** ist ein Siebentupel

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

- Q endliche Menge von Zuständen,
- Σ endliche Menge der Eingabesymbole,
- Γ endliche Menge der Bandsymbole, $\Sigma \subset \Gamma$,
- $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R, S\}$ Übergangsfunktion,
- $q_0 \in Q$ Startzustand,
- $B \in \Gamma - \Sigma$ Blank-Symbol,
- $F \subseteq Q$ Menge von Endzuständen.

Übergänge

$$(q, X; p, Y, D) \in \delta$$

M liest im Zustand q auf dem Band Symbol X

- wechselt nun davon abhängig in den Zustand p,
- überschreibt Symbol X durch Symbol Y,
- bewegt den Lese-/Schreibkopf in die durch D beschriebene Richtung ($D \in \{L, R, S\}$).

Deterministische Turingmaschine (DTM)

Eine Turingmaschine M heißt **deterministisch**, wenn für alle $(q, X) \in Q \times \Sigma \times \Gamma$ höchstens ein Element $(q, X; p, Y, D) \in \delta$ existiert, $D \in \{L, R, S\}$; wir schreiben dann auch $\delta(q, X) = (p, Y, D)$.

(Deterministische) Turingmaschinen: akzeptierte Sprache

Akzeptierte Sprache

Die von der (deterministischen) Turingmaschine M **akzeptierte Sprache** $L(M)$ besteht aus genau all jenen Wörtern, bei deren Anaylse M einen Endzustand erreicht:

$$L(M) = \{w \in \Sigma^* \mid q_0 w \Rightarrow^* \alpha p \beta, \quad p \in F, \quad \alpha, \beta \in \Gamma^*\}$$

Rekursiv aufzählbare Sprachen

Eine formale Sprache heißt **rekursiv aufzählbar**[2] wenn sie von einer (deterministischen) Turingmaschine akzeptiert wird.

[2] engl: recursively enumerable, auch bezeichnet als: Turing-erkennbar (Turing-recognizable), semi-entscheidbar (semi-decidable)

Beispiel

TM für $\{0^n 1^n \mid n \geq 1\}$

$M = (\{q_i \mid 0 \leq i \leq 4\}, \{0, 1\}, \{0, 1, X, Y, B\}, \delta, q_0, B, \{q_4\})$
wobei δ gegeben durch:

	Symbol				
Zustand	0	1	X	Y	B
q_0	(q_1, X, R)			(q_3, Y, R)	
q_1	$(q_1, 0, R)$	(q_2, Y, L)		(q_1, Y, R)	
q_2	$(q_2, 0, L)$		(q_0, X, R)	(q_2, Y, L)	
q_3				(q_3, Y, R)	(q_4, B, R)
q_4					

Beispiel ctd.

akzeptierende Berechnung von 0011

Zustand	Symbol 0	1	X	Y	B
q_0	(q_1, X, R)			(q_3, Y, R)	
q_1	$(q_1, 0, R)$	(q_2, Y, L)		(q_1, Y, R)	
q_2	$(q_2, 0, L)$		(q_0, X, R)	(q_2, Y, L)	
q_3				(q_3, Y, R)	(q_4, B, R)
q_4					

$$
\begin{array}{llllllll}
q_0 0011 & \Rightarrow & Xq_1 011 & \Rightarrow & X0q_1 11 & \Rightarrow & Xq_2 0Y1 & \Rightarrow \\
q_2 X0Y1 & \Rightarrow & Xq_0 0Y1 & \Rightarrow & XXq_1 Y1 & \Rightarrow & XXYq_1 1 & \Rightarrow \\
XXq_2 YY & \Rightarrow & Xq_2 XYY & \Rightarrow & XXq_0 YY & \Rightarrow & XXYq_3 Y & \Rightarrow \\
XXYYq_3 B & \Rightarrow & XXYYBq_4 B & & & & &
\end{array}
$$

Beispiel ctd.

abweisende Berechnung von 0010

Zustand	Symbol 0	1	X	Y	B
q_0	(q_1, X, R)			(q_3, Y, R)	
q_1	$(q_1, 0, R)$	(q_2, Y, L)		(q_1, Y, R)	
q_2	$(q_2, 0, L)$		(q_0, X, R)	(q_2, Y, L)	
q_3				(q_3, Y, R)	(q_4, B, R)
q_4					

$$
\begin{array}{llllllll}
q_0 0010 & \Rightarrow & Xq_1 010 & \Rightarrow & X0q_1 10 & \Rightarrow & Xq_2 0Y0 & \Rightarrow \\
q_2 X0Y0 & \Rightarrow & Xq_0 0Y0 & \Rightarrow & XXq_1 Y0 & \Rightarrow & XXYq_1 0 & \Rightarrow \\
XXY0q_1 B & & & & & & &
\end{array}
$$

Entscheidbare (Rekursive) Sprachen

Wird eine Turingmaschine M auf einem Input w gestartet, so gibt es folgende Möglichkeiten:

- M hält in einem Zustand $q \in F$ und akzeptiert somit die Eingabe
- M hält in einem Zustand $q \notin F$ und verwirft somit die Eingabe
- M gerät in eine Endlosschleife und hält nicht

Rekursive (Entscheidbare) Sprachen

Eine formale Sprache heißt **rekursiv** oder **entscheidbar**[3] wenn sie von einer (deterministischen) Turingmaschine akzeptiert wird, die **immer hält**. (Man sagt dann auch: die TM **entscheidet** die Sprache)

[3] engl: recursive, decidable, auch bezeichnet als: Turing-entscheidbar (Turing-decidable)

(Nicht)deterministische Turingmaschinen

Turingmaschinen können auch Funktionen (auf nicht-negativen ganzen Zahlen) berechnen, bzw. Sprachen erzeugen:

Berechnung von Funktionen

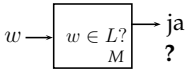
Eine (*deterministische*) Turingmaschine M kann auch zur **Berechnung von Funktionen** verwendet werden, d.h., M beginnt mit dem Input auf dem Band, der Output, also der Funktionswert zum Input, ist das Ergebnis der Berechnung, wenn M hält.

Erzeugung von Sprachen

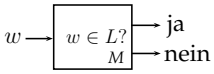
Eine (*nichtdeterministische*) Turingmaschine M kann auch zur **Erzeugung von Wörtern** verwendet werden, d.h., M beginnt mit dem leeren Band, das Ergebnis der Berechnung ist das Wort, das am Ende einer Berechnung, wenn M hält, auf dem Band steht.

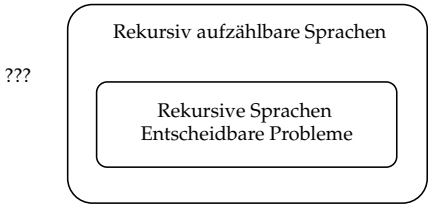
Problemlandschaft

Rekursiv aufzählbare Sprachen L:
Es gibt eine TM M die akzeptiert und hält wenn $w \in L$, sonst aber u.U. unendlich läuft.



Rekursive Sprachen (**Entscheidbare Probleme**) L:
Es gibt eine TM M mit $L(M) = L$ die immer hält. (d.h., es gibt einen **Algorithmus**, der bei Eingabe von w "ja" antwortet, wenn $w \in L$ und "nein" antwortet, wenn $w \notin L$)



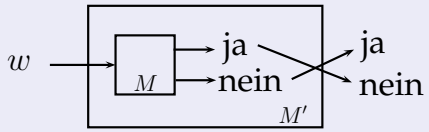


Rekursive (entscheidbare) Sprachen und ihr Komplement

Das Komplement einer rekursiven Sprache ist rekursiv.

Beweis

Sei L rekursiv und M eine TM mit $L(M) = L$, die immer hält. Wir konstruieren M' aus M so, dass M' stoppt ohne zu akzeptieren, wenn M auf einer Eingabe w in einen Endzustand übergeht. Hält M ohne zu akzeptieren, so geht M' in einen Endzustand über. Da eines dieser beiden Ereignisse eintritt, ist $\overline{L} = L(M')$ (das Komplement von L) eine rekursive Sprache.



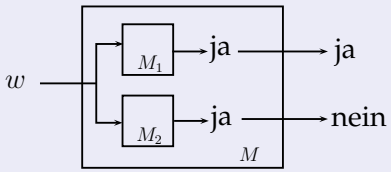
□

Rekursive (entscheidbare) Sprachen und ihr Komplement

Ist eine Sprache L und auch ihr Komplement $\overline{L}$ rekursiv aufzählbar, so ist L (wie auch $\overline{L}$) rekursiv.

Beweis

M_1 und M_2 akzeptieren L bzw. $\overline{L}$. Daraus konstruieren wir M so, dass sie w akzeptiert, wenn w von M_1 akzeptiert wird, und verwirft, wenn w von M_2 akzeptiert wird. Da w entweder in L oder in $\overline{L}$ ist, wird genau eine der beiden Turingmaschinen M_1, M_2 akzeptieren. Also wird M immer entweder "ja" oder "nein" ausgeben, aber nie beides. Somit ist L rekursiv.



□

Codierung von Turingmaschinen

Turingmaschinen können als (binäre) Strings codiert werden, beispielsweise so:

Sei $M = (\{q_1, q_2, ..., q_r\}, \{0, 1\}, \{0, 1, B, X_4, ..., X_s\}, \delta, q_1, B, \{q_2\})$

Codierung von $\delta(q_i, X_j) = (q_k, X_l, D_m)$:

$$0^i 1 0^j 1 0^k 1 0^l 1 0^m$$

Codierung von M:

$$C_1 11 C_2 11 ... 11 C_{n-1} 11 C_n$$

wobei C_i der Code einer Transition ist.

Nicht rekursiv aufzählbare Sprachen

Es gibt Sprachen über $\Sigma = \{0, 1\}$, die nicht rekursiv aufzählbar sind.

Beweis

- Zu jeder rekursiv aufzählbaren Sprache L über $\Sigma = \{0, 1\}$ (d.h. $L \in \mathcal{P}(\Sigma^*)$) gibt es eine Turingmaschine, die sie akzeptiert.
- Jede Turingmaschine M kann als String über $\Sigma = \{0, 1\}$ (d.h. $M \in \Sigma^*$) dargestellt werden.
- Die Menge aller Sprachen $\mathcal{P}(\Sigma^*)$ ist überabzählbar (unendlich), die Menge aller Wörter Σ^* hingegen abzählbar (unendlich).
- Es gibt also überabzählbar viele Sprachen, von denen jedoch nur abzählbar viele von einer Turingmaschine akzeptiert werden.

□

Das Halteproblem ist rekursiv aufzählbar

$L_u = \{(M, w) \mid M$ ist eine TM die w akzeptiert $\}$

L_u ist **rekursiv aufzählbar**, aber **nicht entscheidbar**.

Beweis Teil 1: L_u ist rekursiv aufzählbar

Wir konstruieren eine universelle Turingmaschine U wie folgt:
Mit Eingabe (M, w) simuliert U die TM M auf w. Wenn M die Eingabe akzeptiert, so akzeptiert auch U, akzeptiert M nicht, so akzeptiert auch U nicht:

$$U \text{ akzeptiert } (M, w) \iff M \text{ akzeptiert } w \iff (M, w) \in L_u$$

Folglich akzeptiert U die Sprache L_u. □

Das Halteproblem ist unentscheidbar

Beweis Teil 2: L_u ist nicht entscheidbar

Beweis indirekt. Angenommen, es gibt eine Maschine H, die L_u *entscheidet*:

$$H((M, w)) = \begin{cases} \textbf{ja} & \text{wenn } w \text{ von } M \text{ akzeptiert wird} \\ \textbf{nein} & \text{wenn } w \text{ von } M \text{ nicht akzeptiert wird} \end{cases}$$

Dann gibt es aber auch eine Maschine D, die genau das Gegenteil von H macht.
D simuliert zunächst H mit der Eingabe $(M, (M))$ und gibt dann **ja** aus, wenn M nicht akzeptiert, bzw. **nein**, wenn M akzeptiert:

$$D((M)) = \begin{cases} \textbf{ja} & \text{wenn } (M) \text{ von } M \text{ nicht akzeptiert wird} \\ \textbf{nein} & \text{wenn } (M) \text{ von } M \text{ akzeptiert wird} \end{cases}$$

Offensichtlich gilt: wenn H existiert, so existiert auch D.

Das Halteproblem ist unentscheidbar - ctd.

Setzt man nun D auf ihren eigenen Code an, so ergibt sich folgendes Verhalten:

$$D((D)) = \begin{cases} \textbf{ja} & \text{wenn } (D) \text{ von } D \text{ nicht akzeptiert wird} \\ \textbf{nein} & \text{wenn } (D) \text{ von } D \text{ akzeptiert wird} \end{cases}$$

Was auch immer D macht, sie muss genau das Gegenteil machen.
Das ist offensichtlich ein Widerspruch! Es kann also weder D noch H geben, demnach ist L_u nicht entscheidbar. □

Diagonalargument

- H akzeptiert $(M, w) \iff M$ akzeptiert w
- D weist (M) zurück $\iff M$ akzeptiert (M)
- D weist (D) zurück $\iff D$ akzeptiert (D)

	(M_1)	(M_2)	(M_3)	$\cdots$	(D)	$\cdots$
M_1	**ja**	*nein*	*ja*	$\cdots$	*ja*	$\cdots$
M_2	*ja*	**ja**	*nein*	$\cdots$	*ja*	$\cdots$
M_3	*nein*	*nein*	**nein**	$\cdots$	*nein*	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$		
D	**nein**	**nein**	**ja**	$\cdots$	**?**	
$\vdots$	$\vdots$	$\vdots$	$\vdots$			$\ddots$

Ein nicht rekursiv aufzählbares Problem

$\overline{L_u} = \{(M, w) \mid M$ ist eine TM die w nicht akzeptiert $\}$

$\overline{L_u}$ ist **nicht rekursiv aufzählbar**.

Beweis

Angenommen, $\overline{L_u}$ ist rekursiv aufzählbar. Da L_u rekursiv aufzählbar ist, müssten demnach beide entscheidbar sein, was aber nicht der Fall ist!
$\overline{L_u}$ kann nicht rekursiv aufzählbar sein. □

Berechenbarkeit

Eine Funktion $f : \Sigma^* \longrightarrow \Sigma^*$ heißt **partiell berechenbar**, wenn es eine Turingmaschine M_f gibt, für die mit $x \in \Sigma^*$ gilt:

- Ist $f(x)$ definiert, so hält M_f, gestartet mit x, und $f(x)$ steht dann auf dem Band.
- Ist $f(x)$ nicht definiert, so hält M_f, gestartet mit x, nicht.

Ist f total, d. h., für alle $x \in \Sigma^*$ definiert, und partiell berechenbar, so heißt f (total) **berechenbar**.[5]

[5] engl: computable

Reduktion

Reduktion ist eine Technik, die Unentscheidbarkeit von Problemen durch Rückführung auf andere Probleme nachzuweisen, deren Unentscheidbarkeit bereits bekannt ist.

