Übersichtsfolien zur Übung

Theoretische Informatik Sommersemester 2014

Markus Kaiser

4. Juli 2014

Organisatorisches

- Markus Kaiser
  - Mail: tutor@zfix.org
  - Web: theo.zfix.org
  - Hg: tutor.zfix.org

- Meine Übungen
  - Gruppe 03: Montag, 16:05 - 17:35, 03.11.018
  - Gruppe 15: Donnerstag, 10:15 - 11:45, 00.08.038
- Hausaufgaben
  - Abgabe am Mittwoch, 10h
  - Rückgabe in der Übung
  - Kein Notenbonus
  - Trotzdem machen!
- Klausur
  - Endterm: Do 24.07. 11-14h
  - Wiederholung: Do 25.09. 11-14h

Was ist Theoinf?

Aus der VL

- Automatentheorie
  - Rechner mit endlichem oder kellerartigem Speicher
- Grammatiken
  - Syntax von Programmiersprachen
- Berechenbarkeitstheorie
  - Untersuchung der Grenzen, was Rechner prinzipiell können
- Komplexitätstheorie
  - Untersuchung der Grenzen, was Rechner mit begrenzten Ressourcen können

Definition

- Ein Alphabet Σ ist eine endliche Menge.
- Ein Wort über Σ ist eine endliche Folge von Zeichen.
- Eine Teilmenge $L \subseteq \Sigma^*$ ist eine formale Sprache

Definition (Operationen auf Sprachen)

- $AB := \{uv \mid u \in A \land v \in B\}$
- $A^{n+1} := A^n A,$ $\qquad A^0 := \{\epsilon\}$
- $A^* := \bigcup_{n \in \mathbb{N}_0} A^n$

Definition (Grammatik)

Eine (Phrasenstruktur-)Grammatik $G = (V, \Sigma, P, S)$ ist ein 4-Tupel:

- V endlich viele Nichtterminale (Variablen)
- Σ ein Alphabet von Terminalen
- P endlich viele Produktionen $\subseteq (V \cup \Sigma)^* \times (V \cup \Sigma)^*$
- S ein Startsymbol (Axiom)

Ist $(l, r) \in P$, so schreibt man $l \to r$.

Beispiel

$\Sigma = \{0, 1\}$. Grammatik für alle Wörter ungerader Länge, bei denen alle Nullen vor der ersten Eins stehen und weniger Nullen als Einsen vorhanden sind.

Definition (Grammatik)

Eine (Phrasenstruktur-)Grammatik $G = (V, \Sigma, P, S)$ ist ein 4-Tupel:

- V endlich viele Nichtterminale (Variablen)
- Σ ein Alphabet von Terminalen
- P endlich viele Produktionen $\subseteq (V \cup \Sigma)^* \times (V \cup \Sigma)^*$
- S ein Startsymbol (Axiom)

Ist $(l, r) \in P$, so schreibt man $l \to r$.

Beispiel

$\Sigma = \{0, 1\}$. Grammatik für alle Wörter ungerader Länge, bei denen alle Nullen vor der ersten Eins stehen und weniger Nullen als Einsen vorhanden sind.

$$S \to 0S1 \mid S11 \mid 1$$

Sei $G = (V, \Sigma, P, S)$ eine Grammatik und $\alpha \to \beta \in P$ beliebig.

Definition (Monotonie)

G heißt (längen-)monoton, wenn für $\alpha \neq S$ gilt

$$|\alpha| \leq |\beta|$$

und falls $S \to \epsilon \in P$, dann kommt S nie auf der rechten Seite vor.

Definition (Chomsky-Typen)

Seien $A \in V$, $\gamma, \delta \in (V \cup \Sigma)^*$ und $\beta' \in (V \cup \Sigma)^+$.
Damit G vom Typ k ist, muss für α und β gelten

	α	β
Typ 0	beliebig	beliebig
Typ 1	$= \gamma A \delta$	$= \gamma \beta' \delta$
Typ 2	$\in V$	beliebig
Typ 3	$\in V$	$\in \Sigma^+ \cup \Sigma^* V$

Ab Typ 1 muss G auch monoton sein.

Alle formalen Sprachen

Typ 0 - Rekursiv aufzählbar
Grammatik
Turingmaschine, WHILE-Programm, μ-rekursive Funktion

Typ 1 - Kontextsensitiv
Längenmonotone Grammatik
Linear Beschränkter Automat (LBA)

Typ 2 - Kontextfrei
Links nur ein Nichtterminal
Kellerautomat (PDA)

Typ 3 - Regulär
Links- / Rechtsreguläre Grammatik
DFA, NFA, RE

Definition (Intuitive Berechenbarkeit)

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt intuitiv berechenbar, wenn es einen Algorithmus gibt, der bei Eingabe $(n_1, \dots, n_k) \in \mathbb{N}^k$

- nach endlich vielen Schritten mit Ergebnis $f(n_1, \dots, n_k)$ hält, falls $f(\dots)$ definiert ist,
- und nicht terminiert, falls $f(\dots)$ nicht definiert ist.

Churchsche These (nicht beweisbar)

Turing-Maschinen können genau alle intuitiv berechenbaren Funktionen berechnen.

Definition (Entscheidbarkeit)

Eine Menge A heißt entscheidbar gdw ihre charakteristische Funktion

$$\chi_A(x) := \begin{cases} 1 & \text{falls } x \in A \\ 0 & \text{falls } x \notin A \end{cases}$$

berechenbar ist.

Definition (Semi-Entscheidbarkeit)

Eine Menge A heißt semi-entscheidbar gdw

$$\chi'_A(x) := \begin{cases} 1 & \text{falls } x \in A \\ \perp & \text{falls } x \notin A \end{cases}$$

berechenbar ist.

Definition (kontextfreie Linksableitung)

Eine Ableitung

$$S \rightarrow^* xAz \rightarrow x\beta z \rightarrow^* w$$

heißt (kontextfreie) Linksableitung, wenn für jede Anwendung jeder Produktion $A \rightarrow \beta$ gilt, dass in x kein Nichtterminal vorkommt.

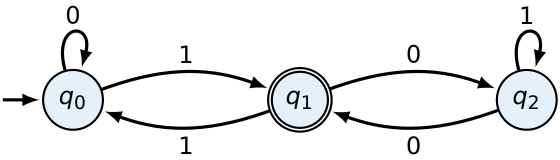
Definition (Eindeutigkeit)

- Eine Grammatik heißt eindeutig, wenn es für jedes Wort genau eine Linksableitung gibt.
- Eine Sprache heißt eindeutig, wenn es für sie eine eindeutige Grammatik gibt.

Definition (Deterministischer endlicher Automat)

Ein DFA ist ein Tupel $M = (Q, \Sigma, \delta, q_0, F)$ aus einer/einem

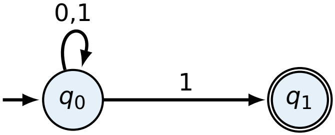
- endlichen Menge von Zuständen Q
- endlichen Eingabealphabet Σ
- totalen Übergangsfunktion $\delta : Q \times \Sigma \to Q$
- Startzustand $q_0 \in Q$
- Menge von Endzuständen $F \subseteq Q$



Definition (Nicht-Deterministischer endlicher Automat)

Ein NFA ist ein Tupel $N = (Q, \Sigma, \delta, S, F)$ mit

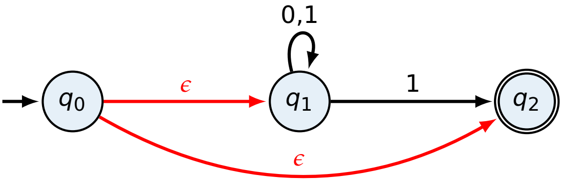
- Q, Σ, F wie ein DFA
- Menge von Startzuständen $S \subseteq F$
- Übergangsfunktion $\delta : Q \times \Sigma \to \mathcal{P}(Q)$



Definition (NFA mit ϵ-Übergängen)

Ein ϵ-NFA ist ein Tupel $N = (Q, \Sigma, \delta, S, F)$ mit

- Q, Σ, F wie ein DFA
- Menge von Startzuständen $S \subseteq F$
- Übergangsfunktion $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)$



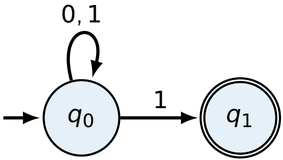
Potenzmengenkonstruktion

Konstruiere einen Automaten, der alle möglichen Pfade gleichzeitig berücksichtigt. Gegeben ein NFA $(Q, \Sigma, \delta, S, F)$, konstruiere einen DFA mit Zuständen aus $\mathcal{P}(Q)$.

