

Einführung in die Theoretische Informatik

Zusammenfassung

Efe Kamasoglu

June 21, 2023

1 Grundbegriffe und Grammatiken

1.1 Grundbegriffe

- Alphabet Σ , endliche Menge
- Wort/String w über Σ , endliche Menge von Zeichen aus Σ
- $|w|$, Länge des Wortes w
- ϵ , das leere Wort mit der Länge 0
- Wörter u und v , uv ist ihre Konkatenation
- Wort w , w^n definiert durch:
 - $w^0 = \epsilon$
 - $w^{n+1} = ww^n$
 - Beispiel: $(ab)^3 = ababab$
- Σ^* , Menge aller Wörter über Σ
- Teilmenge $L \subseteq \Sigma^*$, formale Sprache
 - Beispiel: $\emptyset, \{\epsilon\}, L_1 = \{\epsilon, ab, aabb, aaabbb, \dots\} = \{a^n b^n \mid n \in \mathbb{N}\}$

1.2 Operationen auf Sprachen

Sprachen $A, B \subseteq \Sigma^*$

- **Konkatenation:** $AB = \{uw \mid u \in A \wedge w \in B\}$
 - Beispiel: $\{ab, b\}\{a, bb\} = \{aba, abbb, ba, bbb\}$
 - $A^n = \{w_1 \dots w_n \mid w_1, \dots, w_n \in A\} = \underbrace{A \dots A}_n$
 - $A^0 = \{\epsilon\}, A^{n+1} = AA^n$
 - $A^* = \{w_1 \dots w_n \mid n \geq 0 \wedge w_1, \dots, w_n \in A\} = \bigcup_{n \in \mathbb{N}} A^n$
 - * A^* enthält ϵ immer
 - $A^+ = AA^* = \bigcup_{n \geq 1} A^n$
 - * $\epsilon \in A$ gdw. $\epsilon \in A^+$
- **Kartesisches Produkt:** $A \times B$
 - Beispiel: $\{ab, b\} \times \{a, bb\} = \{(ab, a), (ab, bb), (b, a), (b, bb)\}$
- **Rechenregeln über Sprachen:**
 - Für alle A : $\epsilon \in A^*$
 - $\epsilon \notin \emptyset$

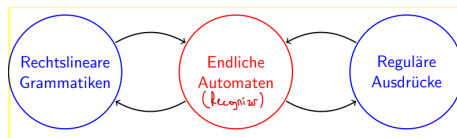
- $\emptyset^* = \{\epsilon\} = \emptyset^0$
- $A\{\epsilon\} = \{\epsilon\}A = A$
- $A\emptyset = \emptyset A = \emptyset$
- $A \times \emptyset = \emptyset$
- $A(B \cup C) = AB \cup AC$
- $(A \cup B)C = AC \cup BC$
- $A(B \cap C) = AB \cap AC$ gilt i.A. **nicht**
- $A(B \setminus C) = AB \setminus AC$ gilt i.A. **nicht**
- $A^*A^* = (A^*)^* = A^*$

1.3 Grammatiken

- Grammatik, 4-Tupel $G = (V, \Sigma, P, S)$
 - V , endliche Menge von **Nichtterminalen**
 - Σ , endliche Menge von **Terminalen**, disjunkt von V
 - $P \subseteq (V \cup \Sigma)^* \times (V \cup \Sigma)^*$, Menge von **Produktionen**
 - $S \in V$, **Startsymbol**
- Eine Grammatik G induziert eine **Ableitungsrelation** \rightarrow_G auf Wörtern über $V \cup \Sigma$:
 - $\alpha \rightarrow \alpha'$ gdw. es eine Regel $\beta \rightarrow \beta'$ in P und Wörter α_1, α_2 gibt, so dass $\alpha = \alpha_1\beta\alpha_2 \wedge \alpha' = \alpha_1\beta'\alpha_2$
- Eine **Sequenz** $\alpha_1 \rightarrow_G \alpha_2 \rightarrow_G \dots \rightarrow_G \alpha_n$ ist eine **Ableitung** von α_n aus α_1 .
- Wenn $\alpha_1 = S$ und $\alpha_n \in \Sigma^*$, dann **erzeugt** G das Wort α_n . Erzeugte Wörter bestehen nur aus **Terminalzeichen**.
- Die Sprache von G ist die Menge aller Wörter (Σ^*), die von G erzeugt werden: $L(G)$
- **Chomsky Hierarchie:** Eine Grammatik G ist vom
 - **Typ 0** immer
 - **Typ 1** falls für jede Produktion $\alpha \rightarrow \beta$ ausser $S \rightarrow \epsilon$ gilt $|\alpha| \leq |\beta|$
 - **Typ 2** falls G vom Typ 1 ist und für jede Produktion $\alpha \rightarrow \beta$ gilt $\alpha \in V$
 - **Typ 3** falls G vom Typ 2 ist und für jede Produktion $\alpha \rightarrow \beta$ ausser $S \rightarrow \epsilon$ gilt $\beta \in \Sigma \cup \Sigma V$
 - Typ 3 \subset Typ 2 \subset Typ 1 \subset Typ 0
 - $L(\text{Typ 3}) \subset L(\text{Typ 2}) \subset L(\text{Typ 1}) \subset L(\text{Typ 0})$