Reduktion

Eine Sprache $A \subseteq \Sigma^*$ heißt **reduzierbar** auf die Sprache $B \subseteq \Gamma^*$, (bezeichnet mit $A \leq B$), wenn es eine (total) berechenbare Funktion $f : \Sigma^* \longrightarrow \Gamma^*$ so gibt, dass

$$\text{für alle } w \in \Sigma^*\text{: } w \in A \text{ genau dann, wenn } f(w) \in B.$$

In diesem Fall nennt man f eine **Reduktion** von A auf B.

(Anmerkung: die Relation $\leq$ ist transitiv und reflexiv)

Reduktionen und (Nicht-)Entscheidbarkeit

Seien A und B Sprachen mit $A \leq B$. Dann gilt:

- Ist A nicht entscheidbar, so ist auch B nicht entscheidbar.
- Ist A nicht rekursiv aufzählbar, so ist auch B nicht rekursiv aufzählbar.
- Ist B entscheidbar, so ist auch A entscheidbar.
- Ist B rekursiv aufzählbar, so ist auch A rekursiv aufzählbar.

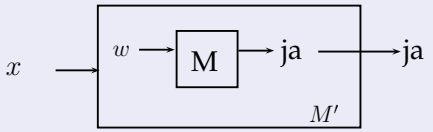
Weitere Unentscheidbare Probleme

$L_{ne} = \{M \mid L(M) \neq \{\} \quad \}$ ist nicht entscheidbar.

Beweis

(Indirekt, durch Reduktion von L_u auf L_{ne} ($L_u \leq L_{ne}$))
Angenommen, es gibt eine Turingmaschine M, sodass $L(M) = L_{ne}$.
Dann konstruieren wir eine Maschine M', die beliebige Eingaben genau dann akzeptiert, wenn M die Eingabe w akzeptiert:

$$w \in L(M) \Leftrightarrow L(M') \neq \{\}$$



Nachdem L_u nicht entscheidbar ist, kann auch L_{ne} nicht entscheidbar sein. □

Eigenschaften von Sprachen

Eine **Eigenschaft** von Sprachen ist eine Teilmenge P von rekursiv aufzählbaren Sprachen (über einem geeigneten Alphabet).

Beispiele für Eigenschaften

$P_1 = \{L \mid L \text{ ist entscheidbar } \}$
$P_2 = \{L \mid L \text{ ist endlich } \}$
$P_3 = \{L \mid L = L^*\}$
$P_4 = \{\{\varepsilon\}\}$
$P_5 = \{\}$
$P_6 = \{L \mid L \text{ ist rekursiv aufzählbar } \}$

$\{a, b\}^*$ hat die Eigenschaften P_1, P_3, P_6, aber nicht P_2, P_4, P_5

Der Satz von Rice[6]

Eine Eigenschaft ist **trivial**, wenn sie entweder leer ist (d.h., sie kommt keiner Sprache zu), oder aus allen rekursiv aufzählbaren Sprachen besteht. Andernfalls ist sie nicht trivial.

Triviale Eigenschaften

$P_5 = \{\}$
$P_6 = \{L \mid L \text{ ist rekursiv aufzählbar } \}$
sind triviale Eigenschaften.

Satz von Rice

Jede nicht triviale Eigenschaft der rekursiv aufzählbaren Sprachen ist unentscheidbar.

[6] Henry Gordon Rice (* 1920)

Anmerkung

Wir können eine Menge von Sprachen nicht als die Sprachen selbst erkennen. Stattdessen müssen wir die Turingmaschinen erkennen, die diese Sprache akzeptieren.

Ist P eine Eigenschaft der rekursiv aufzählbaren Sprachen, dann besteht die Sprache L_P aus der Menge der Codes für Turingmaschinen M_i, sodass $L(M_i)$ eine Sprache aus P ist (d.h. $L_P = \{M_i \mid L(M_i) \in P\}$).

Wenn wir über die Entscheidbarkeit einer Eigenschaft P sprechen, so meinen wir die Entscheidbarkeit der Sprache L_P.

Der Satz von Rice: Anwendungen

Einige Beispiele von Eigenschaften, für die der Satz von Rice unmittelbar die Unentscheidbarkeit impliziert.

- Es ist nicht entscheidbar, ob eine Turingmaschine mehr als sieben Wörter akzeptiert.
  (Die Eigenschaft P ist also definiert durch $P = \{L \mid |L| \geq 7\}$. Um den Satz von Rice anwenden zu können, müssen wir noch überprüfen, ob die Eigenschaft P nicht-trivial ist. Wir müssen also mindestens eine rekursiv aufzählbare Sprache finden, die die Eigenschaft erfüllt, und eine, die die Eigenschaft nicht erfüllt: z.B. $L_1 = \{a, a^2, a^3, \ldots, a^7\} \in P$ und $L_2 = \{\} \notin P$.)
- Es ist nicht entscheidbar, ob eine Turingmaschine nur endlich viele Wörter akzeptiert (d. h. ob die von einer Turingmaschine akzeptierte Sprache endlich ist).
- Es ist nicht entscheidbar, ob die von einer Turingmaschine akzeptierte Sprache regulär ist.

Nicht Anwendbarkeit des Satzes von Rice

Es gibt aber durchaus Eigenschaften für Turingmaschinen, die entscheidbar sind. Für diese Eigenschaften ist dann aber eine der Voraussetzungen des Satzes von Rice nicht erfüllt. (Entweder handelt es sich nicht um eine Eigenschaft der akzeptierten Sprache der Turingmaschine oder die Eigenschaft ist trivial)

- Es ist entscheidbar, ob eine Turingmaschine mindestens 7 Zustände besitzt.
  (Diese Eigenschaft ist keine Eigenschaft der akzeptierten Sprache, sondern eine Eigenschaft der Turingmaschine selbst.)
- Es ist entscheidbar, ob die von einer Turingmaschine akzeptierte Sprache rekursiv aufzählbar ist.
  (Diese Eigenschaft ist trivial, sie ist für jede rekursiv aufzählbare Sprache per Definition erfüllt.)

Das Postsche Korrespondenz Problem[7]

Das Postsche Korrespondenz Problem (PCP)

Gegeben: Alphabet A, $k \in \mathbb{N}$, sowie die Folge von Wortpaaren $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$ mit $x_i, y_i \in A^+$ für $1 \leq i \leq k$.

Frage: Gibt es eine Folge von Indizes $i_1, i_2, \ldots, i_n$ mit $i_j \in \{1, 2, \ldots, k\}$ für $1 \leq j \leq n, n \in \mathbb{N}$ so dass $x_{i_1} x_{i_2} \ldots x_{i_n} = y_{i_1} y_{i_2} \ldots y_{i_n}$ gilt?

(Post 1946)

Das Das Postsche Korrespondenzproblem ist nicht entscheidbar.

Das Das Postsche Korrespondenzproblem ist rekursiv aufzählbar.

[7] Emil Post, amerikanischer Mathematiker, 1897-1954

Das 10. Hilbertsche Problem[8]

10. Hilbertsche Problem

Gegeben: $n \in \mathbb{N}$, ein Polynom $p(x_1, \ldots, x_n)$ in n Unbekannten
Frage: Besitzt p ganzzahlige Nullstellen?

(Matijassewitsch, 1970)

Das 10. Hilbertsche Problem ist nicht entscheidbar.

[8] David Hilbert (1862-1943)

Turingmaschinen - Beispiel

Turingmaschine, die $\{a^n b^n c^n \mid n \geq 1\}$ akzeptiert

$M = (\{p, q, r, s\}, \{a, b, c\}, \{Z_0, A, B, C\}, \delta, p, \{Z_0, Z_1, Z_2\}, B, \{s\})$

wobei

$$\begin{aligned}
1: &\quad \delta(p, a, B) = (p, A, R, R) \\
2: &\quad \delta(p, b, B) = (q, B, S, L) \\
3: &\quad \delta(q, b, A) = (q, C, R, L) \\
4: &\quad \delta(q, c, Z_0) = (r, Z_0, S, R) \\
5: &\quad \delta(r, c, C) = (r, B, R, R) \\
6: &\quad \delta(r, Z_2, B) = (s, B, S, L)
\end{aligned}$$

Turingmaschinen - Beispiel ctd.

Dabei passiert Folgendes:

1 : für jedes eingelesenes Symbol a wird ein A aufs Arbeitsband geschrieben;

2 : wenn das erste Symbol b eingelesen wird, geht M einen Schritt nach links; dieses erste Symbol b wird bei der ersten Anwendung von Übergang 3 nochmals gelesen;

3 : für ein eingelesenes Symbol b wird ein A auf dem Arbeitsband mit C überschrieben;

4 : wird das erste Symbol c gelesen, so muss M genau Z_0 erreicht haben;

5 : für jedes eingelesene Symbol c wird nun ein C auf dem Arbeitsband gelöscht (durch das Blanksymbol B ersetzt);

6 : wird mit dem Ende der Eingabe (d.h., dem Erreichen von Z_2) gleichzeitig das Ende der Symbole C auf dem Arbeitsband erreicht, so hält die Turingmaschine M im Endzustand s, und hat damit die Eingabe akzeptiert.

Normalform

Eine (deterministische) Turingmaschine ist in **Normalform**, wenn

- sie nur einen Endzustand besitzt,
- das Arbeitsband am Ende eines akzeptierenden Laufes der Turingmaschine leer ist und
- der letzte Übergang von der Gestalt $\delta(s, Z_2, Z_0) = (f, Z_0, S, R)$ ist, wobei f dieser einzige Endzustand und s ein beliebiger anderer Zustand ist.

Normalform: Beispiel

M' akzeptiert $\{a^n b^n c^n \mid n \geq 1\}$ in Normalform

$M' = (\{p, q, r, s, f\}, \{a, b, c\}, \{Z_0, A, B, C\}, \delta, p, \{Z_0, Z_1, Z_2\}, B, \{f\})$
wobei

Übergänge 1 bis 6 wie bei M aus vorigem Beispiel.

Damit M' jedoch mit leerem Band akzeptieren kann, benötigen wir zusätzlich noch folgende Übergänge:

$7 : \delta(s, Z_2, B) = (s, B, S, L)$ sowie
$8 : \delta(s, Z_2, Z_0) = (f, Z_0, S, R)$.

TM: eingeschränkte Varianten - LBA

Ein **linear beschränkter Automat** (LBA) ist eine Turingmaschine, die auf dem Arbeitsband nur höchstens soviel Platz verwendet wie das Eingabewort lang ist (der Platz darf sogar eine lineare Funktion der Länge des Eingabewortes sein).

LBA

Eine Turingmaschine M heißt *linear beschränkter Automat*, wenn es eine lineare Funktion $cn + d$ so gibt, dass die Turingmaschine M für jedes Eingabewort w der Länge n während der Analyse höchstens $cn + d$ Felder auf dem Arbeitsband benötigt.

TM: Eingeschränkte Varianten - Kellerautomaten

Turingmaschine erfüllt **Kellerautomatenbedingung**, wenn sie

- auf dem Eingabeband nie nach links gehen kann und
- für den Lese-/Schreibkopf auf dem Arbeitsband in jeder Konfiguration gilt, dass
  - links davon nur Nicht-Blanksymbole und
  - rechts davon nur Blanksymbole stehen können.

Kellerautomaten: Beispiel

Kellerautomat M in Normalform an, der $L = \{0^n 1^{2n+1} \mid n \geq 0\}$ akzeptiert

$$M = \langle \{q_0, q_1, q_2, q_f\}, \{0, 1\}, \{Z_0, A\}, \delta, q_0, \{Z_0 Z_1, Z_2\}, \{q_f\} \rangle,$$

wobei

$$\begin{aligned}
1 &: \quad \delta(q_0, 0, B) = \{(q_1, A, S, R)\} \\
2 &: \quad \delta(q_1, 0, B) = \{(q_0, A, R, R)\} \\
3 &: \quad \delta(q_0, 1, B) = \{(q_2, B, R, L)\} \\
4 &: \quad \delta(q_2, 1, A) = \{(q_2, B, R, L)\} \\
5 &: \quad \delta(q_2, Z_2, Z_0) = \{(q_f, Z_0, S, R)\}
\end{aligned}$$

Kellerautomaten: Beispiel ctd.

Dabei passiert Folgendes:
$1, 2$: für jedes eingelesenes Symbol 0 werden 2 Symbole A in den Keller geschrieben;
3 : das erste Symbol 1 wird überlesen;
4: nun wird für jedes eingelesene Symbol 1 ein Symbol A im Keller gelöscht;
5 : wird gleichzeitig mit dem Ende der Eingabe (dem Erreichen von Z_2) das Kellergrundsymbol Z_0 erreicht, geht der Kellerautomat in den Enzustand q_f über und akzeptiert somit die Eingabe.

Turingmaschinen: Eingeschränkte Varianten - EA

Endlicher Automat: Turingmaschine, die das Arbeitsband gar nicht benötigt. Übergänge besitzen einfache Form $(q, a; p, D_E)$

Erlaubt man nur $D_E \in \{R, S\}$, kann die Übergangsfunktion δ durch Tripel der Gestalt $(q, a; p)$ beschreiben werden ($a \in T \cup \{\varepsilon\}$); dabei ist

$(q, a; p)$ für $a \in T$ als $(q, a; p, R)$ zu interpretieren und
$(q, \varepsilon; p)$ als $(q, a; p, S)$ für ein beliebiges $a \in T$.

Außerdem geht man davon aus, dass ein endlicher Automat seine Analyse auf dem ersten Symbol des Eingabewortes beginnt und in einem Endzustand akzeptiert, wenn der Lesekopf auf dem rechten Begrenzungssymbol steht.

Akzeptierte Sprachen

Eine formale Sprache heißt

0 **rekursiv aufzählbar**, wenn Sie von einer *Turingmaschine* akzeptiert wird
1 **kontextsensitiv**, wenn sie von einem *linear beschränkten Automaten (LBA)* akzeptiert wird
2 **kontextfrei**, wenn sie von einem *Kellerautomaten* akzeptiert wird
3 **regulär**, wenn Sie von einem *endlichen Automaten* akzeptiert wird.

Einschub (WH): Induktive Definition

Gegeben: Grundmenge $A_0 \subseteq B$, Bildungsregel $f : B^n \to B$

Stufenweise Konstruktion von Mengen:

$$A_{i+1} = A_i \cup \{f(x_1, \ldots, x_n) \mid x_1, \ldots, x_n \in A_i\}$$

Die dadurch insgesamt definierte Menge ist dann die Vereinigung all dieser Mengen:

$$\mathcal{A} = \bigcup_{i \geq 0} A_i$$

Abgeschlossenheit

Sei B eine Menge und $f : B^n \to B$ eine Funktion. Eine Menge $A \subseteq B$ heißt *abgeschlossen unter* f, wenn gilt:

$$x_1, \ldots, x_n \in A \quad \Rightarrow \quad f(x_1, \ldots, x_n) \in A$$

Einschub (WH): Induktive Definition ctd.