- Starte in $\{S\}$
- Die Übergangsfunktion speichert alle möglichen Schritte

$$\bar{\delta} : \mathcal{P}(Q) \times \Sigma \rightarrow \mathcal{P}(Q)$$
$$(M, a) \mapsto \bigcup_{q \in M} \delta(q, a)$$

- M ist Endzustand wenn $F \cap M \neq \emptyset$



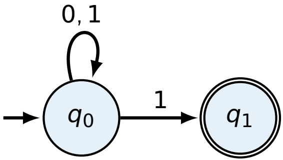
Potenzmengenkonstruktion

Konstruiere einen Automaten, der alle möglichen Pfade gleichzeitig berücksichtigt. Gegeben ein NFA $(Q, \Sigma, \delta, S, F)$, konstruiere einen DFA mit Zuständen aus $\mathcal{P}(Q)$.

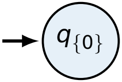
- Starte in $\{S\}$
- Die Übergangsfunktion speichert alle möglichen Schritte

$$\overline{\delta} : \mathcal{P}(Q) \times \Sigma \to \mathcal{P}(Q)$$
$$(M, a) \mapsto \bigcup_{q \in M} \delta(q, a)$$

- M ist Endzustand wenn $F \cap M \neq \emptyset$



$\rightarrow$



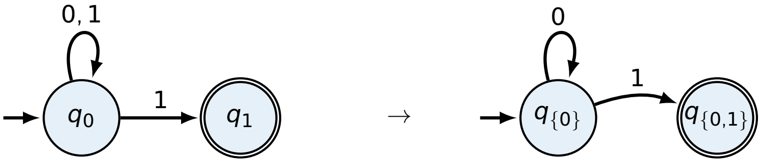
Potenzmengenkonstruktion

Konstruiere einen Automaten, der alle möglichen Pfade gleichzeitig berücksichtigt. Gegeben ein NFA $(Q, \Sigma, \delta, S, F)$, konstruiere einen DFA mit Zuständen aus $\mathcal{P}(Q)$.

- Starte in $\{S\}$
- Die Übergangsfunktion speichert alle möglichen Schritte

$$\begin{aligned}\overline{\delta} : \mathcal{P}(Q) \times \Sigma &\to \mathcal{P}(Q) \\ (M, a) &\mapsto \bigcup_{q \in M} \delta(q, a)\end{aligned}$$

- M ist Endzustand wenn $F \cap M \neq \emptyset$



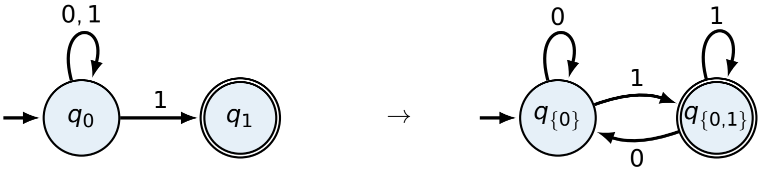
Potenzmengenkonstruktion

Konstruiere einen Automaten, der alle möglichen Pfade gleichzeitig berücksichtigt. Gegeben ein NFA $(Q, \Sigma, \delta, S, F)$, konstruiere einen DFA mit Zuständen aus $\mathcal{P}(Q)$.

- Starte in $\{S\}$
- Die Übergangsfunktion speichert alle möglichen Schritte

$$\overline{\delta} : \mathcal{P}(Q) \times \Sigma \to \mathcal{P}(Q)$$
$$(M, a) \mapsto \bigcup_{q \in M} \delta(q, a)$$

- M ist Endzustand wenn $F \cap M \neq \emptyset$



Definition (Regulärer Ausdruck)

Reguläre Ausdrücke sind induktiv definiert

- $\emptyset$ ist ein regulärer Ausdruck
- ϵ ist ein regulärer Ausdruck
- Für alle $a \in \Sigma$ ist a ein regulärer Ausdruck
- Sind α und β reguläre Ausdrücke, dann auch

  Konkatenation $\alpha\beta$
  Veroderung $\alpha \mid \beta$
  Wiederholung α^*

Analoge Sprachdefinition, z.b. $L(\alpha\beta) = L(\alpha)L(\beta)$

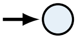
Beispiel

- $\alpha = (0|1)^*00$
- Worte bestehen aus einer beliebigen Folge von Einsen und Nullen gefolgt von zwei Nullen.
- $L(\alpha) \supseteq \{x \mid x \text{ Binärzahl}, x \mod 4 = 0\}$

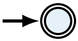
Thompson-Konstruktion

Für einen Ausdruck γ wird rekursiv mit struktureller Induktion ein ϵ-NFA konstruiert.

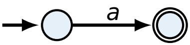
$\gamma = \emptyset$



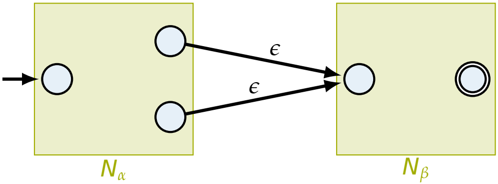
$\gamma = \epsilon$



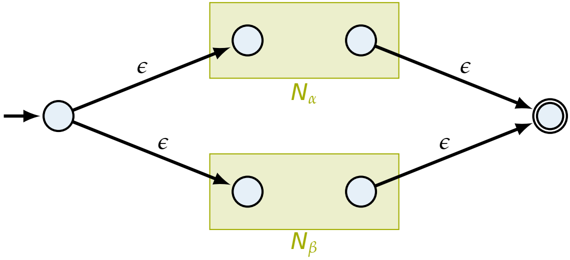
$\gamma = a \in \Sigma$



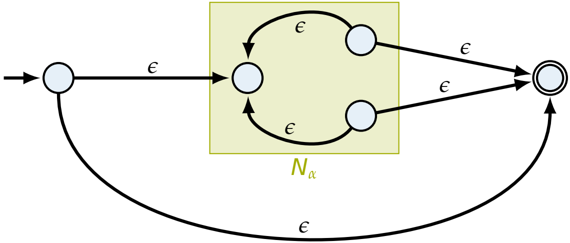
$\gamma = \alpha\beta$



$\gamma = \alpha \mid \beta$



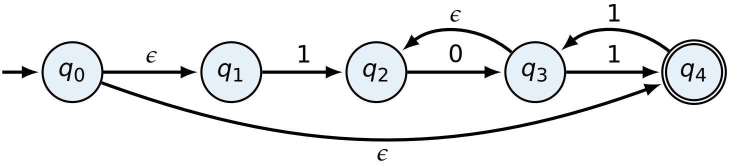
$\gamma = \alpha^*$



Idee

Entferne ϵ-Kanten durch das Bilden von ϵ-Hüllen.

1. Entferne unnötige Knoten.
2. Für jeden Pfad der Form $\epsilon \ldots \epsilon a \epsilon \ldots \epsilon$ verbinde Anfangs- und Endknoten mit einer a-Kante.
3. Entferne alle ϵ-Kanten und unerreichbare Knoten.
4. Wurde das leere Wort akzeptiert mache den Anfangszustand zum Endzustand.

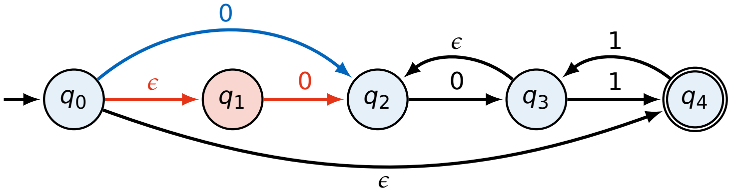


ϵ-NFA $\rightarrow$ NFA

Idee

Entferne ϵ-Kanten durch das Bilden von ϵ-Hüllen.

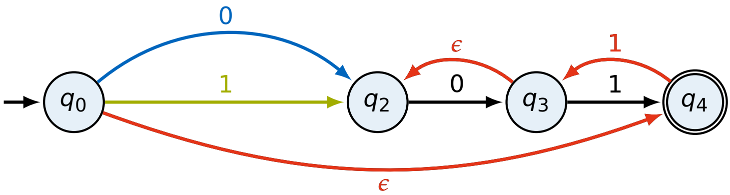
1. Entferne unnötige Knoten.
2. Für jeden Pfad der Form $\epsilon \ldots \epsilon a \epsilon \ldots \epsilon$ verbinde Anfangs- und Endknoten mit einer a-Kante.
3. Entferne alle ϵ-Kanten und unerreichbare Knoten.
4. Wurde das leere Wort akzeptiert mache den Anfangszustand zum Endzustand.