- **Grammatiken und Sprachklassen:**
 - **Typ 3, Rechtslineare Grammatik, Reguläre Sprachen**
 - **Typ 2, Kontextfreie Grammatik, Kontextfreie Sprachen**
 - Typ 1, Kontextsensitive Grammatik, Kontextsensitive Sprachen
 - Typ 0, Phrasenstrukturgrammatik, Rekursiv aufzählbare Sprachen

2 Reguläre Sprachen



2.1 Rechtslineare Grammatik

$X, Y \in V$, Produktionen folgender Gestalt:

- $X \rightarrow aY$
- $X \rightarrow a$
- $X \rightarrow Y$
- $X \rightarrow \epsilon$, nur dann wenn X Startsymbol ist

2.2 Deterministische endliche Automaten (DFA)

- DFA, 5-Tupel $M = (Q, \Sigma, \delta, q_0, F)$
 - Q , endliche Menge von **Zuständen**
 - Σ , endliches **Eingabealphabet**
 - $\delta : Q \times \Sigma \rightarrow Q$, totale **Übergangsfunktion**
 - $q_0 \in Q$, ein **Startzustand**
 - $F \subseteq Q$, endliche Menge von **Endzuständen**
- Die von DFA M **akzeptierte** Sprache ist $L(M) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F\}$, wobei $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$ induktiv definiert durch:
 - $\delta(q, a)$, Zustand, den man aus q mit einem **Zeichen** a erreicht
 - $\hat{\delta}(q, w)$, Zustand, den man aus q mit einem **Wort** w erreicht
 - $\hat{\delta}(q, \epsilon) = q$
 - $\hat{\delta}(q, aw) = \hat{\delta}(\delta(q, a), w)$ für $a \in \Sigma, w \in \Sigma^*$

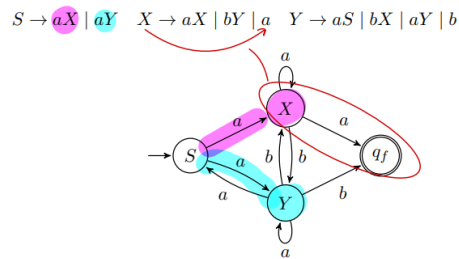
2.3 Nicht-Deterministische endliche Automaten (NFA)

- NFA, 5-Tupel $N = (Q, \Sigma, \delta, q_0, F)$
 - Q, Σ, q_0 und F wie beim DFA
 - $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$, wobei $\mathcal{P}(Q)$ Menge aller Teilmengen von Q
- Die von NFA N **akzeptierte** Sprache ist $L(N) = \{w \in \Sigma^* \mid \hat{\delta}(\{q_0\}, w) \cap F \neq \emptyset\}$, wobei
 - $\bar{\delta}(S, a) = \bigcup_{q \in S} \delta(q, a)$, Menge aller Zustände, die man von einem Zustand in S aus mit einem **Zeichen** a erreicht
 - $\hat{\delta}(S, w)$, Menge aller Zustände, die man von einem Zustand in S aus mit einem **Wort** w erreicht
 - $\bar{\delta} : \mathcal{P}(Q) \times \Sigma \rightarrow \mathcal{P}(Q)$
 - $\hat{\delta} : \mathcal{P}(Q) \times \Sigma^* \rightarrow \mathcal{P}(Q)$

2.4 Rechtslineare Grammatik \rightarrow NFA

1. Füge einen Zustand für jedes Nichtterminal, Startsymbol wird zum Startzustand
2. Füge einen Endzustand für jedes Terminal, falls es $S \rightarrow \epsilon$ gibt, dann Startzustand ist auch ein Endzustand
3. Für jede Kombination $Y \rightarrow aX$ füge eine Kante von Y nach X mit dem Zeichen a
4. Für jede Kombination $Y \rightarrow a$ füge eine Kante von Y nach dem Endzustand mit dem Zeichen a