Sei B eine Menge, $A_0 \subseteq B$ und $f\colon B^n \to B$. Weiters sein $A_{i+1} = A_i \cup \{f(x_1, \ldots, x_n) \mid x_1, \ldots, x_n \in A_i\}$ sowie $\mathcal{A} = \bigcup_{i \geq 0} A_i$. Dann gilt:

- $\mathcal{A}$ ist abgegeschlossen unter f.
- Ist $\mathcal{A}'$ abgeschlossen unter f und gilt $A_0 \subseteq \mathcal{A}' \subseteq B$, dann gilt $\mathcal{A} \subseteq \mathcal{A}'$.

D.h.: $\mathcal{A}$ ist die kleinste Menge, die A_0 enthält und abgeschlossen ist unter f.

Schema der induktiven Definition

$\mathcal{A}$ ist die kleinste Menge, für die gilt:

- $A_0 \subseteq \mathcal{A}$
- $x_1, \ldots, x_n \in \mathcal{A} \Rightarrow f(x_1, \ldots, x_n) \in \mathcal{A}$
  ($\mathcal{A}$ ist abgeschlossen unter f)

Reguläre Sprachen

Reguläre Mengen (Sprachen)

Die Menge $\mathcal{L}_{\text{reg}}(\Sigma)$ der regulären Mengen (Sprachen) über Σ ist die kleinste Menge, sodass

- $\{\}, \{a\} \in \mathcal{L}_{\text{reg}}(\Sigma)$ für alle $a \in \Sigma$
- $A, B \in \mathcal{L}_{\text{reg}}(\Sigma) \Rightarrow A \cup B, A \cdot B, A^* \in \mathcal{L}_{\text{reg}}(\Sigma)$

Reguläre Ausdrücke (Algebraische Notation)

s	statt	$\{s\}$ für $s \in \Sigma$	$L_1 + L_2$	statt	$L_1 \cup L_2$
ε	statt	$\{\varepsilon\}$	$L_1 L_2$	statt	$L_1 \cdot L_2$
$\emptyset$	statt	$\{\}$	L^*	bleibt	L^*

* hat die höchste Priorität, $+$ die niedrigste.

Anmerkung: Genau genommen, ist ein regulärer Ausdruck E keine Sprache. Will man auf die Sprache Bezug nehmen, die von E beschrieben wird, sollte man eigentlich $L(E)$ verwenden.

Deterministische endliche Automaten (DEA)

DEA

Ein deterministischer endlicher Automat (DEA) ist ein 5-Tupel

$$\mathcal{A} = (Q, \Sigma, \delta, q_0, F), \text{wobei}$$

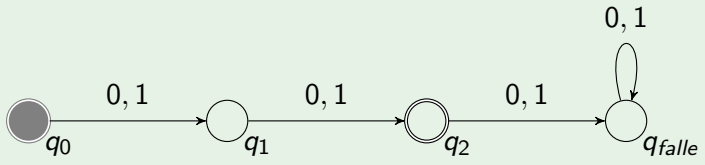
Q ... endliche Menge von Zuständen
Σ ... Eingabealphabet
$\delta\colon Q \times \Sigma \to Q$ ... Übergangsfunktion (total)
$q_0 \in Q$ ... Anfangszustand
$F \subseteq Q$ ... Menge von Endzuständen

DEA: Falle

Falle

Sei $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$ ein DEA und $q \in Q - F$ mit $\delta(q, a) = q$ für alle $a \in \Sigma$. Dann heißt q *Falle*.

Beispiel: Falle



Akzeptierte Sprache: $\{\varepsilon, 00, 01, 10, 11\}$

Deterministischer endlicher Automat

Um das Verhalten eines DEA auf einer Zeichenkette formal zu beschreiben, erweitern wir die Übergangsfunktion δ auf beliebige Wörter aus Σ^*:

Erweiterte Übergangsfunktion

$\delta^*\colon Q \times \Sigma^* \to Q$

$$\delta^*(q, \varepsilon) = q, \qquad \delta^*(q, aw) = \delta^*(\delta(q, a), w)$$

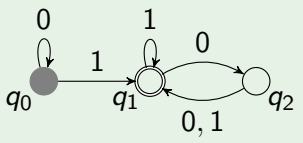
für alle $q \in Q$, $w \in \Sigma^*$, $a \in \Sigma$.

Akzeptierte Sprache

$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}$

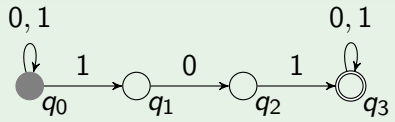
Nichtdeterministische endliche Automaten (NEA)

DEA



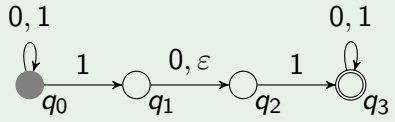
Von einem Zustand aus gibt es mit ein und demselben Eingabesymbol genau einen Folgezustand.

NEA



Von einem Zustand aus kann es mit ein und demselben Eingabesymbol mehrere Folgezustände geben.

ε-NEA



Von einem Zustand aus kann es mit ein und demselben Eingabesymbol mehrere Folgezustände geben, Übergänge sind auch mit ε möglich (ε-Übergänge).

Nichtdeterministische endliche Automaten (NEA)

Übergangsfunktion NEA: $\delta\colon Q \times \Sigma \to \mathcal{P}(Q)$

Übergangsfunktion ε-NEA: $\delta\colon Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$

Erweiterte Übergangsfunktion: $\delta^*\colon Q \times \Sigma^* \to \mathcal{P}(Q)$

$$\delta^*(q, w) = \{q' \in Q \mid q \rightsquigarrow wq'\}$$

$q \rightsquigarrow wq'$ ... es gibt einen mit w beschrifteten Pfad von q nach q'

Akzeptierte Sprache:

$$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \cap F \neq \{\}\}$$

Automaten $\mathcal{A}$ und $\mathcal{A}'$ sind äquivalent, falls $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

Determinisierung (NEA $\to$ DEA)

Zu jedem NEA gibt es einen äquivalenten DEA.

Beweis (Teilmengenkonstruktion)

NEA: $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$
DEA: $\hat{\mathcal{A}} = \langle \hat{Q}, \Sigma, \hat{\delta}, \hat{q}_0, \hat{F}\rangle$, wobei

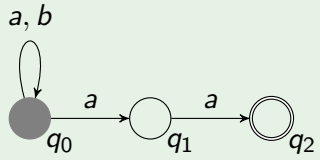
$$\begin{aligned}
\hat{Q} &= \mathcal{P}(Q) \\
\hat{\delta}(\hat{q}, a) &= \bigcup_{q \in \hat{q}} \delta^*(q, a) \qquad \text{für alle } \hat{q} \in \hat{Q},\ a \in \Sigma \\
\hat{q}_0 &= \{q_0\} \\
\hat{F} &= \begin{cases} \{\hat{q} \in \hat{Q} \mid \hat{q} \cap F \neq \{\}\} \cup \{\hat{q}_0\} & \textit{falls } \varepsilon \in \mathcal{L}(\mathcal{A}) \\ \{\hat{q} \in \hat{Q} \mid \hat{q} \cap F \neq \{\}\} & \textit{sonst} \end{cases}
\end{aligned}$$

□

Tipp: Berechne die Übergangsfunktion ausgehend von $\hat{q}_0$ nur für tatsächlich erreichbare Zustände.

Determinisierung: Beispiel

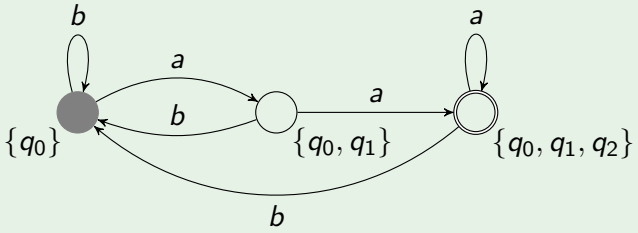
NEA:



δ	a	b
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\{q_2\}$	$\{\}$
q_2	$\{\}$	$\{\}$

$\hat{\delta}$	a	b
SZ $\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$	$\{q_0\}$
EZ $\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0\}$

DEA:



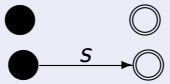
Reguläre Menge → ε-NEA

Zu jeder regulären Sprache L gibt es einen endlichen Automaten $\mathcal{A}$, sodass $L = \mathcal{L}(\mathcal{A})$.

Beweis

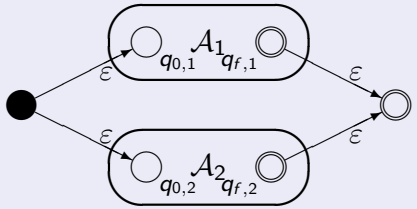
$L = \{\}$:

$L = \{s\}$:



$s \in (\Sigma \cup \{\varepsilon\})$

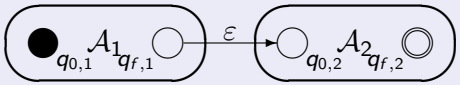
$L = L_1 \cup L_2$:



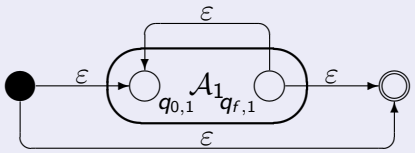
Reguläre Menge → ε-NEA

Beweis ctd.

$L = L_1 \cdot L_2$:



$L = (L_1)^*$:



□

DEA → reguläre Menge

Jede von einem DEA akzeptierte Sprache ist regulär.

Beweis

$\mathcal{A} = \langle \{q_1, \ldots, q_n\}, \Sigma, \delta, q_1, F \rangle$

R_{ij}^k ... Menge aller Wörter, mit denen man von q_i nach q_j gelangt, ohne einen Zustand mit einem Index größer als k zu berühren.

$$R_{ij}^0 = \begin{cases} \{s \in \Sigma \mid \delta(q_i, s) = q_j\} & i \neq j \\ \{s \in \Sigma \mid \delta(q_i, s) = q_j\} \cup \{\varepsilon\} & i = j \end{cases}$$

$$R_{ij}^k = R_{ij}^{k-1} \cup R_{ik}^{k-1} \cdot (R_{kk}^{k-1})^* \cdot R_{kj}^{k-1} \quad k > 0$$

$$\mathcal{L}(\mathcal{A}) = \bigcup_{q_j \in F} R_{1j}^n$$

□

Einschub: Graphen

Markierte gerichtete Graphen

Seien U und W endliche Mengen. Ein markierter gerichteter Graph g über U und W ist ein Tripel (K, E, L) wobei

- K die Menge der Knoten
- $E \subseteq K \times W \times K$ die Menge der Kanten
- $L : K \to U$ die Knotenmarkierungsfunktion ist.

Ein Element $(x, w, y) \in E$ stellt eine *gerichtete Kante* vom Knoten x zum Knoten y mit Markierung w dar.

Äquivalenz und Isomorphie

Zwei markierte gerichtete Graphen $g = (K, E, L)$ und $g' = (K', E', L')$ über U und W heißen *isomorph*, wenn wenn es eine bijektive Abbildung $f : K \to K'$ gibt, sodass für alle $k, l \in K$ und $w \in W$ gilt: $(k, w, l) \in E$ genau dann wenn $(f(k), w, f(l)) \in E'$.
Gilt außerdem $L(k) = L'(f(k))$ für alle $k \in K$, so heißen g und g' *äquivalent*.

Minimierungsalgorithmus von Brzozowski[1]

Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein DEA mit $L(A) = L$. Dann ist $A^r = (Q, \Sigma, \{(q, a, p) \mid (p, a, q) \in \delta\}, F, \{q_0\})$ mit $L(A^r) = L^r$.

Minimierungsalgorithmus von Brzozowski

Gegeben ein DEA A. Um den zu A äquivalenten Minimalautomaten C zu erhalten:

1. konstruiere einen NEA A^r durch "Spiegelung" von A
2. konstruiere einen DEA B durch Determinisierung von A^r
3. konstruiere einen NEA B^r durch "Spiegelung" von B
4. konstruiere einen DEA C durch Determinisierung von B^r

[1] Janusz (John) Antoni Brzozowski (*1935)

Abschlusseigenschaften regulärer Sprachen

$\mathcal{L}_{\text{reg}}(\Sigma)$ ist **abgeschlossen** gegenüber:

- Vereinigung, Verkettung, Stern-Operator: per Definition.
- Plus-Operator: $A^+ = A^* \cdot A$.
- Komplement bzgl. Σ^*: konstruiere DEA und vertausche End- und Nichtendzustände. (Falle nicht vergessen!)
- Durchschnitt: $A \cap B = \overline{\overline{A} \cup \overline{B}}$.
- Differenz: $A - B = A \cap \overline{B}$.
- Spiegelung: Konstruiere EA, vertausche Start- mit Endzustand, kehre alle Zustandsübergänge um.
- Homomorphismen: repräsentiere L durch regulären Ausdruck und ersetze alle Vorkommnisse von a durch Ausdruck für $h(a)$.

Homomorphismus

Seien Σ und Γ zwei Alphabete. **Homomorphismus**: $h : \Sigma \longrightarrow \Gamma^*$
Induktive Erweiterung auf Wörter über Σ:

1. $h(\varepsilon) = \varepsilon$
2. $h(wa) = h(w)h(a)$ für alle $w \in \Sigma^*$ und $a \in \Sigma$.

Anwendung von h auf jedes Wort der Sprache L:

$$h(L) = \{h(w) \mid w \in L\}$$

Beispiel

Sei $h : \{0, 1\}^* \longrightarrow \{a, b\}^*$ ein Homomorphismus mit $h(0) = ab$ und $h(1) = \varepsilon$

Dann ist $h(0011) = abab$ und $h(L(10^*1)) = L((ab)^*)$

Grenzen der Regularität

Sei $L = \{0^n 1^n \mid n \geq 0\}$. Gibt es einen DEA für L?

Es sieht so aus als müsste sich die Maschine die Anzahl der Nullen merken. Nachdem ebendiese Anzahl aber nicht limitiert ist, müsste sich der DEA eine unbeschränkte Anzahl von Möglichkeiten merken können. Mit einer endlichen Anzahl von Zuständen ist das aber nicht möglich!

Aber: Nur weil es so aussieht, als ob eine Sprache unbegrenzten Speicher brauchen würde, ist das nicht notwendigerweise so!

Regulär oder nicht?

$A = \{w \mid w$ enthält gleich viele Symbole 0 wie Symbole $1\}$
$B = \{w \mid$ die Teilwörter[2] 01 und 10 kommen gleich oft in w vor$\}$

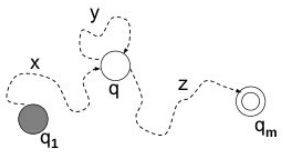
[2] Für ein Wort $w \in \Sigma^*$, wobei $w = xuy$ für Wörter $x, u, y \in \Sigma^*$ heißt u **Teilwort** von w.