Idee

Entferne ϵ-Kanten durch das Bilden von ϵ-Hüllen.

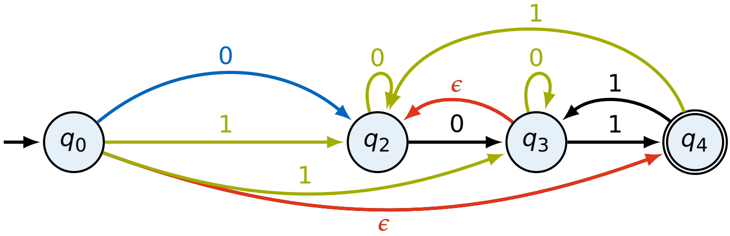
1. Entferne unnötige Knoten.
2. Für jeden Pfad der Form $\epsilon \ldots \epsilon a \epsilon \ldots \epsilon$ verbinde Anfangs- und Endknoten mit einer a-Kante.
3. Entferne alle ϵ-Kanten und unerreichbare Knoten.
4. Wurde das leere Wort akzeptiert mache den Anfangszustand zum Endzustand.



Idee

Entferne ϵ-Kanten durch das Bilden von ϵ-Hüllen.

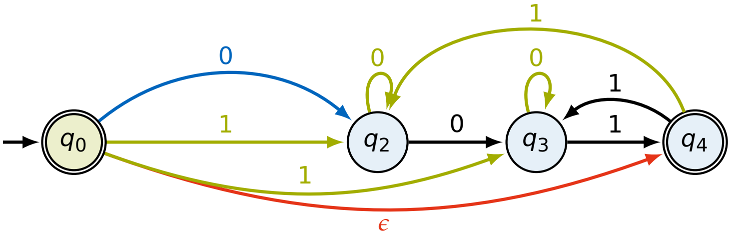
1. Entferne unnötige Knoten.
2. Für jeden Pfad der Form $\epsilon \dots \epsilon a \epsilon \dots \epsilon$ verbinde Anfangs- und Endknoten mit einer a-Kante.
3. Entferne alle ϵ-Kanten und unerreichbare Knoten.
4. Wurde das leere Wort akzeptiert mache den Anfangszustand zum Endzustand.



Idee

Entferne ϵ-Kanten durch das Bilden von ϵ-Hüllen.

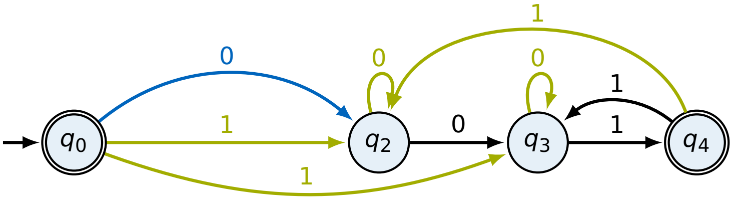
1. Entferne unnötige Knoten.
2. Für jeden Pfad der Form $\epsilon \dots \epsilon a \epsilon \dots \epsilon$ verbinde Anfangs- und Endknoten mit einer a-Kante.
3. Entferne alle ϵ-Kanten und unerreichbare Knoten.
4. Wurde das leere Wort akzeptiert mache den Anfangszustand zum Endzustand.



Idee

Entferne ϵ-Kanten durch das Bilden von ϵ-Hüllen.

1. Entferne unnötige Knoten.
2. Für jeden Pfad der Form $\epsilon \ldots \epsilon a \epsilon \ldots \epsilon$ verbinde Anfangs- und Endknoten mit einer a-Kante.
3. Entferne alle ϵ-Kanten und unerreichbare Knoten.
4. Wurde das leere Wort akzeptiert mache den Anfangszustand zum Endzustand.



Definition (Äquivalente Worte)

Jede Sprache $L \subseteq \Sigma^*$ induziert eine Äquivalenzrelation $\equiv_L \subseteq \Sigma^* \times \Sigma^*$

$$u \equiv_L v \iff (\forall w \in \Sigma^*. uw \in L \Leftrightarrow vw \in L)$$

Definition (Äquivalente Zustände)

Zwei Zustände im DFA A sind äquivalent wenn sie die selbe Sprache akzeptieren.

$$p \equiv_A q \iff (\forall w \in \Sigma^*. \hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F)$$

Definition (Äquivalente Worte)

Jede Sprache $L \subseteq \Sigma^*$ induziert eine Äquivalenzrelation $\equiv_L \subseteq \Sigma^* \times \Sigma^*$

$$u \equiv_L v \iff (\forall w \in \Sigma^*. uw \in L \Leftrightarrow vw \in L)$$

Definition (Äquivalente Zustände)

Zwei Zustände im DFA A sind äquivalent wenn sie die selbe Sprache akzeptieren.

$$p \equiv_A q \iff (\forall w \in \Sigma^*. \hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F)$$

Unterscheidbare Zustände

Definition (Unterscheidbarkeit)

Zwei Zustände sind unterscheidbar, wenn sie unterschiedliche Sprachen akzeptieren.

$$p \not\equiv_A q \iff \left(\exists w \in \Sigma^*.\hat{\delta}(p,w) \in F \wedge \hat{\delta}(q,w) \notin F\right)$$

Satz

Sind $\delta(p,a)$ und $\delta(q,a)$ unterscheidbar, dann auch p und q.

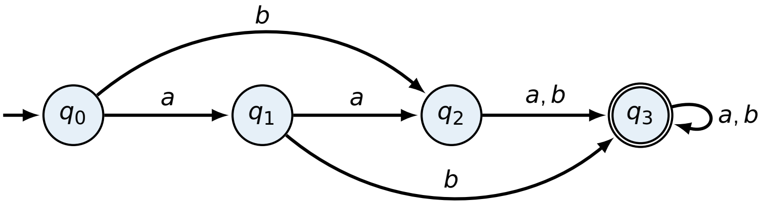
Definition (Unterscheidbarkeit)

Zwei Zustände sind unterscheidbar, wenn sie unterschiedliche Sprachen akzeptieren.

$$p \not\equiv_A q \iff \left(\exists w \in \Sigma^*. \hat{\delta}(p, w) \in F \wedge \hat{\delta}(q, w) \notin F\right)$$

Satz

Sind $\delta(p, a)$ und $\delta(q, a)$ unterscheidbar, dann auch p und q.



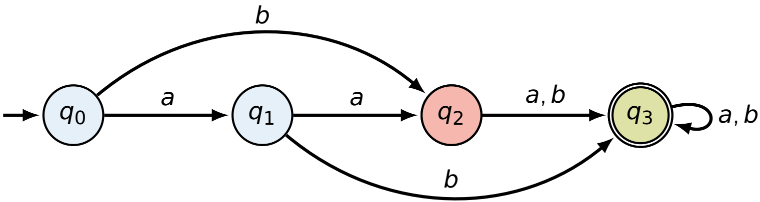
Definition (Unterscheidbarkeit)

Zwei Zustände sind unterscheidbar, wenn sie unterschiedliche Sprachen akzeptieren.

$$p \not\equiv_A q \iff \left(\exists w \in \Sigma^*. \hat{\delta}(p, w) \in F \land \hat{\delta}(q, w) \notin F\right)$$

Satz

Sind $\delta(p, a)$ und $\delta(q, a)$ unterscheidbar, dann auch p und q.



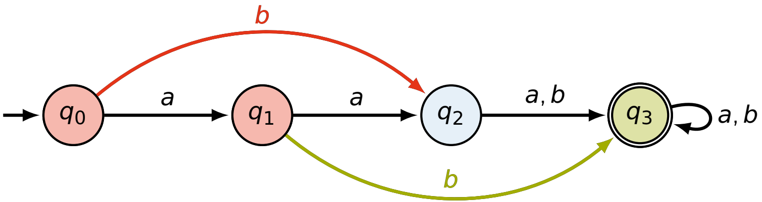
Definition (Unterscheidbarkeit)

Zwei Zustände sind unterscheidbar, wenn sie unterschiedliche Sprachen akzeptieren.

$$p \not\equiv_A q \iff \left(\exists w \in \Sigma^*. \hat{\delta}(p, w) \in F \wedge \hat{\delta}(q, w) \notin F\right)$$

Satz

Sind $\delta(p, a)$ und $\delta(q, a)$ unterscheidbar, dann auch p und q.