Beispiel:

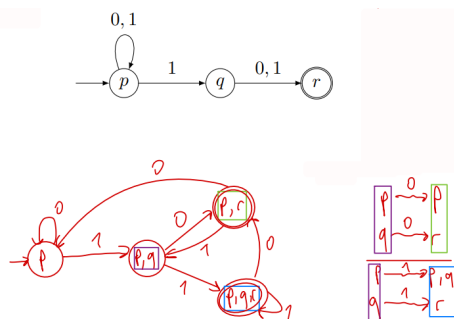


2.5 NFA \rightarrow DFA, Potenzmengenkonstruktion

Für jede NFA mit n Zuständen kann der DFA max bis zu 2^n Zustände haben.

1. Für alle Zustände wiederhole (beginnend mit Startzustand q_0):
 - (a) Bestimme wohin man mit welcher Kante geht
 - (b) Erzeuge neue Zustände durch Vereinigung der auf der rechten Seite stehenden Zuständen mit der selben Kanten, verbinde diese
 - (c) **Mindestens einer von den Zuständen**, die in dem neuen Zustand sind, ist ein **Endzustand** \rightarrow der neue Zustand wird ein Endzustand

Beispiel:

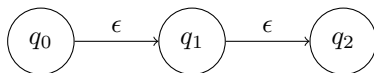


2.6 ϵ -NFA

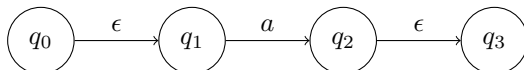
- Ein NFA mit ϵ -Übergängen ist ein NFA mit $\epsilon \notin \Sigma$ und $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q)$

2.7 ϵ -NFA \rightarrow NFA

1. Lösche überflüssige Zustände:



2. Verbinde die Zustände in der Form mit einer einzigen Kante:



wird zu



3. Lösche nicht erreichbare Zustände
4. Falls ϵ in der Sprache ist, dann mache den Startzustand Endzustand

2.8 Reguläre Ausdrücke (REs)

- Reguläre Ausdrücke sind induktiv definiert:
 - \emptyset
 - ϵ
 - Für jedes $a \in \Sigma$
 - Wenn α, β RE, auch:
 - $\alpha\beta$
 - $\alpha \mid \beta = \alpha + \beta$
 - α^*
- $*$, Kleene'sche Iteration
- Für RE γ ist die Sprache induktiv definiert:
 - $L(\emptyset) = \emptyset$
 - $L(\epsilon) = \{\epsilon\}$
 - $L(a) = \{a\}$
 - $L(\alpha\beta) = L(\alpha)L(\beta)$
 - $L(\alpha \mid \beta) = L(\alpha) \cup L(\beta)$
 - $L(\alpha^*) = L(\alpha)^*$
- $\alpha \equiv \beta$ gdw. $L(\alpha) = L(\beta)$
- **Rechenregeln über REs:**
 - **Null und Eins Lemma:**
 - $\emptyset \mid \alpha \equiv \alpha \mid \emptyset \equiv \alpha$
 - $\emptyset\alpha \equiv \alpha\emptyset \equiv \emptyset$
 - $\epsilon\alpha \equiv \alpha\epsilon \equiv \alpha$
 - $\emptyset^* \equiv \epsilon$
 - $\epsilon^* \equiv \epsilon$
 - **Assoziativität:**
 - $(\alpha \mid \beta) \mid \gamma \equiv \alpha \mid (\beta \mid \gamma)$
 - $\alpha(\beta\gamma) \equiv (\alpha\beta)\gamma$
 - **Kommutativität:**
 - $\alpha \mid \beta \equiv \beta \mid \alpha$
 - **Distributivität:**
 - $\alpha(\beta \mid \gamma) \equiv \alpha\beta \mid \alpha\gamma$
 - $(\beta \mid \gamma)\alpha \equiv \beta\alpha \mid \gamma\alpha$

– **Idempotenz:**

$$- \alpha \mid \alpha \equiv \alpha$$

• **Stern Lemma:**

- $\epsilon \mid \alpha\alpha^* \equiv \alpha^*$
- $\alpha^*\alpha \equiv \alpha\alpha^*$
- $(\alpha^*)^* \equiv \alpha^*$

2.9 Strukturelle Induktion für REs

Um zu beweisen, dass eine Eigenschaft $P(r)$ für alle regulären Ausdrücke gilt:

1. Zeige $P(\emptyset)$
2. Zeige $P(\epsilon)$
3. Zeige $P(a)$ für alle $a \in \Sigma$
4. Unter der Annahme $P(\alpha)$ und $P(\beta)$ (I.H.), zeige $P(\alpha\beta)$
→ verwende $L(\alpha\beta) = L(\alpha)L(\beta)$
5. Unter der Annahme $P(\alpha)$ und $P(\beta)$ (I.H.), zeige $P(\alpha \mid \beta)$
→ verwende $L(\alpha \mid \beta) = L(\alpha) \cup L(\beta)$
6. Unter der Annahme $P(\alpha)$ (I.H.), zeige $P(\alpha^*)$
→ verwende $L(\alpha^*) = L(\alpha)^*$

Beispiel: $\text{empty}(r)$ entscheidet, ob $L(r) = \emptyset$. Zeige die Korrektheit der Konstruktion:

(a) **Konstruktion**

- | | |
|---|---|
| • $\text{empty}(\emptyset) = \text{true}$ | • $\text{empty}(\alpha\beta) = \text{empty}(\alpha) \vee \text{empty}(\beta)$ |
| • $\text{empty}(a) = \text{false}$ | • $\text{empty}(\alpha \mid \beta) = \text{empty}(\alpha) \wedge \text{empty}(\beta)$ |
| • $\text{empty}(\epsilon) = \text{false}$ | • $\text{empty}(\alpha^*) = \text{false}$ |

Korrektheit Wir zeigen $L(r) = \emptyset \iff \text{empty}(r)$ mittels struktureller Induktion.

Fall $r = \emptyset, r = a, r = \epsilon$. Trivial.

Fall $r = \alpha^*$.

Wir haben $\epsilon \in L(\alpha^*) \neq \emptyset \iff \neg \text{empty}(\alpha^*)$. Die Aussage gilt per Definition von empty .

Fall $r = \alpha\beta$.

Als Induktionshypothesen erhalten wir $L(\alpha) = \emptyset \iff \text{empty}(\alpha)$ und $L(\beta) = \emptyset \iff \text{empty}(\beta)$.

Es gilt

$$\begin{aligned} L(\alpha\beta) = \emptyset &\iff L(\alpha)L(\beta) = \emptyset \\ &\iff L(\alpha) = \emptyset \vee L(\beta) = \emptyset \\ &\stackrel{\text{I.H.}}{\iff} \text{empty}(\alpha) \vee \text{empty}(\beta) = \text{empty}(\alpha\beta). \end{aligned}$$

Fall $r = \alpha \mid \beta$.

Es gelten dieselben Induktionshypothesen wie im vorherigen Fall.

Wir haben

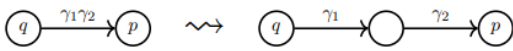
$$\begin{aligned} L(\alpha \mid \beta) = \emptyset &\iff L(\alpha) \cup L(\beta) = \emptyset \\ &\iff L(\alpha) = \emptyset \wedge L(\beta) = \emptyset \\ &\stackrel{\text{I.H.}}{\iff} \text{empty}(\alpha) \wedge \text{empty}(\beta) = \text{empty}(\alpha \mid \beta). \end{aligned}$$

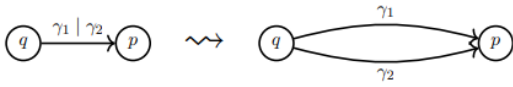
2.10 RE \rightarrow ϵ -NFA

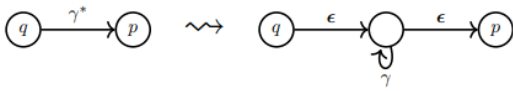
1. Wende folgende Ersetzungsregeln an

$$\begin{array}{lll} \gamma \emptyset \rightsquigarrow \emptyset & \gamma \mid \emptyset \rightsquigarrow \gamma & \emptyset^* \rightsquigarrow \epsilon \\ \emptyset \gamma \rightsquigarrow \emptyset & \emptyset \mid \gamma \rightsquigarrow \gamma & \end{array}$$

2. Wende folgende Transformationsregeln an

Konkatenation: 

Auswahl: 

Iteration: 

Beispiel:

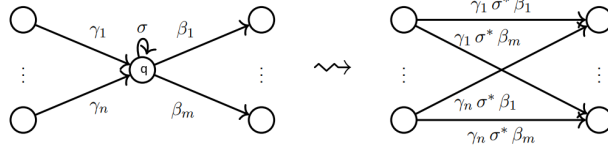


2.11 ϵ -NFA \rightarrow RE

1. Hat Startzustand q_1 eingehende Übergänge, füge einen neuen Startzustand q_0 mit einem ϵ -Übergang nach q_1
2. Füge einen neuen Endzustand q_3 und ϵ -Übergänge nach q_3 von allen Endzuständen (q_2 in diesem Beispiel)



3. Wähle ein Zustand q , der weder Start- noch Endzustand ist
4. Eliminiere q , wende dabei folgende Regeln:



Beispiel:



2.12 Ardens Lemma

- Sind A, B, X Sprachen mit $\epsilon \notin A$, so gilt:

$$X = AX \cup B \implies X = A^*B$$

- Sind α, β, X REs mit $\epsilon \notin L(\alpha)$, so gilt:

$$X \equiv \alpha X \mid \beta \implies X \equiv \alpha^* \beta$$

• **Bemerkungen:**

- $X \equiv X\alpha \mid \beta \implies X \equiv \beta\alpha^*$
- $X \equiv \alpha X \mid \beta$ für $\epsilon \in L(\alpha)$
 - * hat **keine eindeutige Lösung**: jede Sprache $B \subseteq X$ ist Lösung
 - * *Beispiel*: für $\alpha = \epsilon$ und $\beta = b$ kann $X = b$ oder $X = a \mid b$ oder $X = aba \mid b$
- $X \equiv X \mid aX$
 - * hat **keine eindeutige Lösung**: $X = \emptyset$ oder $X = \Sigma^*$ oder $X = a^*$
- $X \equiv \alpha X$ für $\epsilon \notin L(\alpha)$
 - * hat **eine eindeutige Lösung**: $X = \emptyset$
- $X \equiv \alpha X$ für $\epsilon \in L(\alpha)$
 - * hat **keine eindeutige Lösung**: $X = \Sigma^*$ oder $X = ab^*a$
- $X \equiv aXb \mid \epsilon$
 - * hat **keine reguläre Lösung**: $X = L$ für $L = \{a^n b^n, n \geq 0\}$
- $X \equiv abX \mid \epsilon$
 - * hat **eine eindeutige Lösung**: $X = (ab)^* \epsilon = (ab)^*$

2.13 FA \rightarrow RE mittels Ardens Lemma

1. FA als Gleichungssystem schreiben

- für jeden Endzustand X_f füge $X_f \equiv \epsilon$ ein
- für jeden Zustand X mit



machte $X \equiv aX_1$

2. Gleichungen einsetzen und damit eliminieren

3. Rechenregeln über REs und Ardens Lemma verwenden

2.14 Konversionen bezüglich regulärer Sprachen



RE \rightarrow ϵ -NFA: RE der Länge $n \rightsquigarrow O(n)$ Zustände

ϵ -NFA \rightarrow NFA: $Q \rightsquigarrow Q$

NFA \rightarrow DFA: n Zustände $\rightsquigarrow O(2^n)$ Zustände

NFA \rightarrow RE: n Zustände \rightsquigarrow RE der Länge $O(3^n)$

2.15 Abschlusseigenschaften regulärer Sprachen

Seien $L, L_1, L_2 \subseteq \Sigma^*$ reguläre Sprachen, dann sind auch

- $L_1 L_2$
- $L_1 \cup L_2, L_1 \cap L_2, L_1 \setminus L_2$
- \bar{L} bzw. $\Sigma^* \setminus L$
- L^*
- L^R (Spiegelung von L)
- $L_1 \times L_2$

2.16 Komplementierung \bar{L} bezüglich FAs

- **Für DFAs:** Vertauschen von Endzuständen und Nicht-Endzuständen (inkl. Fangzustand!)
- **Für NFAs:** funktioniert das Vertauschen **nicht**

2.17 Schnitt zweier DFAs, Produktkonstruktion

- Sind M_1 und M_2 DFAs. Dann ist der **Produkt-Automat M** mit $L(M) = L(M_1) \cap L(M_2)$.
- **Produktkonstruktion für M_1 und M_2 :**
 1. Erzeuge einen neuen Startzustand aus den Startzuständen der M_1 und M_2
 2. Bestimme wohin man mit welcher Kante geht
 3. Erzeuge neue Zustände durch Vereinigung der auf der rechten Seite stehenden Zuständen mit der selben Kanten, verbinde diese
 4. **Alle Zustände**, die in dem neuen Zustand sind, sind **Endzustände** \rightarrow der neue Zustand wird ein Endzustand