Schubfachprinzip[3]

Schubfachprinzip (pigeonhole principle)

Bei Verteilung von $n+1$ Gegenständen auf n Schubfächer müssen in mindestens einem Schubfach zwei Gegenstände landen.

DEA $\mathcal{A}$ habe m Zustände. Gilt für ein von $\mathcal{A}$ akzeptiertes Wort w, dass $w \geq m$, so muss mindestens ein Zustand mehr als einmal durchlaufen werden. (Schubfachprinzip!)
Zerlege w in xyz: y gehe von q nach q; diese Schleife kann ausgelassen bzw. beliebig oft wiederholt werden, d.h. xy^iz wird für alle i ebenfalls von $\mathcal{A}$ akzeptiert!



[3] Johann Peter Gustav Lejeune Dirichlet, 1805-1859

Pumping Lemma für reguläre Sprachen

(Unendliche) Reguläre Sprachen haben eine spezielle Eigenschaft: Jedes Wort ab einer bestimmten Länge kann "aufgepumpt" werden, d.h. jedes solche Wort enthält ein Teilstück, das beliebig oft wiederholt werden kann, wobei die resultierenden Wörter noch immer in derselben Sprache liegen.

Pumping Lemma für reguläre Sprachen

Sei L eine unendliche reguläre Sprache. Dann gibt es eine (nur von L abhängige) Schranke $m > 0$ so, dass für jedes Wort $w \in L$ mit $|w| \geq m$ Wörter x, y, z so existieren, dass

$$w = xyz \qquad \text{mit } |xy| \leq m \text{ und } |y| > 0$$

sowie

$$w_i = xy^iz \in L \qquad \text{für alle } i \geq 0.$$

Reguläres Pumping Lemma - Beispiel

Zu zeigen: $L = \{a^n b^n \mid n \geq 0\}$ ist nicht regulär.

Beweis durch Widerspruch:

Angenommen L sei regulär. Für **beliebiges** m (Konstante aus dem Pumping Lemma) wähle **ein** $w \in L$ mit $|w| \geq m$, z.B.

$$w = a^m b^m$$

Betrachte **beliebige** Zerlegungen von w in xyz, wobei $|xy| \leq m$ und $|y| > 0$: xy kann nur aus Symbolen a bestehen.

Wähle **ein** i so, dass xy^iz nicht von der Gestalt $a^n b^n$ ist:

z.B. $i = 2$, dann müsste – wäre L regulär – auch $xy^2z = a^{m+|y|}b^m$ aus L sein, was aber nicht der Fall ist! Widerspruch! L kann nicht regulär sein.

Grammatiken

Satz	$\rightarrow$	*HwP ZwP*	*Art*	$\rightarrow$	*die* | *das*
HwP	$\rightarrow$	*Art Hw*	*Hw*	$\rightarrow$	*Studentin* | *Problem*
ZwP	$\rightarrow$	*Hzw HwP Zw*	*Hzw*	$\rightarrow$	*wird*
			Zw	$\rightarrow$	*lösen*

Satz
$\Rightarrow$ *HwP ZwP*
$\Rightarrow_P$ *Art Hw Hzw HwP Zw*
$\Rightarrow_P$ *die Studentin wird Art Hw lösen*
$\Rightarrow_P$ *die Studentin wird das Problem lösen*

Grammatik: Formale Definition

Grammatik

Grammatik $G = \langle N, T, P, S \rangle$, wobei

N ... endliche Menge von Nonterminalen (Variablen)
T ... endliche Menge von Terminalsymbolen ($N \cap T = \{\}$)
$P \subseteq (N \cup T)^+ \times (N \cup T)^*$ ... Produktionen
$S \in N$ ... Startsymbol

Wir schreiben
$\alpha \rightarrow \beta$ statt $(\alpha, \beta) \in P$
$\alpha \rightarrow \beta_1 \mid \cdots \mid \beta_n$ statt $\alpha \rightarrow \beta_1, \ldots, \alpha \rightarrow \beta_n$

Satzformen und erzeugte Sprache

Gilt $S \Rightarrow w$ für ein Wort $w \in (N \cup T)^*$, so nennt man w **Satzform**.

Menge aller in n Schritten ableitbaren Satzformen:
$SF(G, n) = \{w \in (N \cup T)^* | S \Rightarrow^n w\}$

Erzeugte Sprache

Von G erzeugte *Sprache* $\mathcal{L}(G)$:

$$\mathcal{L}(G) \quad = \quad \{w \in T^* \mid S \Rightarrow^* w\}$$

Grammatiken G_1 und G_2 heißen äquivalent, wenn $\mathcal{L}(G_1) = \mathcal{L}(G_2)$ gilt.

Grammatiken: Beispiele

$\{a^n \mid n \geq 0\}$

$$G_1 = (\{S\}, \{a\}, \{S \rightarrow \varepsilon, S \rightarrow aS\}, S)$$

$$\mathcal{L}(G_1) = \{\varepsilon\} \cup \{a^n \mid n \geq 1\} = \{a\}^*$$

Alle in G_1 möglichen Ableitungen sind von der Gestalt

$$S \Rightarrow \varepsilon \quad \text{bzw.} \quad S \Rightarrow aS \Rightarrow^n a^{n+1}S \Rightarrow a^{n+1}$$

für ein $n \geq 0$.

Grammatiken: Beispiele

$\{a^n b^n | n \geq 0\}$

$$G_2 = (\{S\}, \{a, b\}, \{S \to aSb, S \to \varepsilon\}, S)$$

$$\mathcal{L}(G_2) = \{a^n b^n | n \geq 0\}$$

Alle in G_2 möglichen Ableitungen sind von der Gestalt

$$S \Rightarrow^n a^n S b^n \Rightarrow a^n b^n \text{ für alle } n \geq 0$$

Formaler Beweis mittels *Induktion*:
Menge aller Satzformen nach genau $n \geq 1$ Schritten: $SF(G_2, n) = \{a^n S b^n, a^{n-1} b^{n-1}\}$

Induktionsbasis: $SF(G_2, 1) = \{a^1 S b^1, \varepsilon\}$
Induktionshypothese: $SF(G_2, n) = \{a^n S b^n, a^{n-1} b^{n-1}\}$
Induktionsbehauptung: $SF(G_2, n+1) = \{a^{n+1} S b^{n+1}, a^n b^n\}$

Beweis: Das Wort $a^{n-1} b^{n-1}$ ist terminal und daher nicht mehr weiter ableitbar. Aus $a^n S b^n$ ist ableitbar:

mittels $S \to aSb$: $a^{n+1} S b^{n+1}$ mittels $S \to \varepsilon$: $a^n b^n$

Grammatiken: Beispiele

$\{a^n b^n c^n | n \geq 1\}$

$$G_3 = (\{S, A, C\}, \{a, b, c\}, P_3, S) \text{wobei}$$

$$P_3 = \{S \to abc, S \to aAbc, A \to aAbC, A \to abC, Cb \to bC, Cc \to cc\}$$

$$\mathcal{L}(G_3) = \{a^n b^n c^n | n \geq 1\}$$

Alle in G_3 möglichen Ableitungen sind von der Gestalt $S \Rightarrow abc$ bzw. für $n \geq 2$:

$$S \Rightarrow aAbc \Rightarrow^{n-2} a^{n-1} A (bC)^{n-2} bc \Rightarrow a^n (bC)^{n-1} bc \Rightarrow^* a^n b^n c^n$$

Grammatik-Typen: Typ-i-Grammatiken

Typ-i-Grammatiken

Sei $G = \langle N, T, P, S \rangle$ eine Grammatik. Dann heißt G auch **unbeschränkte Grammatik** (*Typ-0*).
Gilt für alle Produktionen $\alpha \to \beta \in P$

- $|\alpha| \leq |\beta|$ so heißt G **monoton**;
- $\alpha = uAv$ und $\beta = uwv$ für ein $A \in N$, $w \in (N \cup T)^+$ und $u, v \in (N \cup T)^*$ so heißt G **kontextsensitiv** (**Typ-1**);
- $A \to \beta$ für ein $A \in N$, so heißt G **kontextfrei** (**Typ-2**);
- $A \to aB$ oder $A \to \varepsilon$ für $A, B \in N$ und $a \in T$, so heißt G **regulär** (**Typ-3**).

Für monotone und kontextsensitive Grammatiken gilt: Kommt S nicht auf der rechten Seite einer Produktion vor, so ist auch die Produktion $S \to \varepsilon$ erlaubt.

Erzeugte Sprachen

Erzeugte Sprachen

Eine formale Sprache heißt **rekursiv aufzählbar**, **monoton**, **kontextsensitv**, **kontextfrei** bzw. **regulär**, wenn sie von einer Typ-0-, monotonen, Typ-1-, Typ-2-, bzw. Typ-3-Grammatik erzeugt wird.

Aufgrund der Definition können wir nun die einzelnen Sprachen aus den vorigen Beispielen klassifizieren:

Es ergibt sich, dass
$\mathcal{L}(G_1) = \{a^n | n \geq 0\}$ regulär
$\mathcal{L}(G_2) = \{a^n b^n | n \geq 0\}$ kontextfrei und
$\mathcal{L}(G_3) = \{a^n b^n c^n | n \geq 1\}$ monoton ist.

Reguläre Grammatiken

Reguläre Grammatiken [1]

Eine Grammatik heißt *regulär*, wenn alle Produktionen von der Form

$$A \to aB \qquad \text{oder} \qquad A \to \varepsilon$$

sind ($A, B \in N$, $a \in T$).

Alternativ:

Reguläre Grammatiken [2]

Eine Grammatik heißt *regulär*, wenn alle Produktionen von der Form

$$A \to aB \qquad \text{oder} \qquad A \to a$$

sind ($A, B \in N$, $a \in T$). Um auch das Leerwort erzeugen zu können, ist $S \to \varepsilon$ erlaubt, sofern S nicht auf der rechten Seite einer Produktion vorkommt.

Reguläre Grammatiken: Beispiele

$L = \{a\}^*$

$$\begin{aligned} G_1 &= (\{S\}, \{a\}, \{S \to aS \mid \varepsilon\}, S) && \text{(Def. [1])} \\ G_2 &= (\{S, T\}, \{a\}, \{S \to aT \mid a \mid \varepsilon, T \to aT \mid a\}, S) && \text{(Def. [2])} \end{aligned}$$

$$\mathcal{L}(G_1) = \mathcal{L}(G_2) = L$$

$L = \{a\}^+$

$$\begin{aligned} G_1 &= (\{S, T\}, \{a\}, \{S \to aT, T \to aT \mid \varepsilon\}, S) && \text{(Def. [1])} \\ G_2 &= (\{S\}, \{a\}, \{S \to aS \mid a\}, S) && \text{(Def. [2])} \end{aligned}$$

$$\mathcal{L}(G_1) = \mathcal{L}(G_2) = L$$

Kontextfreie Grammatiken

Kontextfreie Grammatik

Eine Grammatik heißt *kontextfrei*, wenn alle Produktionen von der Form $A \to \beta$ sind, wobei $A \in N$ und $\beta \in (N \cup T)^*$.
Eine Sprache L heißt kontextfrei, wenn es eine kontextfreie Grammatik G gibt, sodass $L = \mathcal{L}(G)$.

Linksableitung:

$xAy \Rightarrow_L x\beta y$ falls $A \to \beta \in P$ und $x \in T^*$

Rechtsableitung:

$xAy \Rightarrow_R x\beta y$ falls $A \to \beta \in P$ und $y \in T^*$

Parallelableitung:

$x_0 A_1 x_1 \cdots A_n x_n \Rightarrow_P x_0 \beta_1 x_1 \cdots \beta_n x_n$ falls
$A_1 \to \beta_1, \ldots, A_n \to \beta_n \in P$ und $x_0, \ldots, x_n \in T^*$

Kontextfreie Grammatiken: Beispiele

Palindrome über $\{a; b\}$

$$\begin{aligned} G = \langle\, &\{S\},\ \{a, b\}, \\ &\{S \to \varepsilon \mid a \mid b \mid a\,S\,a \mid b\,S\,b\},\ S \,\rangle \end{aligned}$$

$\mathcal{L}(G) = \{\varepsilon, a, b, aa, bb, aaa, aba, bab, bbb, \ldots\}$

$L = \{w \in \{a, b\}^* \mid |w|_a = 2 \cdot |w|_b\}$

$G = \langle \{S\}, \{a, b\}, P, S \rangle$, wobei
$P = \{S \to a\,S\,a\,S\,b\,S \mid a\,S\,b\,S\,a\,S \mid b\,S\,a\,S\,a\,S \mid \varepsilon\}$

Kontextfreie Grammatiken: Beispiele

$L = \{a^m + b^n = c^{m+n} \mid m, n \geq 0\}$

Kontextfreie Grammatik $G = \langle\{S, T\}, \{a, b, c, +, =\}, P, S\rangle$
wobei P:

$$P = \{ \begin{array}{lcl} S & \rightarrow & a\,S\,c \mid + T \\ T & \rightarrow & b\,T\,c \mid = \end{array} \}$$

$aa + bb = cccc \in \mathcal{L}(G)$:

$$\begin{array}{lllllll} S & \Rightarrow & a\,S\,c & \Rightarrow & aa\,S\,cc & \Rightarrow & aa + T\,cc \\ & \Rightarrow & aa + b\,T\,ccc & \Rightarrow & aa + bb\,T\,cccc & \Rightarrow & aa + bb = cccc \end{array}$$

Einschub: Bäume

Baum

Ein (geordneter, markierter gerichteter) *Baum* über U und W ist ein Graph $g = (K, E, L)$ über U und W, der folgende Bedingungen erfüllt:

1. $W = \{1, ..., n\}$ für ein $n \geq 1$.
2. Es gibt genau einen ausgezeichneten Knoten p_0 (Wurzel), der keinen Vorgänger hat. Außerdem gibt es von der Wurzel aus zu jedem anderen Knoten q von g einen Pfad $(p_0, e_1, p_1, \ldots, p_{k-1}, e_k, q)$ der Länge $k \geq 1$, $(p_i, e_{i+1}, p_{i+1}) \in E$, $0 \leq i < k$, $q = p_k$.
3. Jeder von der Wurzel verschiedene Knoten hat genau einen Vorgänger.
4. Jeder Knoten ohne Nachfolger heißt Blatt.
5. Ist p kein Blatt, so sind die Nachfolger von p geordnet (die Kanten tragen die Bezeichnungen 1 bis k für ein k).