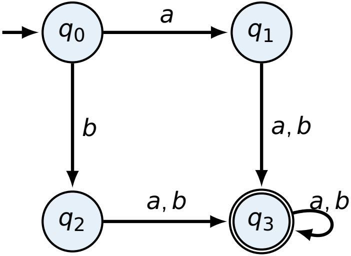


Quotientenautomat

1. Entferne alle von q_0 nicht erreichbaren Zustände
2. Berechne die unterscheidbaren Zustände
3. Kollabiere die äquivalenten Zustände

Quotientenautomat

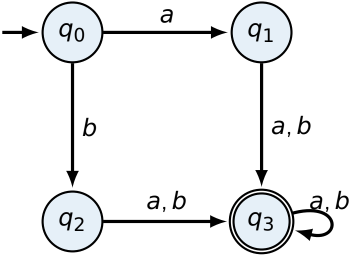
1. Entferne alle von q_0 nicht erreichbaren Zustände
2. Berechne die unterscheidbaren Zustände
3. Kollabiere die äquivalenten Zustände



Quotientenautomat

1. Entferne alle von q_0 nicht erreichbaren Zustände
2. Berechne die unterscheidbaren Zustände
3. Kollabiere die äquivalenten Zustände

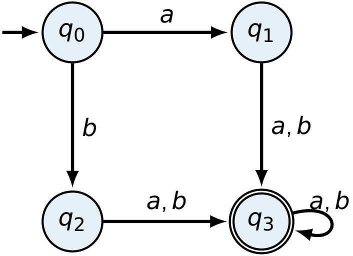
	0	1	2
1			
2			
3			



Quotientenautomat

1. Entferne alle von q_0 nicht erreichbaren Zustände
2. Berechne die unterscheidbaren Zustände
3. Kollabiere die äquivalenten Zustände

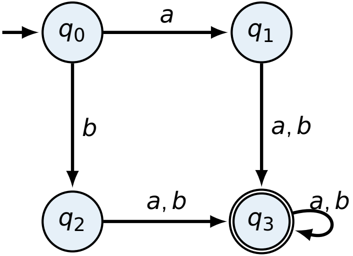
	0	1	2
1			
2			
3	×	×	×



Quotientenautomat

1. Entferne alle von q_0 nicht erreichbaren Zustände
2. Berechne die unterscheidbaren Zustände
3. Kollabiere die äquivalenten Zustände

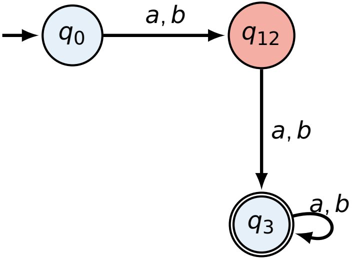
	0	1	2
1	$1/a$		
2	$1/a$		
3	×	×	×



Quotientenautomat

1. Entferne alle von q_0 nicht erreichbaren Zustände
2. Berechne die unterscheidbaren Zustände
3. Kollabiere die äquivalenten Zustände

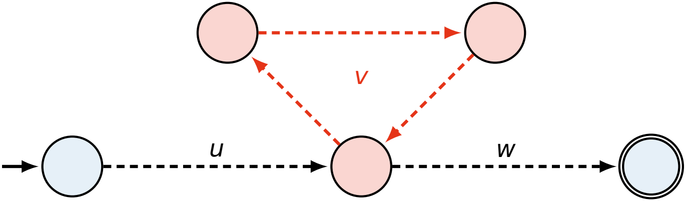
	0	1	2
1	$1/a$		
2	$1/a$		
3	×	×	×



Satz (Pumping Lemma für reguläre Sprachen)

Sei $R \subseteq \Sigma^*$ regulär.
Dann gibt es ein $n > 0$, so dass sich jedes $z \in R$ mit $|z| \geq n$ so in $z = uvw$ zerlegen lässt, dass

- $v \neq \epsilon$
- $|uv| \leq n$
- $\forall i \geq 0. uv^i w \in R$



Definition

Sei $G = (V, \Sigma, P, S)$ eine CFG.
Ein Symbol $X \in V \cup \Sigma$ ist

nützlich	es gibt $S \to_G^* w \in \Sigma^*$ in der X vorkommt
erzeugend	es gibt $X \to_G^* w \in \Sigma^*$
erreichbar	es gibt $S \to_G^* \alpha X \beta$

Satz

Nützliche Symbole sind erzeugend und erreichbar. Aber nicht notwendigerweise umgekehrt.

$$S \to AB \mid a, \quad A \to b$$

Definition (Chomsky-Normalform)

Eine kontextfreie Grammatik ist in Chomsky-Normalform (CNF) genau dann wenn alle Produktionen die Form

$$A \to a \quad \text{oder} \quad A \to BC$$

haben.

Satz

Zu jeder CFG G existiert eine CFG G' in Chomsky-Normalform mit

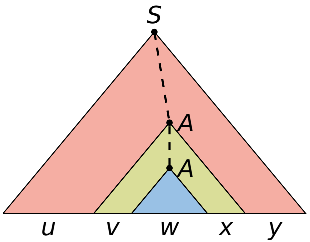
$$L(G') = L(G) \setminus \{\epsilon\}$$

Satz (Pumping Lemma für kontextfreie Sprachen)

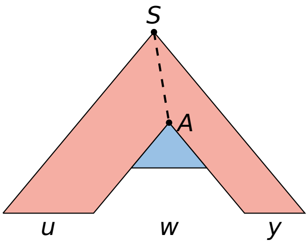
Sei $L \subseteq \Sigma^*$ kontextfrei.
Dann gibt es ein $n > 0$, so dass sich jedes $z \in L$ mit $|z| \geq n$ so in $z = uvwxy$ zerlegen lässt, dass

- $vx \neq \epsilon$
- $|vwx| \leq n$
- $\forall i \geq 0. uv^iwx^iy \in L$

$S \to^* uAy \to^* uvAxy \to^* uvwxy$



$S \to^* uAy \to^* uwy$



CNF Konstruktion

Sei $G = (V, \Sigma, P, S)$ eine CFG.

1. Eliminiere ϵ-Produktionen
2. Eliminiere Kettenproduktionen
3. Ersetze Terminale durch Nichtterminale
4. Verkürze Ketten von Nichtterminalen der Länge ≥ 3

CNF Konstruktion

Sei $G = (V, \Sigma, P, S)$ eine CFG.

1. Eliminiere ϵ-Produktionen
2. Eliminiere Kettenproduktionen
3. Ersetze Terminale durch Nichtterminale
4. Verkürze Ketten von Nichtterminalen der Länge ≥ 3

Sind $B \rightarrow \epsilon$ und $A \rightarrow \alpha B \beta$ in P, dann füge $A \rightarrow \alpha\beta$ hinzu. Entferne danach alle ϵ-Produktionen.

$$S \rightarrow Ab, \quad A \rightarrow aAA \mid \epsilon$$

wird zu:

$$S \rightarrow Ab \mid b$$
$$A \rightarrow aAA \mid aA \mid a$$

CNF Konstruktion

Sei $G = (V, \Sigma, P, S)$ eine CFG.

1. Eliminiere ϵ-Produktionen
2. Eliminiere Kettenproduktionen
3. Ersetze Terminale durch Nichtterminale
4. Verkürze Ketten von Nichtterminalen der Länge ≥ 3

Sind $A \rightarrow B$ und $B \rightarrow \alpha$ in P, dann füge $A \rightarrow \alpha$ hinzu. Entferne danach alle Kettenproduktionen und unerreichbaren Symbole.

$$S \rightarrow A, \quad A \rightarrow a \mid B, \quad B \rightarrow bS$$

wird zu:

$$A \rightarrow a \mid bS$$
$$S \rightarrow a \mid bS$$

CNF Konstruktion

Sei $G = (V, \Sigma, P, S)$ eine CFG.

1. Eliminiere ϵ-Produktionen
2. Eliminiere Kettenproduktionen
3. Ersetze Terminale durch Nichtterminale
4. Verkürze Ketten von Nichtterminalen der Länge ≥ 3

Ersetze jedes $a \in \Sigma$ in einer rechten Seite länger als 1 durch ein neues Nichtterminal.

$$S \to aa \mid Bb \mid b, \quad B \to \ldots$$

wird zu:

$$S \to X_a X_a \mid B X_b \mid b$$
$$X_a \to a, \quad X_b \to b$$

CNF Konstruktion

Sei $G = (V, \Sigma, P, S)$ eine CFG.