2.18 Vereinigung zweier DFAs

- Sind M_1 und M_2 DFAs. Dann ist M mit $L(M) = L(M_1) \cup L(M_2)$.
- **Konstruktion für M_1 und M_2 :** Gleich wie die Produktkonstruktion bis auf 4.
 4. **Mindestens einer von den Zuständen**, die in dem neuen Zustand sind, ist ein **Endzustand** \rightarrow der neue Zustand wird ein Endzustand

2.19 Pumping Lemma für reguläre Sprachen

- Sei $L \subseteq \Sigma^*$ regulär. Dann gibt es ein $n > 0$, so dass sich jedes $z \in L$ mit $|z| \geq n$ so in $z = uvw$ zerlegen lässt, dass
 1. $v \neq \epsilon$
 2. $|uv| \leq n$
 3. $\forall i \geq 0. uv^i w \in L$
- Um zu zeigen, dass eine Sprache **nicht regulär** ist \rightarrow **Regulärität** mit Pumping Lemma zu zeigen **nicht möglich**
- Es gibt nicht-reguläre Sprachen, für die das Pumping-Lemma gilt
 \rightarrow regulär \subset Pumping Lemma gilt \subset alle Sprachen
- Beispiel: $L = \{0^{m^2} \mid m \geq 0\}$

Angenommen L sei regulär.

Sei n eine Pumping-Lemma-Zahl für L .

Wähle $z = 0^{n^2} \in L$. Sei uvw eine Zerlegung von z mit $1 \leq |v| \leq |uv| \leq n$.

Zeige, dass $uv^i w \notin L$ für den Fall $i = 2$ gilt:

$$n^2 = |z| = |uvw| < |uv^2w| \leq n^2 + n \leq n^2 + 2n + 1 = (n+1)^2$$

Da es keine Quadratzahl zwischen n^2 und $(n+1)^2$ geben kann, ist $uv^2w \notin L$, damit ist L nicht regulär.

2.20 Entscheidungsprobleme für reguläre Sprachen

- **Wortproblem:** Gegeben w und D , gilt $w \in L(D)$?
 - für DFA M , in $O(|w| + |M|^1)$ entscheidbar
 - für NFA N , in $O(|Q|^2|w| + |N|)$ entscheidbar
- **Leerheitsproblem:** Gegeben D , gilt $L(D) = \emptyset$?
 - für DFA M , in $O(|Q||\Sigma|)$ entscheidbar
 - für NFA N , in $O(|Q|^2|\Sigma|)$ entscheidbar

¹Konstante für die Entscheidung, ob der Zustand den wir am Ende erreichen, ein Endzustand ist

- **Endlichkeitsproblem:** Gegeben D , ist $L(D)$ endlich?
 - für DFA und NFA entscheidbar
 - $L(M) = \infty$ gdw. vom Startzustand aus eine nicht-leere Schleife erreichbar ist, von der aus F erreichbar ist



- **Äquivalenzproblem:** Gegeben D_1 und D_2 , gilt $L(D_1) = L(D_2)$?
 - schaue, ob $L(M_1) \cap \overline{L(M_2)} = \emptyset$ und $\overline{L(M_1)} \cap L(M_2) = \emptyset$ gelten
 - für DFAs, in $O(|Q_1||Q_2||\Sigma|)$ entscheidbar
 - für NFA N , in $O(2^{|Q_1|+|Q_2|})$ entscheidbar (bei fixem Σ)

2.21 Minimierung von FAs

- Zustände p und q sind **unterscheidbar**, wenn $\exists w \in \Sigma^*$ mit $\hat{\delta}(p, w) \in F$ und $\hat{\delta}(q, w) \notin F$ oder umgekehrt
- Zustände p und q sind **äquivalent**, wenn $\forall w \in \Sigma^*$ mit $\hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F$
- Gilt $p \in F$ und $q \notin F$, dann sind p und q unterscheidbar
- Sind $\delta(p, a)$ und $\delta(q, a)$ unterscheidbar, dann auch p und q

2.22 Minimierungsalgorithmus für DFAs

1. Konstruiere die Treppe $\forall q \in Q$
2. Markiere Endzustände und Nichtendzustände mit einem ϵ
3. Für alle unmarkierten Paare (q, p) : Falls $(\delta(q, a), \delta(p, a))$ markiert, markiere das Paar (q, p) mit a
 - (a) Falls es b im Kästchen von $(\delta(q, a), \delta(p, a))$ gibt, dann markiere das Paar (q, p) mit ab
4. Für alle unmarkierten Paare (q, p) : Schmelze q und p zusammen, evtl. markiere das Paar mit $=$