Ableitungsbäume für k.f. Grammatiken

Ableitungsbaum

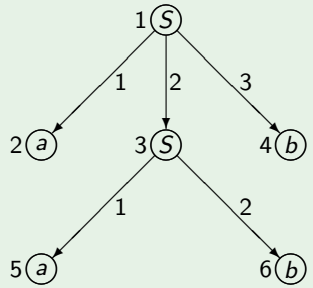
Sei $G = (N, T, P, S)$ eine kontextfreie Grammatik. Ein Baum $g = (K, E, L)$ über $U = N \cup T \cup \{\varepsilon\}$ und $W = \{1, ..., n\}$ heißt *Ableitungsbaum* für G, wenn Folgendes gilt:

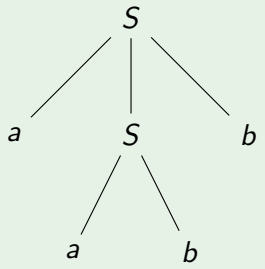
1. Ist p_0 die Wurzel von g, so gilt $L(p_0) = S$.
2. Ist p kein Blatt, so muss $L(p) \in N$ gelten.
3. Ist p ein Blatt mit $L(p) = \varepsilon$, so ist p der einzige Nachfolger seines Vorgängers.
4. Ist $\{(p, i, q_i) \mid i \in 1, ..., n\}$ die geordnete Menge der von p mit $L(p) = A$ wegführenden Kanten, so ist $A \rightarrow L(q_1) \ldots L(q_k)$ eine Produktion in P.

A-Baum für G: für Wurzel gilt $L(p_0) = A$
Ein Ableitungsbaum ist ein S-Baum für G.

Ableitungsbäume: Beispiel

$G = (\{S\}, \{a, b\}, \{S \rightarrow aSb, S \rightarrow ab\}, S)$
Ableitungsbaum für G:





Front des Ableitungsbaums:
ergibt sich als Folge der Labels der Knoten 2, 5, 6, 4 zu *aabb*

Eindeutigkeit, (inhärente) Mehrdeutigkeit

Sei $G = \langle N, T, P, S \rangle$ eine kontextfreie Grammatik. G heißt **eindeutig**, wenn es zu jedem in G ableitbaren Terminalwort genau eine Linksableitung in G gibt. Ansonsten heißt G **mehrdeutig**.

Eine kontextfreie *Sprache* L heißt **inhärent mehrdeutig**, wenn jede Grammatik, die L erzeugt, mehrdeutig ist.

Mehrdeutigkeit: Beispiel 1

$G = \langle \{E\}, \{+, *, (,), id\}, P, E \rangle$ wobei
$P = \{E \to E + E \mid E * E \mid (\ E\ ) \mid id\}$

G ist mehrdeutig:

$$\begin{aligned} E &\Rightarrow_L E + E \Rightarrow_L \left\{ \begin{matrix} E + E + E \\ id + E \end{matrix} \right\} \Rightarrow_L id + E + E \Rightarrow_L id + id + E \\ &\Rightarrow_L id + id + id \end{aligned}$$

$\mathcal{L}(G)$ ist aber nicht mehrdeutig, da es eindeutige Grammatik gibt:

$$G' = \langle \{E, T, F\}, \{+, *, (,), id\}, P', E \rangle$$

wobei

$$P' = \{E \to E + T \mid T,\ T \to T * F \mid F,\ F \to (\ E\ ) \mid id\}$$

Inhärent mehrdeutige Sprachen: Beispiel

Die kontextfreie Sprache $L = L_1 \cup L_2$ mit

$L_1 = \{a^m b^m c^n \mid m, n \geq 1\}$ und $L_2 = \{a^m b^n c^n \mid m, n \geq 1\}$

ist inhärent mehrdeutig.
L ist kontextfrei:

$$\begin{array}{lcl} S & \to & S_1 \mid S_2 \\ S_1 & \to & S_1\, c \mid A \\ A & \to & a\, A\, b \mid \varepsilon \end{array} \qquad \begin{array}{lcl} S_2 & \to & a\, S_2 \mid B \\ B & \to & b\, B\, c \mid \varepsilon \end{array}$$

Grammatik ist mehrdeutig, da $a^n b^n c^n$ zwei verschiedene Ableitungen besitzt.
Man kann zeigen, dass *alle* Grammatiken für L mehrdeutig sind, d.h., L ist inhärent mehrdeutig.

Bemerkung: $L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 1\}$

Transformation von Produktionen

Nutzlose Produktionen: Entfernen aller Produktionen, die vom Startsymbol aus nicht erreichbar sind oder aus denen kein $w \in T^*$ ableitbar ist.

ε-**Produktionen:** Entfernen aller Produktionen $A \to \varepsilon$

Einheitsproduktionen: Entfernen aller Produktionen $A \to B$

Zu jeder kontextfreien Sprache ohne Leerwort gibt es eine kontextfreie Grammatik ohne nutzlose, ohne ε- und ohne Einheitsproduktionen.

1. Entfernen nutzloser Produktionen (Teil 1)

Menge aller Variablen, aus denen direkt ein *Terminalwort ableitbar* ist:

$$N_1^{(1)} = \{A \in N \mid A \to w \in P, w \in T^*\}$$

Für $i > 1$ definieren wir iterativ

$$N_1^{(i)} = \{A \in N \mid A \to w \in P, w \in (N_1^{(i-1)} \cup T)^*\} \cup N_1^{(i-1)}$$

Sobald $N_1^{(m+1)} = N_1^{(m)}$ (für ein $m > 1$), erhalten wir
$G_1 = \langle N_1, T, P_1, S\rangle$, wobei
$N_1 = N_1^{(m)}$
P_1 enthält alle Produktionen aus P, welche nur Symbole aus $N_1 \cup T$ enthalten.

Vorbereitende Schritte zur Chomsky NF: Beipiel

Schritt 1

$G = \langle\{S, A, B, C, D\}, \{a\}, P, S\rangle$ mit
$P = \{S \to aA \mid B \mid D, A \to aB, B \to A, B \to \varepsilon, C \to a\}$

Wir untersuchen, ob aus jeder Variablen in G ein Terminalwort ableitbar ist:

- $N_1^{(1)} = \{B, C\}$
- $N_1^{(2)} = \{S, A\} \cup \{B, C\} = N_1^{(3)}$.

Aus D ist kein Terminalwort ableitbar, wir erhalten also

$$\begin{aligned} G_1 &= \langle N_1, \{a\}, P_1, S\rangle \text{ mit} \\ N_1 &= \{S, A, B, C\} \\ P_1 &= \{S \to aA \mid B, A \to aB, B \to A, B \to \varepsilon, C \to a\} \ . \end{aligned}$$

Nun ist in G_1 aus jeder Variablen ein Terminalwort ableitbar und es gilt $\mathcal{L}(G_1) = \mathcal{L}(G)$.

2. Entfernen nutzloser Produktionen (Teil 2)

Ausgehend von G_1 bestimmen wir nun die Menge aller Symbole V, die vom Startsymbol S aus *erreichbar* sind:
$V_2^{(1)} = \{S\}$
sowie für $i > 1$

$$\begin{aligned} V_2^{(i)} &= \{\alpha \in N_1 \cup T \mid A \to u\alpha v \in P_1, A \in (V_2^{(i-1)} \cap N_1), \\ &\qquad u, v \in (N_1 \cup T)^*\} \\ &\cup \ V_2^{(i-1)} \end{aligned}$$

Soblad $V_2^{(m+1)} = V_2^{(m)}$, erhalten wir $G_2 = \langle N_2, T_2, P_2, S\rangle$ (mit $\mathcal{L}(G_2) = \mathcal{L}(G)$), wobei
$N_2 = V_2^{(m)} \cap N_1$, sowie
$T_2 = V_2^{(m)} \cap T_1$ und
P_2 enthält alle Produktionen, die nur Symbole aus $N_2 \cup T_2$ enthalten.

Vorbereitende Schritte zur Chomsky NF: Beipiel ctd.

Schritt 2

Ausgehend von G_1 überprüfen wir, ob alle Symbole vom Startsymbol S aus erreichbar sind:

- $V_2^{(1)} = \{S\}$
- $V_2^{(2)} = \{A, B, a\} \cup \{S\} = V_2^{(3)}$

C nicht vom Startsymbol erreichbar. Daher erhalten wir

$$\begin{aligned} G_2 &= \langle N_2, \{a\}, P_2, S\rangle \text{ mit} \\ N_2 &= \{S, A, B\} \\ P_2 &= \{S \to aA \mid B, A \to aB, B \to A, B \to \varepsilon\} \ . \end{aligned}$$

Mit G_2 haben wir nun also eine zu G äquivalente Grammatik ohne nutzlose Symbole und es gilt $\mathcal{L}(G_2) = \mathcal{L}(G)$.

3. Elimination der ε-Produktionen

Ausgehend von der zu G äquivalenten Grammatik G_2 ohne nutzlose Symbole, bestimmen wir nun die Menge aller Variablen, aus denen das Leerwort ableitbar ist (n_2 bezeichnet die Anzahl der Symbole in N_2):

$$N_3^{(1)} = \{A \in N_2 \mid A \to \varepsilon \in P_2\}.$$

Für i mit $1 < i \leq n_2$ definieren wir iterativ

$$N_3^{(i)} = \{A \in N_2 \mid A \to w \in P_2, w \in (N_3^{(i-1)})^*\} \cup N_3^{(i-1)}.$$

Klarerweise gilt, dass aus einer Variablen A genau dann das Leerwort ableitbar ist, wenn $A \in N_3^{(n_2)}$. Daher erhält man sofort für die Startvariable S, dass $\varepsilon \in \mathcal{L}(G_2)$ $(= \mathcal{L}(G))$ genau dann, wenn $S \in N_3^{(n_2)}$.

3. Elimination der ε-Produktionen ctd.

Wir entfernen nun alle ε-Produktionen aus P_2 und nehmen anstelle einer Produktion $A \to X_1 ... X_k$ mit $X_i \in N_2 \cup T$, $1 \leq i \leq k, k \geq 1$, aus P_2 alle Produktionen $A \to Y_1 ... Y_k$ in P_3' auf, die folgende Bedingungen erfüllen:

1. $Y_1 ... Y_k \neq \varepsilon$
2. für alle i mit $1 \leq i \leq k$:
   1. $Y_i = X_i$ für $X_i \in N_2 - N_3^{(n_2)}$
   2. $Y_i = X_i$ oder $Y_i = \varepsilon$ für $X_i \in N_3^{(n_2)}$

Wiederholen wir nun auf $\langle N_3, T_2, P_3', S\rangle$ die Schritte 1 und 2, um eventuell neu entstandene nutzlose Symbole zu entfernen, so erhalten wir schließlich G_3.

Achtung: $\mathcal{L}(G_3) = \mathcal{L}(G) - \varepsilon$!!

Vorbereitende Schritte zur Chomsky NF: Beipiel ctd.

Schritt 3

Nun sehen wir uns die ε-Produktionen in G_2 an:

- $N_3^{(1)} = \{B\}$
- $N_3^{(2)} = \{S\} \cup \{B\}$ Also: $\varepsilon \in \mathcal{L}(G)$!

Wir erhalten somit

$$\begin{aligned} G_3 &= \langle N_3, \{a\}, P_3, S\rangle \text{ mit} \\ N_3 &= N_2 = \{S, A, B\} \\ P_3 &= \{S \to aA \mid B, A \to aB \mid a, B \to A\} \ . \end{aligned}$$

G_3 enthält nun weder nutzlose Symbole noch ε-Produktionen, und es gilt $\mathcal{L}(G_3) = \mathcal{L}(G) - \varepsilon$.

4. Elimination von Einheitsproduktionen:

Ausgehend von G_3 eliminieren wir die dort enthaltenen Einheitsproduktionen folgendermaßen:
Ist $B_0 \to B_1$ eine Produktion in P_3.
Betrachte alle Ableitungen $B_0 \Rightarrow B_1 \Rightarrow B_2 \Rightarrow ... \Rightarrow B_k \Rightarrow w, w \notin N_3$, (d.h., die im letzten Schritt angewendete Produktion ist keine Einheitsproduktion.)
Ersetze nun jede Produktion der Gestalt $B_0 \to B_1$ durch die Produktionen $B_0 \to w$.
Nachdem es jedoch passieren kann, dass durch dieses Verfahren wiederum nutzlose Symbole erzeugt werden, muss man noch einmal die Schritte 1 und 2 wiederholen, ehe man die gewünschte Grammatik G_4 erhält.

Vorbereitende Schritte zur Chomsky NF: Beipiel ctd.

Schritt 4

In G_3 gibt es noch die Einheitsproduktion $B \to A$, die wir nun eliminieren. Nachdem $B \Rightarrow A \Rightarrow aB$ bzw. $B \Rightarrow A \Rightarrow a$, ersetzen wir nun $B \to A$ durch $B \to aB \mid a$ sowie $S \to B$ durch $S \to aB \mid a$ und erhalten damit die reduzierte Grammatik

$$\begin{aligned} G_4 &= \langle N_4, \{a\}, P_4, S\rangle \text{ mit} \\ N_4 &= N_3 = N_2 = \{S, A, B\} \\ P_4 &= \{S \to aA \mid aB \mid a, A \to aB \mid a, B \to aB \mid a\} \end{aligned}$$

Wiederholt man die Schritte 1 und 2, so stellt man fest, dass in diesem Fall keine neuen nutzlosen Symbole erzeugt wurden.

Chomsky[4] Normalform

reduzierte Grammatik: Als Ergebnis der Schritte 1-4 erhalten wir eine sogenannte *reduzierte* Grammatik, also eine kontextfreie Grammatik, die weder nutzlose Symbole, noch ε- oder Einheitsproduktionen enthält.

Chomsky Normalform

Alle Produktionen besitzen die Form

$$A \to B\,C \qquad \text{oder} \qquad A \to a \qquad (a \in T,\ A, B, C \in N)\ .$$

Um das Leerwort erzeugen zu können, ist $S \to \varepsilon$ erlaubt, sofern S nicht auf der rechten Seite einer Produktion vorkommt.

Zu jeder kontextfreien Grammatik G kann man effektiv eine äquivalente Grammatik G' konstruieren, sodass $G' = \langle N', T', P', S'\rangle$ in Chomsky NF ist.

[4] Noam Chomsky, *1928

Chomsky NF (Beweis)

- Gilt $\mathcal{L}(G) = \{\}$: fertig
- Gilt $\mathcal{L}(G) \neq \{\}$: Von der reduzierten Grammatik $G_4 = \langle N_4, T_4, P_4, S\rangle$ (die $\mathcal{L}(G) - \varepsilon$ erzeugt) konstruiere G'':
  1. ersetze jedes Terminalsymbol $a \in T$ auf den rechten Seiten der Produktionen in P_4 durch eine entsprechende neue Variable X_a (soferne die rechte Seite nicht ohnehin nur aus einem einzigen Terminalsymbol besteht) und füge $X_a \to a$ hinzu.
  2. ersetze jede Produktion der Gestalt $A \to B_1 ... B_k$ für $k > 2$, durch $A \to B_1 Y_1$, $Y_1 \to B_2 Y_2$, ..., $Y_{k-2} \to B_{k-1} B_k$.