1. Eliminiere ϵ-Produktionen
2. Eliminiere Kettenproduktionen
3. Ersetze Terminale durch Nichtterminale
4. Verkürze Ketten von Nichtterminalen der Länge ≥ 3

Ersetze jede Produktion der Form $A \to B_1 B_2 \dots B_k$ durch neue Nichtterminale mit Produktionen der Länge 2.

$$S \to X_a X_b B X_a, \quad X_a \to a, \quad X_b \to b, \quad B \to \dots$$

wird zu:

$$\begin{aligned} S &\to X_a T_1 \\ T_1 &\to X_b T_2, \quad T_2 \to B X_a \end{aligned}$$

Definition (Cocke-Younger-Kasami-Algorithmus)

Der CYK-Algorithmus entscheidet das Wortproblem für kontextfreie Grammatiken in Chomsky-Normalform in $\mathcal{O}(n^3)$.
Gegeben eine Grammatik $G = (V, \Sigma, P, S)$ in CNF und ein Wort $w = a_1 \dots a_n \in \Sigma^*$. Mit

$$V_{ij} := \{A \in V \mid A \to_G^* a_i \dots a_j\}$$

ist

$$w \in L(G) \Leftrightarrow S \in V_{1n}$$

$$V_{ii} = \{A \in V \mid (A \to a_i) \in P\}$$
$$V_{ij} = \{A \in V \mid \exists k, B \in V_{ik}, C \in V_{k+1,j} \,.\, (A \to BC) \in P\}$$

CYK-Algorithmus

Kombiniere Teilwörter zum ganzen Wort, wenn möglich.

1. Initialisiere mit den V_{ii}.
2. Befülle die Tabelle von unten nach oben.

$$S \rightarrow AB \mid BC, \quad A \rightarrow BA \mid a, \quad B \rightarrow CC \mid b, \quad C \rightarrow AB \mid a$$

4				
3				
2				
1				
	b	a	a	a

CYK-Algorithmus

Kombiniere Teilwörter zum ganzen Wort, wenn möglich.

1. Initialisiere mit den V_{ii}.
2. Befülle die Tabelle von unten nach oben.

$$S \rightarrow AB \mid BC, \quad A \rightarrow BA \mid a, \quad B \rightarrow CC \mid b, \quad C \rightarrow AB \mid a$$

4				
3				
2				
1	B	A, C	A, C	A, C
	b	a	a	a

CYK-Algorithmus

Kombiniere Teilwörter zum ganzen Wort, wenn möglich.

1. Initialisiere mit den V_{ii}.
2. Befülle die Tabelle von unten nach oben.

$$S \rightarrow AB \mid BC, \quad A \rightarrow BA \mid a, \quad B \rightarrow CC \mid b, \quad C \rightarrow AB \mid a$$

4				
3				
2	A, S	B	B	
1	B	A, C	A, C	A, C
	b	a	a	a

CYK-Algorithmus

Kombiniere Teilwörter zum ganzen Wort, wenn möglich.

1. Initialisiere mit den V_{ii}.
2. Befülle die Tabelle von unten nach oben.

$$S \to AB \mid BC, \quad A \to BA \mid a, \quad B \to CC \mid b, \quad C \to AB \mid a$$

4				
3	$\emptyset$	S, A, C		
2	A, S	B	B	
1	B	A, C	A, C	A, C
	b	a	a	a

CYK-Algorithmus

Kombiniere Teilwörter zum ganzen Wort, wenn möglich.

1. Initialisiere mit den V_{ij}.
2. Befülle die Tabelle von unten nach oben.

$$S \to AB \mid BC, \quad A \to BA \mid a, \quad B \to CC \mid b, \quad C \to AB \mid a$$

4	$S, \dots$			
3	$\emptyset$	S, A, C		
2	A, S	B	B	
1	B	A, C	A, C	A, C
	b	a	a	a

Induktive Sprachdefinition

Die induktive Definition zu einer Grammatik G ergibt sich direkt aus ihren Produktionen. Dabei werden kleinere Worte zu größeren Worten zusammengesetzt, die Definition erfolgt bottom-up.

Beispiel

Mit den Produktionen $S \to 0S1 \mid S11 \mid 1$:

$$\begin{array}{rcr} & & 1 \in L_G(S) \\ u \in L_G(S) & \Longrightarrow & 0u1 \in L_G(S) \\ u \in L_G(S) & \Longrightarrow & u11 \in L_G(S) \end{array}$$

Also z.B:

$$1 \in L_G(S) \Longrightarrow 011 \in L_G(S) \Longrightarrow 01111 \in L_G(S)$$

Satz (Ogden Lemma für kontextfreie Sprachen)

Sei $L \subseteq \Sigma^*$ kontextfrei.
Dann gibt es ein $n > 0$, so dass für jedes $z \in L$ mit $|z| \geq n$ gilt:
Für jede Markierung M von mindestens n Buchstaben in z gibt es eine Zerlegung $z = uvwxy$ mit

- $|vx|_M \geq 1$
- $|vwx|_M \leq n$
- $\forall i \geq 0. uv^iwx^iy \in L$

Beispiel (Markierung)

Sei $w = abaabaaa$ ein Wort. Dann ist

$$w = a\color{blue}{ba}a\color{blue}{b}aaa$$

eine Markierung mit $|w|_M = 3$.

Definition (Greibach-Normalform)

Eine kontextfreie Grammatik ist in Greibach-Normalform (GNF) genau dann wenn alle Produktionen außer $S \to \epsilon$ die Form

$$A \to a\alpha \quad \text{mit} \quad a \in \Sigma, \alpha \in V^*$$

haben.

Satz

Zu jeder CFG G existiert eine CFG G' in Greibach-Normalform mit

$$L(G') = L(G)$$

Satz (Einsetzen von Produktionen)

Enthält eine CFG die Produktionen

$$A \to \alpha_1 B \alpha_2$$
$$B \to \beta_1 \mid \cdots \mid \beta_k$$

so ändert sich die erzeugte Sprache nicht, wenn man B in A einsetzt.

$$A \to \alpha_1 \beta_1 \alpha_2 \mid \cdots \mid \alpha_1 \beta_k \alpha_2$$

Beispiel

Die Grammatik

$$S \to a \mid aBc$$
$$B \to b \mid bS$$

ist äquivalent zur Grammatik

$$S \to a \mid abc \mid abSc$$

Definition (Linksrekursive Produktion)

Man nennt eine Produktion linksrekursiv, wenn sie die Form

$$A \to A\alpha_1 \mid \cdots \mid A\alpha_k \mid \beta_1 \mid \cdots \mid \beta_l \quad \text{mit} \quad \alpha_i, \beta_i \in (V \cup \Sigma)^+$$

hat, wobei die β_i nicht mit A beginnen.

Satz (Ersetzen von linksrekursiven Produktionen)

Sei A eine linksrekursive Produktion einer CFG.
Dann ändert sich die erzeugte Sprache nicht, wenn wir A ersetzen durch

$$A \to \beta_1 \mid \cdots \mid \beta_l \mid \beta_1 B \mid \cdots \mid \beta_l B$$
$$B \to \alpha_1 \mid \cdots \mid \alpha_k \mid \alpha_1 B \mid \cdots \mid \alpha_k B$$

B ist niemals linksrekursiv.

GNF Konstruktion

Sei $G = (V, \Sigma, P, S)$ eine CFG.

1. Nummeriere Nichtterminale
2. Mache Prduktionen aufsteigend und nicht rekursiv
3. Setze Produktionen absteigend ein

GNF Konstruktion

Sei $G = (V, \Sigma, P, S)$ eine CFG.

1. Nummeriere Nichtterminale
2. Mache Prduktionen aufsteigend und nicht rekursiv
3. Setze Produktionen absteigend ein

Benenne alle Nichtterminale beliebig um in $A_1, \ldots, A_{|V|}$.

$$S \to Ab, \quad A \to aAS \mid \epsilon$$

wird zu

$$A_1 \to A_2 b$$
$$A_2 \to aA_2A_1$$

GNF Konstruktion

Sei $G = (V, \Sigma, P, S)$ eine CFG.

1. Nummeriere Nichtterminale
2. Mache Prduktionen aufsteigend und nicht rekursiv
3. Setze Produktionen absteigend ein

Betrachte alle Produktionen $A_l \rightarrow \dots$ in aufsteigender Reihenfolge.