Beispiel:



2.23 Äquivalenz von Zuständen eines DFAs

- **Äquivalenzklasse:** $[p]_{\equiv_M} = \{q \mid p \equiv_M q\}$, Menge der Zustände, die mit p äquivalent sind
- **Quotientenmenge:** $Q/\equiv_M = \{[p]_{\equiv_M} \mid p \in Q\}$, Menge der Äquivalenzklassen
- $p \equiv_M q \Leftrightarrow L_M(p) = L_M(q)$
 - $L_M(q) = \{w \in \Sigma^* \mid \hat{\delta}(q, w) \in F\}$, Sprache vom Zustand q
 - Zwei Zustände sind äquivalent wenn sie die gleiche Sprache erkennen
 - **Fakt:** $|Q/\equiv_M| = \text{Anzahl der Sprachen, die von Zuständen von } M \text{ erkannt werden}$
- **Quotientenautomat:** $M/\equiv = (Q/\equiv, \Sigma, \delta', [q_0]_{\equiv}, F/\equiv)$ mit $\delta'([p]_{\equiv}, a) = [\delta(p, a)]_{\equiv}$
- Quotientenautomat M/\equiv ist ein **minimaler DFA** für $L(M)$
- $L(M/\equiv) = L(M)$

2.24 Residualsprache, Äquivalenz von Wörtern

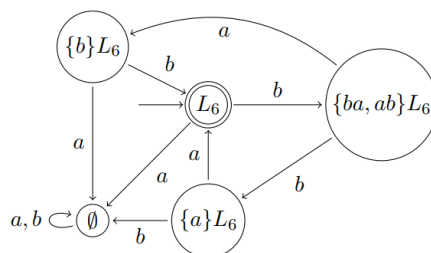
- $L^w = \{z \in \Sigma^* \mid wz \in L\}$, Residualsprache von L bzgl. $w \in \Sigma^*$
- $u \equiv_L v \Leftrightarrow L^u = L^v$
 - Zwei Wörter sind äquivalent wenn sie die gleiche Residualsprache haben
 - **Fakt:** $|\Sigma^*/\equiv_L| = \text{Anzahl der Residualsprachen von } L$
- **Fakt:** $|Q/\equiv_M| = |\Sigma^*/\equiv_L|$, Anzahl der Residualsprachen entspricht der Anzahl der Zustände im minimalen Automat

2.25 Myhill-Nerode Relation

- Eine Sprache L ist **regulär** gdw. Anzahl der Residualsprachen von L **endlich**
- **Beweis mittels Myhill-Nerode Relation:** $L = \{a^i b^i c^i \mid i \in \mathbb{N}\}$
 1. Bestimmen einer unendlichen Menge von Wörtern mit unterschiedlichen Residualsprachen: $\{a^i b^i \mid i \in \mathbb{N}\}$
 2. Sei $i, j \in \mathbb{N}$ verschieden. Dann $a^i b^i c^i \in L$, aber $a^j b^j c^i \notin L$. Daher $L^{a^i b^i} \neq L^{a^j b^j}$.
 3. Somit sind alle Residualsprachen unterschiedlich und L **keine reguläre Sprache**

2.26 Kanonischer Minimalautomat

- $M_L = \{R_L, \Sigma, \delta_L, L, F_L\}$ mit $\delta_L(R, a) = R^a$ und $F_L = \{R \in R_L \mid \epsilon \in R\}$, kanonischer Minimalautomat M_L mit $L(M_L) = L$
- R_L , Menge der Residualsprachen von L
- **Durch Umbenennung von Zuständen** von jedem minimalen DFA bekommt man den **kanonischen Minimalautomat**
- Kanonischer Minimalautomat M_L ist **gleich gross** wie Quotientenautomat und damit ein **minimaler DFA** für $L(M)$
- Beispiel: $L = L((bba \mid bab)^*)$ mit $\Sigma = \{a, b\}$



2.27 Regulärkeit der Sprachen

Eine Sprache ist **regulär**

- gdw. sie von einer **DFA, NFA, ϵ -NFA, RE, rechtslinearen Grammatik akzeptiert wird**
- gdw. sie durch **Abschlusseigenschaften der regulären Sprachen** entsteht
- gdw. sie **endliche Anzahl von Residualsprachen** hat