  Gemäß diesen Umformungen enthält G'' nur Produktionen der Gestalt $A \to a$ sowie der Gestalt $A \to BC$ für $a \in T'$ und $A, B, C \in N'$, d.h., G'' ist in Chomsky Normalform.

Chomsky NF (Beweis ctd.: $\varepsilon \in L(G)$?)

- Gilt $\varepsilon \notin L(G)$: fertig ($G' = G''$).
- Sonst:
  - S ist nicht auf der rechten Seite einer Produktion enthalten: Füge zu P' noch die Produktion $S \to \varepsilon$ hinzu ($S' = S$), um die zu G äquivalente reduzierte Grammatik G' in Chomsky Normalform zu erhalten.
  - sonst: füge noch ein neues Startsymbol S' hinzu (d.h., $N' = N'' \cup \{S'\}$) sowie die Produktionen $S' \to w$, wobei w die rechte Seite einer S-Produktion ist, d.h., $P' = P \cup \{S' \to w \mid S \to w \in P''\}$.

$\square$

Chomsky NF - Beispiel ctd.

Schritt 5 - Chomsky NF

Wir konstruieren zu G_4 (die $\mathcal{L}(G) - \varepsilon$ erzeugt) eine reduzierte Grammatik G' in Chomsky-Normalform mit $\mathcal{L}(G') = \mathcal{L}(G)$:
Ersetze a durch X_a in allen Produktionen, in denen a nicht alleine auf der rechten Seite vorkommt:

$$\{S \to X_aA \mid X_aB \mid a, A \to X_aB \mid a, B \to X_aB \mid a, X_a \to a\}.$$

Nachdem $\varepsilon \in \mathcal{L}(G)$, muss noch $S \to \varepsilon$ hinzugefügt werden, also:

$$\begin{aligned} G' &= \langle\{S, A, B, X_a, Y_1\}, \{a\}, P', S\rangle \text{ mit} \\ P' &= \{S \to X_aA \mid X_aB \mid a \mid \varepsilon, A \to X_aB, \\ &\quad A \to a, B \to X_aB \mid a, X_a \to a\} \end{aligned}$$

Greibach[5] Normalform für kf Grammatiken

Greibach Normalform:

Alle Produktionen haben die Form

$$A \to a\,w \qquad (a \in T,\ A \in N,\ w \in N^*)\ .$$

Erweiterte Greibach Normalform:

Alle Produktionen haben die Form

$$A \to a\,w \qquad (a \in T,\ A \in N,\ w \in (N \cup T)^*)\ .$$

Um das Leerwort erzeugen zu können, ist $S \to \varepsilon$ erlaubt, sofern S nicht auf der rechten Seite einer Produktion vorkommt.

Zu jeder kontextfreien Grammatik G kann man effektiv eine äquivalente Grammatik G' konstruieren, sodass G' in (erweiterter) Greibach NF ist.

[5] Sheila Greibach, *1939

Erweiterte Greibach Normalform - Beispiele

$L_1 = \left\{0^{2n}1^{4k}0^{2n} \mid n, k \geq 1\right\}$

Grammatik für L_1 in erweiterter Greibach Normalform:
$G_1 = \langle\{S, X\}, \{0, 1\}, P_1, S\rangle$ mit
$P_1 = \{S \to 0^2S0^2, S \to 0^2X0^2, X \to 1^4X, X \to 1^4\}$

Linksableitungen:
$S \Rightarrow^{n-1} 0^{2(n-1)}S0^{2(n-1)} \Rightarrow 0^{2n}X0^{2n} \Rightarrow^{k-1} 0^{2n}1^{4(k-1)}X0^{2n} \Rightarrow$
$0^{2n}1^{4k}0^{2n} \qquad \forall n, k \geq 1$

$L_2 = \{a^{n-4}b^ka^{5n} \mid n > 4, k \geq 3\}$

Grammatik für L_2 in erweiterter Greibach Normalform:
$G_2 = \langle\{S, A, B\}, \{a, b\}, P_2, S\rangle$ mit
$P_2 = \{S \to aAa^{25}, A \to aAa^5 \mid b^2B, B \to bB \mid b\}$

Eigenschaften kontextfreier Sprachen

Abschlusseigenschaften

Kontextfreie Sprachen sind **abgeschlossen** unter

- $\cup$, $\cdot$, *,
- Homomorphismen,
- Schnitt mit regulären Sprachen

Kontextfreie Sprachen sind **nicht abgeschlossen** unter

- Durchschnitt und Komplement.

Abschlusseigenschaften kontextfreier Sprachen

Kontextfreie Sprachen sind unter $\cup$, $\cdot$, * abgeschlossen.

Seien G_1, G_2 kontextfreie Grammatiken mit $N_1 \cap N_2 = \{\}$ und $S \notin N_1 \cup N_2$:

$$G_1 = \langle N_1, T_1, P_1, S_1 \rangle, \quad G_2 = \langle N_2, T_2, P_2, S_2 \rangle$$

Dann gilt:

$G_{union} = \langle N_1 \cup N_2 \cup \{S\}, T_1 \cup T_2, \{S \to S_1 \mid S_2\} \cup P_1 \cup P_2, S \rangle$
$G_{cat} = \langle N_1 \cup N_2 \cup \{S\}, T_1 \cup T_2, \{S \to S_1 \cdot S_2\} \cup P_1 \cup P_2, S \rangle$
$G_{star} = \langle N_1, T_1, \{S \to \varepsilon, S \to S_1 S\} \cup P_1, S \rangle$

Sowie

$\mathcal{L}(G_{union})$, $\mathcal{L}(G_{cat})$, $\mathcal{L}(G_{star})$ kontextfrei

□

WKA revisited: Dyck[6] Sprachen

Dyck Sprachen

Sei D_n über $\Gamma_n = \{(_1,)_1, ..., (_n,)_n\}$ die kleinste Menge, für die gilt:

- $\varepsilon \in D_n$.
- Ist $v \in D_n$, so ist auch $(_i v)_i \in D_n, 1 \leq i \leq n$.
- Ist $v_1, v_2 \in D_n$, so ist auch $v_1 v_2 \in D_n$.

[6] Walther von Dyck, 1856-1934

Satz von Chomsky-Schützenberger[7]

Jede kontextfreie Sprache ist homomorphes Bild des Durchschnitts einer regulären Sprache mit einer Dyck-Sprache:

Satz von Chomsky-Schützenberger

Eine Sprache L über Σ ist genau dann kontextfrei, wenn zu einem $n \geq 0$ ein Homomorphismus $h : \Gamma_n^* \longrightarrow \Sigma^*$ so existiert, dass

$$L = h(D_n \cap R),$$

wobei R eine reguläre Sprache über Γ_n bezeichnet.

$L = \{a^n b^n \mid n \geq 0\}$ ist kontextfrei, da

$$L = h(D_1 \cap \{(\}^*\{)\}^*)$$

wobei $h : \{(,)\}^* \longrightarrow \{a, b\}^*$ mit $h(() = a, \quad h()) = b$

[7] Marcel-Paul Schützenberger, 1920 - 1996

Abschlusseigenschaften kontextfreier Sprachen

Kontextfreie Sprachen sind **nicht** abgeschlossen unter $\cap$.

$$L_1 = \{a^m b^m c^n \mid m, n \geq 0\} \qquad L_2 = \{a^m b^n c^n \mid m, n \geq 0\}$$

L_1, L_2 sind kontextfrei:

$$\begin{array}{lcl} S_1 & \to & S_1\, c \mid A \\ A & \to & a\, A\, b \mid \varepsilon \end{array} \qquad \begin{array}{lcl} S_2 & \to & a\, S_2 \mid B \\ B & \to & b\, B\, c \mid \varepsilon \end{array}$$

Nun gilt

$$L = L_1 \cap L_2 = \{a^m b^m c^m \mid m \geq 0\}$$

L ist aber **nicht** kontextfrei!

□

Abschlusseigenschaften kontextfreier Sprachen

Kontextfreie Sprachen sind **nicht** abgeschlossen unter Komplement

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

Wir haben gesehen, dass kontextfreie Sprachen unter $\cup$ abgeschlossen sind. Wären diese auch unter Komplement abgeschlossen, so – wegen obiger Beziehung – auch unter Durchschnitt. Wir haben aber gezeigt, dass dies nicht der Fall ist.

$\square$

Pumping Lemma für kontextfreie Sprachen

Pumping Lemma für kontextfreie Sprachen

Sei L eine unendliche kontextfreie Sprache. Dann existiert eine (nur von L abhängige) Schranke $m > 0$ so, dass für jedes Wort $w \in L$ mit $|w| \geq m$ Wörter u, v, x, y, z so existieren, dass

$$w = uvxyz \qquad \text{mit } |vxy| \leq m \text{ und } |vy| \geq 1$$

sowie

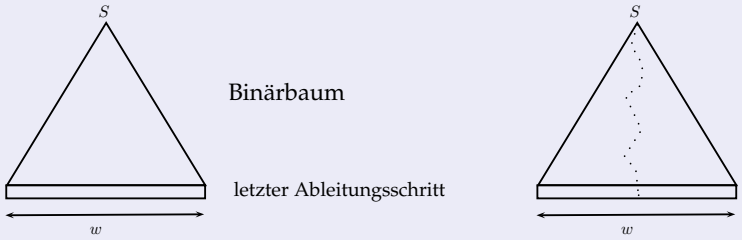
$$w_i = uv^ixy^iz \in L \qquad \text{für alle } i \geq 0.$$

Beweis Idee:
Während man das Pumping Lema für reguläre Sprachen mit der Anzahl der Zustände entlang eines Pfades in einem DEA begründet, kann man hier in ähnlicher Weise mit der Anzahl der Nonterminale entlang von Pfaden in den binären Ableitungsbäumen einer kontextfreien Grammatik (in Chomsky Normalform) argumentieren.

Pumping Lemma für kontextfreie Sprachen - Beweis

Beweis

Sei G eine Grammatik in Chomsky Normalform mit k Nonterminalen. Wähle $m = 2^k$. Betrachte nun den Ableitungsbaum eines beliebigen Wortes $w \in L(G)$ mit $|w| \geq m$. Dieser ist (bis auf den letzten Ableitungsschritt) ein Binärbaum:

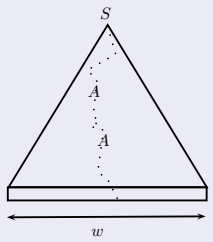


Dieser Binärbaum hat $\geq 2^k$ Blätter. Daher muss es mindestens einen Pfad der Länge $\geq k$ geben. Wir halten einen solchen Pfad fest.

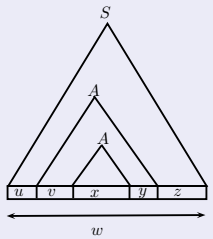
Pumping Lemma für kontextfreie Sprachen - Beweis ctd.

Beweis ctd.

Einschließlich des Startsymbols befinden sich auf diesem Pfad $\geq k + 1$ Nonterminale. Da die Grammatik G nur k verschiedene Nonterminale besitzt, muss mindestens eines davon doppelt vorkommen (Schubfachprinzip!).



Wir betrachten nun die Teilwörter, die aus den beiden Nonterminalen A abgeleitet werden können. Diese induzieren eine Zerlegung von w in dei Teilwörter $uvxyz$.



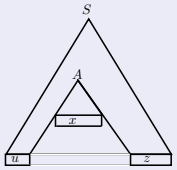
Da das obere Nonterminal A mittels einer Produktion der Form $A \to BC$ weiter abgeleitet wird, ist (sind) v oder y (oder beide) nicht leer.

Pumping Lemma für kontextfreie Sprachen - Beweis ctd.

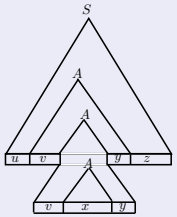
Beweis ctd.

Der Ableitungsbaum für w kann auf verschiedene Arten modifiziert werden:

Beispielsweise kann an das obere A der Ableitungsbaum des unteren A gehängt werden.
Wir erhalten damit eine Ableitung von $uxz = uv^0xy^0z$. Das heißt, $uv^0xy^0z \in L(G)$.



Man kann auch an das untere A den Teilbaum anhängen, der am oberen A beginnt. Dies ergibt eine Ableitung von $uvvxyyz = uv^2xy^2z$.



Allgemein gilt, dass für jedes $i \geq 0$ $uv^ixy^iz \in L(G)$. □

Pumping Lemma für k.f. Sprachen - Beipiel

$L = \{a^nb^nc^n \mid n \geq 0\}$ ist nicht kontextfrei

Beweis: indirekt. Angenommen L ist kontextfrei. Sei dann m die Konstante des Pumping Lemmas. Wähle z.B.

$$w = a^mb^mc^m$$

Für Wörter $w \in L$ gilt: $|w|_a = |w|_b = |w|_c$.

Da $|w| = 3m > m$ kann w zerlegt werden in $w = uvxyz$ mit $|vxy| \leq m$.

Daraus folgt, dass vxy nicht gleichzeitig Symbole a und Symbole c enthalten kann, d.h., entweder ist $|vxy|_a = 0$ oder $|vxy|_c = 0$.
(In der Tat liegen die Symbole a und c zu weit auseinander.)

Pumping Lemma für k.f. Sprachen - Beipiel ctd.

$$w = a^mb^mc^m, \quad w = uvxyz, \quad |vxy| \leq m, \quad |vy| \geq 1.$$

Fallunterscheidung:

Fall (a): $|vxy|_c = 0$.

Wegen $|vy| \geq 1$ gilt $|vy|_a + |vy|_b > 0$. Wähle $i = 0$, d.h.,

$$w_0 = uv^0xy^0z = uxz$$

(Nach dem Pumping Lemma müsste $w_0 \in L$ gelten.)

Für uxz gilt nun
$|uxz|_c = |uvxyz|_c = |w|_c$, aber $|uxz|_a < |w|_a$ oder $|uxz|_b < |w|_b$.
Das heißt, es gilt *nicht*
$|uxz|_a = |uxz|_b = |uxz|_c$ und daher $uxz \notin L$. **Widerspruch!**

Pumping Lemma für k.f. Sprachen - Beipiel ctd.

$$w = a^mb^mc^m, \quad w = uvxyz, \quad |vxy| \leq m, \quad |vy| \geq 1.$$

Fallunterscheidung ctd.:

Fall (b): $|vxy|_c > 0$ und daher $|vxy|_a = 0$.

Wegen $|vy| \geq 1$ gilt $|vy|_b + |vy|_c > 0$.