- Existieren Produktionen der Form $A_l \rightarrow A_r\alpha$ mit $r < l$, dann setze A_r in A_l ein.

$$A_1 \rightarrow A_2 \mid a \mid b$$
$$A_2 \rightarrow A_1 A_1$$

wird zu

$$A_1 \rightarrow A_2 \mid a \mid b$$
$$A_2 \rightarrow A_2 A_1 \mid a A_1 \mid b A_1$$

- Entferne danach alle linksrekursiven A_l-Produktionen.

GNF Konstruktion

Sei $G = (V, \Sigma, P, S)$ eine CFG.

1. Nummeriere Nichtterminale
2. Mache Prduktionen aufsteigend und nicht rekursiv
3. Setze Produktionen absteigend ein

Betrachte alle Produktionen $A_l \rightarrow \dots$ in aufsteigender Reihenfolge.

- Existieren Produktionen der Form $A_l \rightarrow A_r\alpha$ mit $r < l$, dann setze A_r in A_l ein.
- Entferne danach alle linksrekursiven A_l-Produktionen.

$$A_2 \rightarrow A_2A_1 \mid aA_1 \mid bA_1$$

wird zu

$$A_2 \rightarrow aA_1 \mid bA_1 \mid aA_1A_3 \mid bA_1A_3$$
$$A_3 \rightarrow A_1 \mid A_1A_3$$

GNF Konstruktion

Sei $G = (V, \Sigma, P, S)$ eine CFG.

1. Nummeriere Nichtterminale
2. Mache Prduktionen aufsteigend und nicht rekursiv
3. Setze Produktionen absteigend ein

Betrachte alle Produktionen $A_l \to \dots$ in absteigender Reihenfolge.

- Existieren Produktionen der Form $A_l \to A_r\alpha$ mit $r > l$, dann setze A_r in A_l ein.

$$
\begin{aligned}
A_1 &\to a \mid b \mid A_2 \\
A_2 &\to aA_1 \mid bA_1 \mid aA_1A_3 \mid bA_1A_3 \\
A_3 &\to bA_3 \mid c
\end{aligned}
$$

A_1 wird zu

$$A_1 \to a \mid b \mid aA_1 \mid bA_1 \mid aA_1A_3 \mid bA_1A_3$$

Definition (Kellerautomat)

Ein PDA (Push-Down-Automaton) ist ein Tupel $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ aus einer/einem

- endlichen Menge von Zuständen Q
- endlichen Eingabealphabet Σ
- endlichen Kelleralphabet Γ
- Übergangsfunktion $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to P(Q \times \Gamma^*)$
- Startzustand $q_0 \in Q$
- Kellerinitialisierung $Z_0 \in \Gamma$
- Menge von Endzuständen $F \subseteq Q$

$$q_i \xrightarrow{a, X / \gamma} q_j$$

Definition (Kellerautomat)

Ein PDA (Push-Down-Automaton) ist ein Tupel $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ aus einer/einem

- Übergangsfunktion $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to P(Q \times \Gamma^*)$

Definition (Akzeptanz)

Ein PDA P akzeptiert $w \in \Sigma^*$ mit Endzustand gdw

$$\exists f \in F, \gamma \in \Gamma^*. (q_0, w, Z_0) \to_P^* (f, \epsilon, \gamma)$$

Ein PDA P akzeptiert $w \in \Sigma^*$ mit leerem Keller gdw

$$\exists q \in Q. (q_0, w, Z_0) \to_P^* (q, \epsilon, \epsilon)$$

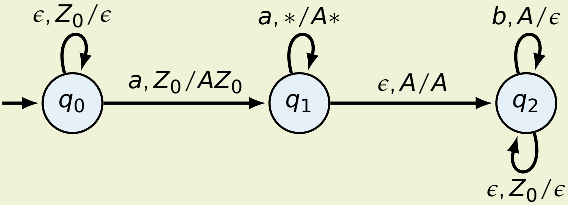
Definition (Kellerautomat)

Ein PDA (Push-Down-Automaton) ist ein Tupel $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ aus einer/einem

- Übergangsfunktion $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to P(Q \times \Gamma^*)$

Beispiel

PDA akzeptierend mit leerem Keller zu $L = \{a^n b^n \mid n \in \mathbb{N}_0\}$.



Definition (Kellerautomat)

Ein PDA (Push-Down-Automaton) ist ein Tupel $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ aus einer/einem

- Zustandsmenge Q, Eingabealphabet Σ, Kelleralphabet Γ
- Übergangsfunktion $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to P(Q \times \Gamma^*)$
- Startzustand $q_0 \in Q$, Kellerinitialisierung $Z_0 \in \Gamma$
- Menge von Endzuständen $F \subseteq Q$

Definition (Deterministischer Kellerautomat)

Ein PDA heißt deterministisch (DPDA) wenn für alle Zustände $q \in Q$, Buchstaben $a \in \Sigma$ und Kellerbuchstaben $X \in \Gamma$ gilt

$$|\delta(q, a, Z)| + |\delta(q, \epsilon, Z)| \leq 1$$

Definition (Präfixbedingung)

Eine Sprache L erfüllt die Präfixbedingung, wenn kein Wort der Sprache echtes Präfix eines anderen Wortes der Sprache ist.

$$\forall w \in L\ \forall s \in \Sigma^+ .\ ws \notin L$$

Satz

Deterministisch kontextfreie Sprachen werden genau dann von einem DPDA mit leerem Keller akzeptiert, wenn sie die Präfixbedingung erfüllen.

Definition (Parsing)

Beim Parsing wird einem Wort ein Ableitungsbaum in einer Grammatik zugeordnet, indem bottom-up die Produktionen (Reduktionen) rückwärts angewandt werden.
Es wird immer die linkestmögliche Reduktion angewandt.

Definition (Lookahead)

Ein Lookahead der Länge k legt fest, dass eine Reduktion nur dann angewandt werden darf, wenn die folgenden k Zeichen im Wort mit dem Lookahead übereinstimmen.

Beispiel

$$S \rightarrow Ac \mid Bbc$$
$$A \rightarrow ab$$
$$B \rightarrow a$$

Produktion	Lookahead
$A \rightarrow ab$	c
$B \rightarrow a$	d

Gegeben das Wort abc.
Dann darf $B \rightarrow A$ nicht angewandt werden, $A \rightarrow ab$ jedoch schon.

Definition (LR(k)-Grammatik)

Eine CFG ist eine LR(k)-Grammatik, wenn Lookaheads der Länge k genügen, um jedem Wort eine eindeutige Ableitung zuzuordnen.

Satz

Die LR(0)-Grammatiken sind eine echte Teilmenge der LR(1)-Grammatiken. Diese entspechen genau den deterministisch kontextfreien Sprachen.

$$LR(0) \subset LR(1) = LR(k \geq 1) = DCFL$$

Definition (Turingmaschine)

Eine deterministische Turingmaschine (TM) ist ein Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ aus einer/einem

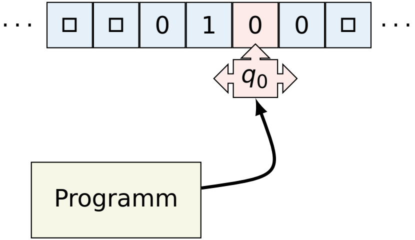
- endlichen Menge von Zuständen Q
- endlichen Eingabealphabet Σ
- endlichen Bandalphabet Γ mit $\Sigma \subset \Gamma$
- Übergangsfunktion $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, N\}$
- Startzustand $q_0 \in Q$
- Leerzeichen $\Box \in \Gamma \setminus \Sigma$
- Menge von Endzuständen $F \subseteq Q$

Definition (Turingmaschine)

Eine deterministische Turingmaschine (TM) ist ein Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ aus einer/einem

- Übergangsfunktion $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$

$$\delta(q_0, 0) = (q_1, 1, R)$$

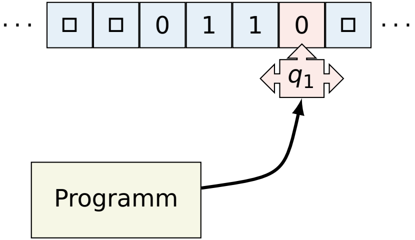


Definition (Turingmaschine)

Eine deterministische Turingmaschine (TM) ist ein Tupel
$M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ aus einer/einem

- Übergangsfunktion $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$

$$\delta(q_0, 0) = (q_1, 1, R)$$



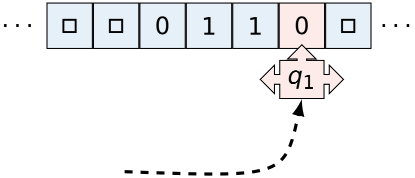
Definition (Konfiguration)

Eine Konfiguration ist ein Tripel $(\alpha, q, \beta) \in \Gamma^* \times Q \times \Gamma^*$.
Dies modelliert eine TM mit:

- Bandinhalt $\ldots\Box\alpha\beta\Box\ldots$
- Zustand q
- Kopf auf dem ersten Zeichen von $\beta\Box$

Die Startkonfiguration bei Eingabe $w \in \Sigma^*$ ist (ϵ, q_0, w).