Für $w_0 = uxz$ gilt nun aber
$|uxz|_a = |uvxyz|_a = |w|_a$, jedoch $|uxz|_b < |w|_b$ oder $|uxz|_c < |w|_c$.
Das heißt, es gilt *nicht*
$|uxz|_a = |uxz|_b = |uxz|_c$ und daher $uxz \notin L$. **Widerspruch!**

Wir haben alle möglichen Zerlegungen von z untersucht, aber in jedem Fall einen Widerspruch erhalten! Somit kann L nicht kontextfrei sein. □

Korollar zum P.L. für kontextfreie Sprachen

Folgerung [Korollar zum P.L. für kontextfreie Sprachen]

Sei $L \subseteq \{a\}^*$, sodass $L = \{a^{f(n)} \mid n \geq 0\}$ für eine streng monoton wachsende Funktion f in den natürlichen Zahlen.
Gibt es für jede natürliche Zahl k eine natürliche Zahl $n(k)$, sodass

$$f(n(k)+1) - f(n(k)) \geq k,$$

dann kann L nicht kontextfrei sein.

Nach obigem Satz können die Sprachen

$$\{a^p \mid p \text{ ist eine Primzahl}\} \quad \text{und}$$
$$\{a^{2^n} \mid n \geq 0\}$$

nicht kontextfrei sein.

Kontextfreie Sprachen über einem einelementigen Alphabet

Bemerkung: Für kontextfreie Sprachen über einem einelementigen Alphabet gilt jedoch der folgende Satz:

Jede kontextfreie Sprache über einem einelementigen Alphabet ist regulär.

Jenseits der Kontextfreiheit

Formale Sprachen

$\{a^n b^n c^n \mid n \geq 0\}$ ist nicht kontextfrei.

Programmiersprachen

Typen- und Deklarationsbedinungen sind nicht oder nur schwer kontextfrei darstellbar.

Natürliche Sprachen

Schweizer Dialekt:

Jan säit das *mer d'chind em Hans es huus haend wele laa hälfe aastriiche.*

Hochdeutsch:
Jan sagt, dass *wir die Kinder dem Hans helfen lassen wollten, das Haus anzustreichen.*

Monotone Grammatiken

Monotone Grammatik

Eine Grammatik heißt *monoton*, wenn für alle Produktionen $\alpha \to \beta$ die Länge von α kleiner gleich der Länge von β ist ($\alpha \neq \varepsilon$).
Kommt das Startsymbol S auf keiner rechten Seite vor, ist auch $S \to \varepsilon$ zugelassen.

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

$G = \langle \{S, T, C\}, \{a, b, c\}, P, S \rangle$

wobei P folgende Produktionen enthält:

$$\begin{array}{lcl} S & \to & \varepsilon \mid abc \mid a\,T\,bc \\ T & \to & a\,T\,b\,C \mid ab\,C \end{array} \qquad \begin{array}{lcl} C\,b & \to & b\,C \\ C\,c & \to & cc \end{array}$$

Es gilt: $\mathcal{L}(G) = L$.

Kontextsensitive Grammatiken

Kontextsensitive Grammatik

Eine Grammatik heißt *kontextsensitiv*, wenn alle Produktionen die Form $uAv \to u\beta v$ besitzen, wobei $u, v \in (N \cup T)^*$, $A \in N$ und $\beta \in (N \cup T)^+$.
Kommt Startvariable S auf keiner rechten Seite vor, ist auch $S \to \varepsilon$ zugelassen.

Für jede Sprache, die von einer monotonen Grammatik erzeugt wird, gibt es auch eine kontextsensitive Grammatik, und umgekehrt.

Kontextsensitive Grammatiken: Beispiel

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

$G = \langle \{S, T, B, C, X, Y\}, \{a, b, c\}, P, S \rangle$

wobei P folgende Produktionen enthält:

$$\begin{array}{rcl} S & \to & \varepsilon \mid abc \mid aTBc \\ T & \to & aTBC \mid abC \\ B & \to & b \\ Cc & \to & cc \end{array} \qquad \begin{array}{rcl} CB & \to & CY \\ CY & \to & XY \\ XY & \to & XC \\ XC & \to & BC \end{array}$$

Es gilt: $\mathcal{L}(G) = L$.

Rekursive Sprachen

Rekursive Sprachen

Eine Sprache $L \in \mathcal{L}_0(\Sigma)$ heißt *rekursiv*, wenn auch das Komplement der Sprache $\overline{L} = \Sigma^* - L$, eine rekursiv aufzählbare Sprache ist, d.h., $\overline{L} \in \mathcal{L}_0(\Sigma)$. Die Menge der rekursiven Sprachen aus $\mathcal{L}_0(\Sigma)$ wird mit $\mathcal{L}_{rec}(\Sigma)$ bezeichnet, die Familie der rekursiven Sprachen mit $\mathcal{L}_{rec}$.

Chomsky-Hierarchie

Sprachfamilie	Grammatiktyp	Maschinenmodell
$\mathcal{L}_0$ (rekursiv aufzählb.)	unbeschränkt	Turingmaschine (= EA + RAM)
$\mathcal{L}_1$ (kontextsensitiv)	kontextsensitiv monoton	Linear beschränkter Aut. (= EA + beschr.RAM)
$\mathcal{L}_2$ (kontextfrei)	kontextfrei	Kellerautomat (= EA + Stack)
$\mathcal{L}_3$ (regulär)	regulär	endlicher Automat (EA)

Chomsky Hierarchie

$$\mathcal{L}_3 \subsetneq \mathcal{L}_2 \subsetneq \mathcal{L}_1 \subsetneq \mathcal{L}_{rec} \subsetneq \mathcal{L}_0.$$

Abschlusseigenschaften: Homomorphismen

Seien Σ und Γ zwei Alphabete. **Homomorphismus**: $h\colon \Sigma \to \Gamma^*$
Induktive Erweiterung auf Wörter über Σ:

1. $h(\varepsilon) = \varepsilon$;
2. $h(wa) = h(w)h(a)$ für alle $w \in \Sigma^*$ und $a \in \Sigma$.

Anwendung von h auf jedes Wort der Sprache L:

$$h(L) = \{h(w) \mid w \in L\}$$

Ein Homomorphismus h heißt ε-**frei**, wenn $h(a) \neq \varepsilon$ für alle $a \in \Sigma$.

Abschluss unter Homomorphismen

Die Sprachfamilien $\mathcal{L}_i$, $i \in \{0, 2, 3\}$, sind gegenüber beliebigen Homomorphismen abgeschlossen; die Familie der monotonen (kontext-sensitiven) Sprachen ist nur gegenüber ε-freien Homomorphismen abgeschlossen.

Abschlusseigenschaften: inverse Homomorphismen

Ist $h : \Sigma^* \to \Gamma^*$ ein Homomorphismus, so ist der **inverse Homomorphismus** h^{-1} für eine beliebige Sprache L definiert als

$$h^{-1}(L) := \{w \in \Sigma^* \mid h(w) \in L\}$$

Abschluss unter inversen Homomorphismen

Die Sprachfamilien $\mathcal{L}_i$, $i \in \{0, 1, 2, 3\}$, sind gegenüber inversen Homomorphismen abgeschlossen.

Homomorphismen: Beispiel

$L = \{a^{m^4} b^{mn} c^{mn} a^n d^{mn} e^m \mid n \geq 1\}$ nicht kontextfrei

Wir zeigen indirekt, dass L nicht kontextfrei sein kann:

Sei h der Homomorphismus mit

$$h : \{a, b, c, d, e\}^* \to \{a, b, c\}^* \text{ sowie}$$

$$h(a) = \varepsilon, \quad h(b) = a, \quad h(c) = b, \quad h(d) = c, \text{ und } h(e) = \varepsilon,$$

d.h., $h(L) = \{a^{mn} b^{mn} c^{mn} \mid n \geq 1\}$.

Die Familie der kontextfreien Sprachen ist gegenüber beliebigen Homomorphismen abgeschlossen, d.h. auch $h(L)$ müsste daher kontextfrei sein. $h(L)$ ist aber nicht kontextfrei, demnach kann auch L keine kontextfreie Sprache sein.

gsm-Abbildungen

gsm (generalized sequential machine): endlicher Automat mit Ausgabe

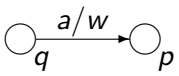
Eine *gsm* ist ein Sechstupel $M = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$, wobei

Q Zustände, Σ Eingabealphabet, Γ Ausgabealphabet,

$\delta : Q \times \Sigma \to Q \times \Gamma^*$.

q_0 Startzustand, F Endzustände

Bedeutung von $\delta(q, a) = (p, w)$: Beim Übergang von q nach p wird Symbol a gelesen und Wort w ausgegeben:



gsm-Abbildungen

gsm-Abbildung $f_M : \Sigma^* \to \mathcal{P}(\Gamma^*)$
$f_M(w)$ definiert durch alle Ausgabewörter v, die sich bei Analyse des Eingabewortes w auf einem Pfad vom Startknoten q_0 zu einem Endknoten aus F ergeben.
Erweiterung auf Sprachen $L \subseteq \Sigma^*$:

$$f_M(L) := \{v \in \Gamma^* \mid v \in f_M(w) \text{ für ein } w \in L\}$$

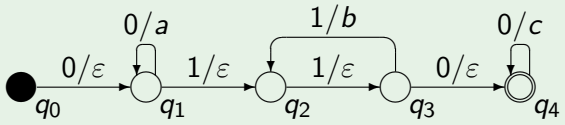
M und f_M heißen *ε-frei*, wenn $\delta : Q \times \Sigma \to Q \times \Gamma^+$.
M und f_M heißen *deterministisch*, wenn für jedes Paar (q, a) höchstens ein $(q, a; p, v) \in \delta$.

gsm-Abbildungen: Beispiel

$\{0^n 1^{2n} 0^n \mid n \geq 2\}$ nicht kontextfrei

Anmerkung: $\{a^n b^n c^n \mid n \geq 1\}$ ist bekanntermaßen nicht kontextfrei.

Beweis indirekt. Angenommen $L = \{0^n 1^{2n} 0^n \mid n \geq 2\}$ ist kf. Sei dann $M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{a, b, c\}, \delta, q_0, \{q_4\})$ die (deterministische) *gsm* mit
$\delta(q_0, 0) = (q_1, \varepsilon)$, $\delta(q_1, 0) = (q_1, a)$, $\delta(q_1, 1) = (q_2, \varepsilon)$,
$\delta(q_2, 1) = (q_3, \varepsilon)$, $\delta(q_3, 1) = (q_2, b)$, $\delta(q_3, 0) = (q_4, \varepsilon)$,
$\delta(q_4, 0) = (q_4, c)$.



Da die Familie der kontextfreien Sprachen gegenüber beliebigen gsm-Abbildungen abgeschlossen ist, müsste auch $M(L) = \{a^n b^n c^n \mid n \geq 1\}$ kontextfrei sein, was aber nicht der Fall ist. Widerspruch! Somit kann auch L nicht kontextfrei sein.

Abschlusseigenschaften von Sprachfamilien

	$\mathcal{L}_3$	$\mathcal{L}_2$	$\mathcal{L}_1$	$\mathcal{L}_0$
Vereinigung	ja	ja	ja	ja
Konkatenation	ja	ja	ja	ja
Kleenescher Stern	ja	ja	ja	ja
Komplement	ja	nein	ja	nein
Durchschnitt	ja	nein	ja	ja
Durchschnitt mit reg. Mengen	ja	ja	ja	ja
Homomorphismen	ja	ja	nein	ja
ε-freie Homomorphismen	ja	ja	ja	ja
inverse Homomorphismen	ja	ja	ja	ja
gsm-Abbildungen	ja	ja	nein	ja
ε-freie *gsm*-Abbildungen	ja	ja	ja	ja

Zeitkomplexität

Zeitkomplexität

Sei M eine deterministische Turingmaschine (DTM), die eine Sprache **entscheidet**. Die Laufzeit oder Zeitkomplexität von M ist die Abbildung $f : \mathbb{N} \longrightarrow \mathbb{N}$, wobei $f(n)$ die maximale Anzahl von Rechenschritten ist, die M benötigt, um auf einem Input der Länge n zu halten.

Die exakte Laufzeit eines Algorithmus ist oft ein komplexer Ausdruck. Eine "angenehmere" Abschätzung bietet die asymptotische Analyse, bei der nur die Terme der höchsten Ordnung betrachtet werden:

Asymptotische Notation

$h(n) = O(f(n))$, falls es Konstanten c und n_0 gibt, sodass $h(n) \leq c \cdot f(n)$ für alle $n \geq n_0$ gilt.

Sei $f(n) = 5n^3 + 2n^2 + 22n + 6$. Dann ist $f(n) = O(n^3)$

Die Komplexitätsklasse **P**

Die Klasse P

$\mathbf{P} = \{L \mid L$ ist in polynomiell beschränkter Zeit von einer DTM entscheidbar$\}$

Lässt man polynomielle Laufzeit-Unterschiede ausser Acht ("vernachlässigbar") so ist **P** invariant für alle Berechnungsmodelle, die polynomiell äquivalent zu DTM mit einem Band sind.

Die Klasse **P** umfasst genau jene Probleme, für die effiziente Algorithmen existieren.

Die Komplexitätsklasse **NP**

DTM ... Determinisitische Turingmaschine
NTM ... Nichtdetermininistische Turingmaschine

Die Klasse NP

$\mathbf{NP} = \{L \mid L$ ist in polynomiell beschränkter Zeit von einer NTM entscheidbar$\}$

Zu jeder Sprache $L \in \mathbf{NP}$ gibt es eine DTM, die L in höchstens exponentieller Zeit ($\rightarrow$ Klasse **EXPTIME**) entscheidet.

$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXPTIME}$

Beispiel: SATISFIABILITY (SAT)

SAT

Gegeben:	aussagenlogische Formel α mit Aussagenvariablen $x_1, x_2, \ldots, x_n$
Gefragt:	Ist α erfüllbar?

SAT: Beispiel

- Sei $\alpha = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$. Für diese Instanz von SAT ist die Antwort "ja", da die Formel mit der Belegung $I(x_1) = 1$ und $I(x_2) = 1$ zu *true* evaluiert.
- Für die Instanz $\alpha = x_1 \wedge \neg x_1$ hingegen ist die Antwort "nein". (Es gibt keine Belegung, die α wahr macht.)

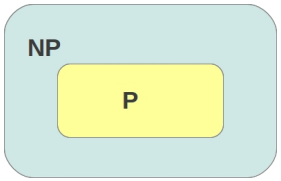
P versus **NP**

P ... Klasse von Sprachen, die in polynomieller Zeit entschieden werden können.
NP... Klasse von Sprachen, die in polynomieller Zeit verifiziert werden können.

Bisher konnte von keiner Sprache gezeigt werden, dass sie in **NP**, aber nicht in **P** liegt.

Dementsprechend ist die Frage, ob die Inklusion $\mathbf{P} \subseteq \mathbf{NP}$ echt ist (d.h., ob $\mathbf{P} = \mathbf{NP}$ gilt oder nicht) eines der bekanntesten offenen Probleme der Theoretischen Informatik und Mathematik.