$(011, q_1, 0)$



Definition (Konfiguration)

Eine Konfiguration ist ein Tripel $(\alpha, q, \beta) \in \Gamma^* \times Q \times \Gamma^*$.
Dies modelliert eine TM mit:

- Bandinhalt $\dots\Box\alpha\beta\Box\dots$
- Zustand q
- Kopf auf dem ersten Zeichen von $\beta\Box$

Die Startkonfiguration bei Eingabe $w \in \Sigma^*$ ist (ϵ, q_0, w).

Definition (Akzeptanz)

Eine TM M akzeptiert die Sprache

$$L(M) = \{w \in \Sigma^* \mid \exists f \in F, \alpha, \beta \in \Gamma^*.(\epsilon, q_0, w) \rightarrow_M^* (\alpha, f, \beta)\}$$

Definition (Nichtdeterministische Turingmaschine)

Eine nichtdeterministische Turingmaschine (TM) ist ein Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ aus einer/einem

- ...
- Übergangsfunktion $\delta : Q \times \Gamma \rightarrow \mathcal{P}\left(Q \times \Gamma \times \{L, R, N\}\right)$
- ...

Satz

Zu jeder nichtdeterministischen TM N gibt es eine deterministische TM M mit $L(N) = L(M)$.

Definition (Linear Beschränkte Automat)

Eine TM heißt linear beschränkt (LBA), wenn sie den Bereich des Bandes, auf dem das Eingabewort steht, niemals verlässt.

Satz

Die linear beschränkten NDTMs akzeptieren genau die Klasse der kontextsensitiven Sprachen (Typ 1).

Definition (Queue-Automat)

Ein Queue-Automat (QA) ist ein Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ die analog zu Kellerautomaten definiert sind.

- Übergangsfunktion $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow P(Q \times \Gamma^*)$

Im Gegensatz zu PDAs werden neue Symbole hinten angefügt.

Beispiel

Gegeben sei die Konfiguration $(q, a, x\alpha)$ eines Queue-Automaten und ein Schritt der Übergangsfunktion.

$$\delta(q, a, x) = (q', yz)$$

Dann ergibt sich die folgende Transition.

$$(q, a, x\alpha) \rightarrow (q', \epsilon, \alpha yz)$$

Satz

Queue-Automaten sind genauso mächtig wie Turingmaschinen.

$$\begin{tikzcd}
\text{RLG} \arrow[r, leftrightarrow] & \text{DFA} \arrow[dr] & \text{NFA} \arrow[l] & \epsilon\text{-NFA} \arrow[l] \\
 & & \text{RE} \arrow[ur] &
\end{tikzcd}$$

Satz

Für eine Darstellung D einer regulären Sprache ist entscheidbar:

Wortproblem	*Gegeben w, gilt $w \in L(D)$?*
Leerheitsproblem	*Ist $L(D) = \emptyset$?*
Endlichkeitsproblem	*Ist $\lvert L(D) \rvert < \infty$?*
Äquivalenzproblem	*Gilt $L(D_1) = L(D_2)$?*

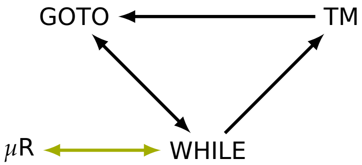
$$\begin{tikzcd}
\mathrm{CFG} \arrow[r, leftrightarrow] & \mathrm{CNF} \arrow[r] & \mathrm{GNF} \arrow[dl] \\
 & \mathrm{PDA}_\epsilon \arrow[ul] \arrow[r, leftrightarrow] & \mathrm{PDA}_F
\end{tikzcd}$$

- Abschlusseigenschaften

	Schnitt	Vereinigung	Komplement	Produkt	Stern
REG	ja	ja	ja	ja	ja
CFL	nein	ja	nein	ja	ja

- Entscheidbarkeit

	Wortproblem	Leerheit	Äquivalenz	Schnittproblem
DFA	$\mathcal{O}(n)$	ja	ja	ja
CFG	$\mathcal{O}(n^3)$	ja	nein	nein



Satz

Sei A formale Sprache, dann ist äquivalent:

- *A ist Typ 0 Sprache*
- *A rekursiv aufzählbar*
- *A semi-entscheidbar, also χ'_A berechenbar*
- *$A = L(M)$ für eine TM M*
- *A ist Bild oder Urbild einer berechenbaren Funktion*

- Abschlusseigenschaften

	Schnitt	Vereinigung	Komplement	Produkt	Stern
REG	ja	ja	ja	ja	ja
CFL	nein	ja	nein	ja	ja
CSL	ja	ja	ja	ja	ja
TM	ja	ja	nein	ja	ja

- Entscheidbarkeit

	Wortproblem	Leerheit	Äquivalenz	Schnittproblem
DFA	$\mathcal{O}(n)$	ja	ja	ja
CFG	$\mathcal{O}(n^3)$	ja	nein	nein
CSL	$\mathcal{O}(2^n)$	nein	nein	nein
TM	nein	nein	nein	nein

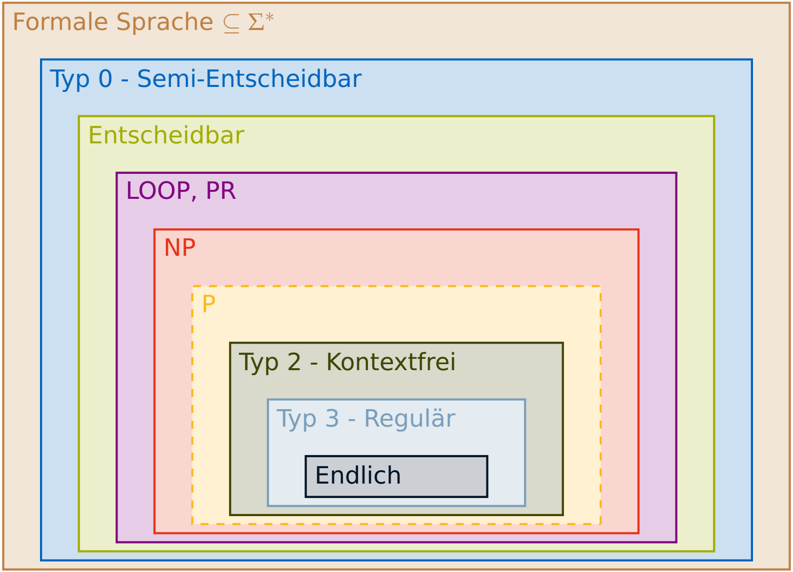
Alle formalen Sprachen

Typ 0 - Rekursiv aufzählbar
Grammatik
Turingmaschine, WHILE-Programm, μ-rekursive Funktion

Typ 1 - Kontextsensitiv
Längenmonotone Grammatik
Linear Beschränkter Automat (LBA)

Typ 2 - Kontextfrei
Links nur ein Nichtterminal
Kellerautomat (PDA)

Typ 3 - Regulär
Links- / Rechtsreguläre Grammatik
DFA, NFA, RE


Formale Sprache ⊆ Σ*
Typ 0 - Semi-Entscheidbar
Entscheidbar
LOOP, PR
NP
P
Typ 2 - Kontextfrei
Typ 3 - Regulär
Endlich

Definition (Basisfunktionen der PR)

Die Menge der primitiv rekursiven (PR) Funktionen ist induktiv definiert. Die Basisfunktionen sind die

konstante Funktion $f(x) = 0$

Nachfolgerfunktion $s(n) = n + 1$

Projektionsfunktion $\pi_i^k : \mathbb{N}^k \to \mathbb{N}, i \in [k]$

$$\pi_i^k(x_1, \ldots, x_k) = x_i$$

Definition (Komposition)

Seien g und h_i PR und $\bar{x} = (x_1, \ldots, x_n)$.
Dann ist auch die Komposition f PR.

$$f(\bar{x}) = g(h_1(\bar{x}), \ldots, h_k(\bar{x}))$$

Basisfunktionen und Komposition

Die folgenden Funktionen sind schon als primitiv rekursiv bekannt.

konstante Funktion	$f(x) = 0$
Nachfolgerfunktion	$s(n) = n + 1$
Projektionsfunktion	$\pi_i^k : \mathbb{N}^k \to \mathbb{N}, i \in [k]$
Komposition	$f(\bar{x}) = g(h_1(\bar{x}), \ldots, h_k(\bar{x}))$

Definition (Primitive Rekursion)

Das Schema der primitiven Rekursion erzeugt aus g und h die neue Funktion f.

$$f(0, \bar{x}) = g(\bar{x})$$
$$f(m+1, \bar{x}) = h(f(m, \bar{x}), m, \bar{x})$$

f ist ebenfalls primitiv rekursiv.