Mutmaßliche Beziehung zwischen **P** und **NP**:



Polynomielle Reduktion

Polynomielle Reduktion

Seien $A, B \subseteq \Sigma^*$ Mengen. Dann heißt A auf B polynomiell reduzierbar (in Zeichen $A \leq_p B$) falls es eine totale und mit polynomieller Komplexität berechenbare Funktion $f : \Sigma^* \longrightarrow \Sigma^*$ gibt, so dass für alle $a \in \Sigma^*$ gilt:

$$x \in A \Longleftrightarrow f(x) \in B.$$

Polynomielle Reduktion

Gilt $A \leq_p B$ und $B \in \mathbf{NP}$, so ist auch $A \in \mathbf{NP}$.

Gilt $A \leq_p B$ und $B \in \mathbf{P}$, so ist auch $A \in \mathbf{P}$.

Beweis

Sei $A \leq_p B$ und $B \in \mathbf{P}$. Dann existieren Polynome $p(n)$ und $q(n)$ sowie Turingmaschinen M und N mit folgenden Eigenschaften:

- M berechnet aus einer Eingabe $w \in \Sigma^*$ in Zeit $p(|w|)$ ein Wort $f(w)$ mit $w \in A \Longleftrightarrow f(w) \in B$
  (Beachte: Da M in $p(|w|)$ Schritten nur eine Ausgabe der Länge höchstens $p(|w|)$ erzeugen kann, gilt $|f(w)| \leq p(|w|)$)
- N akzeptiert die Sprache B in Zeit $q(n)$

Polynomielle Reduktion

Beweis ctd.

Ein Turingmaschine für die Sprache A arbeitet dann bei einer Eingabe w wie folgt:

1. Berechne $f(w)$ (Zeitaufwand: $p(|w|)$)
2. Simuliere M auf $f(w)$ (Zeitaufwand: $q(p(|w|))$)

Der Gesamte Zeitaufwand ist also $p(|w|) + q(p(|w|))$, was wiederum ein Polynom ist.
Die Aussage für die Klasse **NP** kann analog bewiesen werden

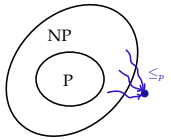
□

NP-harte und **NP**-vollständige Probleme

Die Komplexität einiger Probleme in **NP** spiegelt jene der gesamten Klasse wieder. Würde ein Polynomialzeit-Algorithmus für eines dieser Probleme existieren, wären alle Probleme in **NP** in polynomieller Zeit lösbar. Solche Probleme heißen **NP**-vollständig.

NP-hart

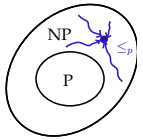
Ein Problem A heißt **NP**-hart genau dann, wenn $A' \leq_p A$ für alle Mengen $A' \in \mathbf{NP}$ gilt.



NP-vollständig

Ein Problem A heißt **NP**-vollständig genau dann, wenn

- A **NP**-hart ist und
- $A \in \mathbf{NP}$ gilt.



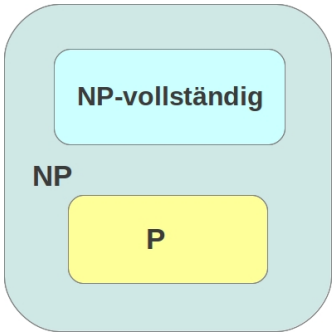
sofern $\mathbf{P} \neq \mathbf{NP}$

Informell: Wenn A **NP**-hart ist, dann ist es mindestens so schwer zu lösen wie irgendein Problem in **NP**. Ist A **NP**-vollständig, dann ist es eines der schwierigsten Probleme in **NP**.

P, NP, NP-vollständig

Sei A **NP**-vollständig. Dan gilt: $A \in \mathbf{P}$ genau dann wenn $\mathbf{P} = \mathbf{NP}$

Mutmaßliche Beziehung zwischen **P**, **NP** und **NP**-vollständig:



Beweis der NP-vollständigkeit

Seien A und B Sprachen aus **NP** mit $A \leq_p B$, und sei A **NP**-vollständig. Dann ist auch B **NP**-vollständig.

Damit erhalten wir folgendes Verfahren, um die **NP**-Vollständigkeit eines Problems B nachzuweisen:

1. Zeige, dass B in **NP** ist.
2. Zeige, dass irgendein Problem, dessen **NP**-Vollständigkeit bekannt ist, auf B polynomiell reduzierbar ist.

3-SAT ist NP-vollständig

3-SAT

Gegeben:	aussagenlogische Formel α in konjunktiver Normalform mit höchstens 3 Variablen je Klausel
Gefragt:	Ist die Formel α erfüllbar?

3-SAT ist **NP**-vollständig.

Beweis

Wie SAT ist auch 3-SAT in **NP**. Für die **NP**-Härte reicht es, SAT $\leq_p$ 3-SAT zu zeigen (da SAT **NP**-hart ist). Wir müssen also in polynomieller Zeit aus einer beliebigen Formel F eine Formel F' in KNF mit höchstens drei Literalen pro Klausel so bestimmen, dass

$$F \text{ erfüllbar} \iff F' \text{ erfüllbar}$$

3-SAT ist NP-vollständig - ctd.

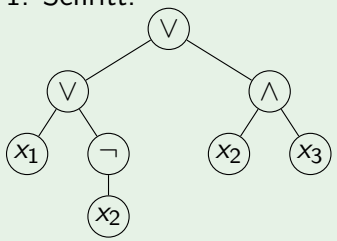
Beweis ctd.

1. Schritt: Betrachte Formel F als Baum, dessen innere Knoten mit den Operatoren $\wedge, \vee$ und $\neg$ und dessen Blätter mit Variablen beschriftet sind.

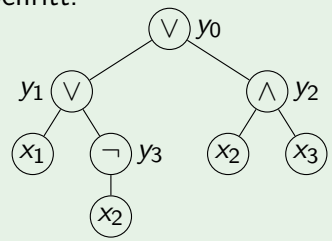
2. Schritt: Ordne jedem inneren Knoten eine neue Variable $y_0, y_1, y_2, \ldots$ zu. Der Wurzel wird y_0 zugeordnet.

Beispiel: $F = (x_1 \vee \neg x_2) \vee (x_2 \wedge x_3)$

1. Schritt:



2. Schritt:



3-SAT ist **NP**-vollständig

Beweis ctd.

3. Schritt:
für alle inneren Knoten bilden wir eine Formel und betrachten deren Konjunktion (zusammen mit der Formel y_0):

- ist y_i die Beschriftung eines inneren Knotens mit Operator $\neg$ und u jene des Kindes, so betrachte Formel $y_i \leftrightarrow \neg u$
- ist y_i die Beschriftung eines inneren Knotens mit binärem Operator $\circ$ und sind u, v die Beschriftungen der Kinder, so betrachte Formel $y_i \leftrightarrow u \circ v$

Beachte: die so konstruierte Formel ist genau dann erfüllbar, wenn F erfüllbar ist.

Beispiel ctd.

In unserem Beispiel ergibt sich:
$(y_0 \leftrightarrow (y_1 \vee y_2)) \wedge (y_1 \leftrightarrow (x_1 \vee y_3)) \wedge (y_2 \leftrightarrow (x_2 \wedge x_3)) \wedge (y_3 \leftrightarrow \neg x_2) \wedge y_0$

3-SAT ist **NP**-vollständig

Beweis ctd.

4. Schritt: Forme diese Formel in die verlangte KNF mit maximal drei Literalen pro Klausel um:

$$\begin{aligned} y \leftrightarrow (u \vee v) &\equiv (\neg y \vee u \vee v) \wedge (\neg(u \vee v) \vee y) \\ &\equiv (\neg y \vee u \vee v) \wedge ((\neg u \wedge \neg v) \vee y) \\ &\equiv (\neg y \vee u \vee v) \wedge (\neg u \vee y) \wedge (\neg v \vee y) \\ &\quad \text{(Distributivgesetz!)} \end{aligned}$$

Analog:

$$\begin{aligned} y \leftrightarrow (u \wedge v) &\equiv (\neg y \vee u) \wedge (\neg y \vee v) \wedge (\neg u \vee \neg v \vee y) \text{ sowie} \\ y \leftrightarrow (\neg z) &\equiv (\neg y \vee \neg z) \wedge (y \vee z) \end{aligned}$$

Damit erhält man eine Formel F', die genau dann erfüllbar ist, wenn F erfüllbar ist. Außerdem werden alle Umformungsschritte mit nur polynomialem Aufwand durchgeführt. □

SAT, 3-SAT, 2-SAT

Die Probleme SAT und 3-SAT sind **NP**-vollständig.

Also: Formeln in KNF mit höchstens drei Literalen pro Klausel enthalten "die volle Schwierigkeit des Erfüllbarkeitsproblems" SAT.

Was ist aber mit noch einfacher aussehenden Formeln?
z.B. 2-SAT: Formeln in KNF, die höchstens zwei Literale pro Klausel enthalten?

2-SAT $\in$ P

Mit Hilfe des *Resolutionsverfahrens* ist in Polynomialzeit entscheidbar, ob eine Formel F in KNF mit höchstens zwei Literalen pro Klausel erfüllbar ist.

Die Grenze zwischen "einfachen" und "schwierigen" Formeln liegt also zwischen Formeln in KNF mit höchstens zwei bzw. höchstens drei Literalen pro Klausel.

CLIQUE ist **NP**-vollständig

CLIQUE ist **NP**-vollständig

Beweis

CLIQUE $\in$ **NP**: raten und prüfen

CLIQUE ist **NP**-hart: 3-SAT $\leq_p$ CLIQUE

- Sei α in KNF, wobei jede Klausel genau drei Literale enthält
- In polynomieller Zeit muss eine Zahl k und ein Graph $G = (V, E)$ so konstruiert werden, dass gilt:

$$\alpha \text{ ist erfüllbar} \iff G \text{ enthält eine } k\text{−Clique.}$$

Einige **NP**-vollständige Probleme

DIRECTED HAMILTONIAN PATH (DHC)

Gegeben: Gerichteter Graph $G = (V, E)$ mit Knotenmenge V und Kantenmenge $E \subseteq V \times V$

Gefragt: Besitzt G einen Hamiltonkreis, d.h. kann man den Graphen so durchlaufen, daß jeder Knoten genau einmal besucht wird?

TRAVELING SALES PERSON (TSP)

Gegeben: Schranke $M \in \mathbb{N}$, n Städte, $n \times n$-Kostenmatrix C mit $C_{i,j}$ = Kosten einer Fahrt von Stadt i nach Stadt j

Gefragt: Gibt es eine Rundreise durch alle Städte mit Kosten von höchstens M?

Einige weitere **NP**-vollständige Probleme

k-Färbbarkeit von Graphen

Gegeben: Ungerichteter Graph $G = (V, E)$

Gefragt: Gibt es Zuordnung von k verschiedenen Farben zu Knoten in V so, dass keine zwei benachbarten Knoten v_1, v_2 dieselbe Farbe haben?

Ein Graph ist genau dann 2-färbbar, wenn er bipartit ist. Für $k = 2$ ist das Problem daher in **P**, für alle $k \geq 3$ ist es **NP**- vollständig (Reduktion von 3-SAT).

Effiziente Lösungen des Färbbarkeitsproblems sind relevant für die Lösung von Scheduling-Problemen (z.B. Konstruktion von Stundenplänen).

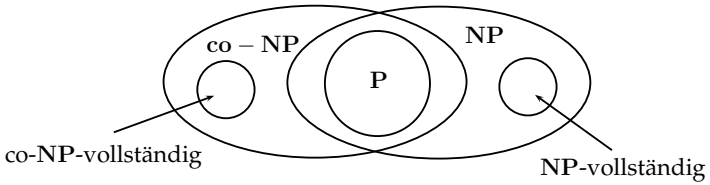
Co-NP

Co-NP

Co-**NP** enthält genau die Sprachen, deren Komplement in **NP** ist.

- **P** ist abgeschlossen unter Komplement. (Akzeptieren/Verwerfen einer DTM ist (bei rekursiven Sprachen) vertauschbar)
- Also: **P** $\subseteq$ Co-**NP**
- Wenn **P** = **NP**, dann **P** = **NP** = Co-**NP**

Vermutete Beziehung:



Die Klassen **PSPACE** und **NPSPACE**

Platzkomplexität

Sei M eine deterministische Turingmaschine (DTM), die eine Sprache **entscheidet**. Die Platzkomplexität von M ist die Abbildung $f : \mathbb{N} \longrightarrow \mathbb{N}$, wobei $f(n)$ die maximale Anzahl von Bandzellen ist, die M benötigt, um auf einem Input der Länge n zu halten.

Die Klasse PSPACE

PSPACE $= \{L \mid L$ ist von einer polynomiell platzbeschränkten DTM entscheidbar$\}$

Die Klasse NPSPACE

NPSPACE $= \{L \mid L$ ist von einer polynomiell platzbeschränkten NTM entscheidbar$\}$

Ein Problem in **PSPACE**: QBF

Quantifizierte aussagenlogische Formeln

- jede Aussagenvariable x ist eine quantifizierte aussagenlogische Formel
- sind F und G quantifizierte aussagenlogische Formeln, so auch $F \vee G$, $F \wedge G$, $\neg F$, $\forall x : F$ und $\exists x : F$ (wobei x eine Aussagenvariable ist)

QBF

Gegeben: eine quanitifizierte aussagenlogische Formel F

Gefragt: Besitzt F eine erfüllende Belegung?

QBF $\in$ PSPACE

QBF ist **PSPACE**-vollständig.

Platz- vs. Zeit Komplexität

Jede Zeitbeschränkung ist zugleich eine Platzbeschränkung (Mit n Konfigurationswechsel können höchstens $n+1$ Bandzellen besucht werden):

$$\mathbf{P} \subseteq \mathbf{PSPACE} \quad \text{und} \quad \mathbf{NP} \subseteq \mathbf{NPSPACE}$$

Offensichtlich gilt auch:

$$\mathbf{PSPACE} \subseteq \mathbf{NPSPACE}$$

Jedes Problem, dass sich nichtdeterministisch mit einem Polynomialzeitalgorithmus lösen lässt, ist zugleich durch ein polynomiell platzbeschränktes deterministisches Verfahren lösbar. Polynomielle Platzbeschränkung wiederum hat eine exponentielle Laufzeitbeschränkung zur Folge:

$$\mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXPTIME}$$

Satz von Savitch

Satz von Savitch[4] (1970)

$\mathbf{PSPACE} = \mathbf{NPSPACE}$

Insgesamt erhalten wir:

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} = \mathbf{NPSPACE} \subseteq \mathbf{EXPTIME}$$

Es ist nur bekannt, dass mindestens eine diese Inklusionen echt ist, da:

$$\mathbf{P} \neq \mathbf{EXPTIME}$$

Es wird jedoch vermutet, dass alle obigen Inklusionen echt sind.

[4] Walter J. Savitch (* 1943)