Definition (Erweitertes PR-Schema)

Das erweiterte Schema der primitiven Rekursion erlaubt

$$f(0, \bar{x}) = t_0$$
$$f(m+1, \bar{x}) = t$$

wobei t_0 und t Terme sind für die gilt

- t_0 enthält nur PR-Funktionen und die x_i
- t enthält nur $f(m, \bar{x})$, PR Funktionen, m und die x_i.

Satz

Das erweiterte Schema der primitiven Rekursion führt nicht aus PR heraus.

Unter anderem diese Programme sind laut Vorlesung oder Übung PR:

- $succ(x) = x + 1$
- $pred(x) = \max\{0, x - 1\}$
- $add(x, y) = x + y$
- $x \dot{-} y = \max\{0, x - y\}$
- $mult(x, y) = x \cdot y$
- $div(x, y) = x \div y$ (Ganzzahldivision)
- $pow(x, y) = x^y$
- Die restliche einfache Arithmetik...

- $tower(n) = 2^{2^{2^{\cdot^{\cdot^{\cdot}}}}}$ mit $tower(4) = 2^{16}$
- $sqr(x) = x^2$
- $sqrt(x) = \sqrt{x}$
- $c(x), p_1(x), p_2(x)$ (Cantorsche Paarungsfunktion)
- $ifthen(n, a, b) = \begin{cases} a & n \neq 0 \\ b & n = 0 \end{cases}$

Definition (LOOP-Programm)

Syntax von LOOP-Programmen.
Es ist $X \in \{x_0, x_1, \ldots\}$ und $C \in \mathbb{N}$.

$$\begin{aligned} P \to\ & X := X + C \\ \mid\ & X := X - C \\ \mid\ & P; P \\ \mid\ & \textbf{LOOP } X \textbf{ DO } P \textbf{ END} \\ \mid\ & \textbf{IF } X = 0 \textbf{ DO } P \textbf{ ELSE } Q \textbf{ END} \end{aligned}$$

- Ausgabe steht in x_0, Eingaben in $x_1, \ldots, x_n$, Rest ist 0.
- **LOOP** x_i **DO** P **END** führt P genau n mal aus, wobei n der Anfangswert von x_i ist. Zuweisungen an x_i in P ändern die Anzahl der Durchläufe nicht.

Definition (Intuitive Berechenbarkeit)

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt intuitiv berechenbar, wenn es einen Algorithmus gibt, der bei Eingabe $(n_1, \ldots, n_k) \in \mathbb{N}^k$

- nach endlich vielen Schritten mit Ergebnis $f(n_1, \ldots, n_k)$ hält, falls $f(\ldots)$ definiert ist,
- und nicht terminiert, falls $f(\ldots)$ nicht definiert ist.

Churchsche These (nicht beweisbar)

Turing-Maschinen können genau alle intuitiv berechenbaren Funktionen berechnen.

Beispiel (Berechenbarkeit)

Sind die folgenden Funktionen intuitiv berechenbar?

$$f_1(n) = \begin{cases} 1 & \text{falls } n \text{ prim} \\ 0 & \text{sonst} \end{cases}$$

$$f_2(n) = \begin{cases} 1 & \text{falls } n \text{ die ersten } n \text{ Ziffern von } \pi \text{ darstellt} \\ 0 & \text{sonst} \end{cases}$$

$$f_3(n) = \begin{cases} 1 & \text{falls in } \pi\ n \text{ Nullen am Stück vorkommen} \\ 0 & \text{sonst} \end{cases}$$

Beispiel (Berechenbarkeit)

Sind die folgenden Funktionen intuitiv berechenbar?

$$f_1(n) = \begin{cases} 1 & \text{falls } n \text{ prim} \\ 0 & \text{sonst} \end{cases}$$

$$f_2(n) = \begin{cases} 1 & \text{falls } n \text{ die ersten } n \text{ Ziffern von } \pi \text{ darstellt} \\ 0 & \text{sonst} \end{cases}$$

$$f_3(n) = \begin{cases} 1 & \text{falls in } \pi \ n \text{ Nullen am Stück vorkommen} \\ 0 & \text{sonst} \end{cases}$$

Alle drei Funktionen sind intuitiv berechenbar.

Definition (Entscheidbarkeit)

Eine Menge A heißt entscheidbar gdw ihre charakteristische Funktion

$$\chi_A(x) := \begin{cases} 1 & \text{falls } x \in A \\ 0 & \text{falls } x \notin A \end{cases}$$

berechenbar ist.

Definition (Semi-Entscheidbarkeit)

Eine Menge A heißt semi-entscheidbar gdw

$$\chi'_A(x) := \begin{cases} 1 & \text{falls } x \in A \\ \bot & \text{falls } x \notin A \end{cases}$$

berechenbar ist.

Definition (Reduzierbarkeit)

Eine Menge $A \subseteq \Sigma^*$ heißt reduzierbar auf eine Menge $B \subseteq \Gamma^*$ gdw es eine totale und berechenbare Funktion $f : \Sigma^* \to \Gamma^*$ gibt mit

$$\forall w \in \Sigma^*. w \in A \iff f(w) \in B$$

Wir schreiben dann $A \leq B$.

Intuitiv gilt:

- B ist mindestens so schwer zu lösen wie A
- Ist A unlösbar, dann auch B.
- Ist B lösbar, dann erst recht A.

Definition (Spezielles Halteproblem)

Gegeben ein Wort $w \in \{0,1\}^*$ und die durch Gödelisierung kodierte Turingmaschine M_w.
Die als spezielles Halteproblem bezeichnete Sprache K enthält alle Turingmaschinen, die bei sich selbst als Eingabe halten.

$$K := \{w \mid M_w[w] \downarrow\}$$

Satz

Das spezielle Halteproblem ist semi-entscheidbar, aber es ist nicht entscheidbar.
$\overline{K}$ ist also nicht semi-entscheidbar.

Definition (Allgemeines Halteproblem)

Gegeben Wörter $w, x \in \{0,1\}^*$ und die durch Gödelisierung kodierte Turingmaschine M_w.
Die als allgemeines Halteproblem bezeichnete Sprache H enthält alle Paare M_w und x, sodass M_w bei Eingabe x hält.

$$H := \{w\#x \mid M_w[x]\downarrow\}$$

Satz

Das allgemeine Halteproblem ist semi-entscheidbar, aber es ist nicht entscheidbar.
$\overline{H}$ ist also nicht semi-entscheidbar.

- Es ist $K \leq H$. Warum?

Definition (Rekursiv aufzählbar)

Eine Menge A heißt rekursiv aufzählbar wenn $A = \emptyset$ oder es eine berechenbare totale Funktion $f : \mathbb{N} \to A$ gibt, so dass

$$A = \{f(0), f(1), \ldots\} = \bigcup_{n \in \mathbb{N}} \{f(n)\}$$

Satz

Sei A formale Sprache, dann ist äquivalent:

- *A ist Typ 0 Sprache*
- *A rekursiv aufzählbar*
- *A semi-entscheidbar, also χ'_A berechenbar*
- *$A = L(M)$ für eine TM M*
- *A ist Bild oder Urbild einer berechenbaren Funktion*

Satz (Rice)

Sei F eine Menge berechenbarer Funktionen.
Sei weder $F = \emptyset$ noch $F = \{f \mid f \text{ berechenbar}\}$ (F nicht trivial).
Der Satz von Rice besagt, dass es dann unentscheidbar ist, ob die von einer gegebenen TM M_w berechnete Funktion in F ist.
Die formale Sprache

$$\{w \in \Sigma^* \mid \varphi_w \in F\}$$

ist unentscheidbar.

Intuitiv gilt:

- Nicht-triviale semantische Eigenschaften von Programmen sind unentscheidbar.
- Termination ist unentscheidbar.