Eine Sprache ist **nicht regulär**

- gdw. für sie **Pumping-Lemma nicht gilt**
- gdw. sie **unendliche Anzahl von Residualsprachen hat** (Myhill-Nerode)

3 Kontextfreie Sprachen (CFL)

3.1 Kontextfreie Grammatik (CFG)

- Kontextfreie Grammatik $G = (V, \Sigma, P, S)$ mit $P \subseteq V \times (V \cup \Sigma)^*$
- Produktionen folgender Gestalt:
 - $X \rightarrow \alpha$, mit $\alpha \in (V \cup \Sigma)^*$
- $\alpha_1 \rightarrow_G \alpha_2 \rightarrow_G \dots \rightarrow_G \alpha_n$ nennt man eine **Linksableitung** gdw. in jedem Schritt **das linkeste Nichtterminal** in α_i ersetzt wird
- **Rechtsableitung** analog zu Linksableitung

3.2 Induktive Definition einer Sprache mittels Grammatik

- Sei G eine Grammatik: $S \rightarrow \epsilon \mid +S - S \mid +S \mid \epsilon$
- Um zu zeigen, dass G die Menge aller *nicht überziehenden* Wörter erzeugt, betrachten wir die Grammatik als **induktive Definition einer Sprache** $L(G)$
- Wort w ist überziehend gdw. $\exists i$, sodass $\Delta(w_1 \dots w_i) < 0$ mit $\Delta(w) = |w|_+ - |w|_-$
- **nicht überziehend:** $\epsilon, ++--+-$
- **überziehend:** $-+, +-+- -++$
- **Induktive Definition von $L(G)$:**
 - $\epsilon \in L(G)$
 - $u \in L(G) \implies +u \in L(G)$
 - $u \in L(G) \wedge v \in L(G) \implies +u - v \in L(G)$
- Produktionen (\rightarrow) erzeugen Wörter **top-down**: Nichtterminal \rightarrow Wort
- Induktive Definition (\implies) erzeugt Wörter **bottom-up**: kleinere Wörter \implies grössere Wörter
- Induktive Definition betrachtet nur Wörter aus Σ^*

3.3 (Strukturelle) Induktion über die Erzeugung eines Wortes

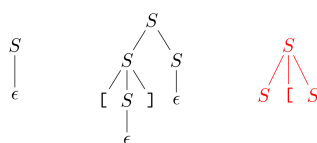
- $w \in L(G) \implies$ beweist man mit **Induktion über Erzeugung von w**
- Um zu zeigen, dass für alle $u \in L(G)$ eine Eigenschaft $P(u)$ gilt, zeige:
 - $P(\epsilon)$
 - $P(u) \implies P(+u)$
 - $P(u) \wedge P(v) \implies P(uv)$
- Beispiel: Grammatik $S \rightarrow \epsilon \mid +S - S \mid +S \mid \epsilon$ erzeugt die Wörter, die nicht überziehend sind.
 - Induktionsbasis: $\epsilon \in L(G)$ ist nicht überziehend
 - Induktionsschritt: Seien u, v nicht überziehende Wörter. Es gibt 2 Fälle:
 1. $w = +u$: Für beliebiges i gilt $\Delta(w_1 \dots w_i) = 1 + \Delta(u_1 \dots u_{i-1})$. Da u nicht überziehend ist, folgt $\Delta(u_1 \dots u_{i-1}) \geq 0$ und somit $\Delta(w_1 \dots w_i) \geq 1 \geq 0$. Also ist w auch nicht überziehend.
 2. $w = +u - v$: Für $i \in \{1, \dots, |u|+2\}$ wissen wir bereits $\Delta(w_1 \dots w_i) \geq 0$, analog zum ersten Fall. Für $i > |u|+2$ gilt nun $\Delta(w_1 \dots w_i) = \Delta(+u - v_1 \dots v_j)$, mit $j = i - |u| - 2$. Es folgt $\Delta(+u - v_1 \dots v_j) = \Delta(u) + \Delta(v_1 \dots v_j) \geq 0$, da sowohl u als auch v nicht überziehend ist.

3.4 Induktion über die Länge des Wortes

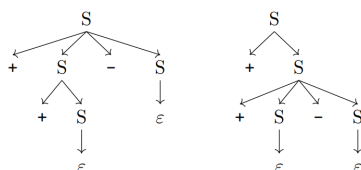
- $P(w) \implies w \in L(G)$ beweist man mit **Induktion über $|w|$**
- Beispiel: Die nicht überziehenden Wörter werden durch die Grammatik $S \rightarrow \epsilon \mid +S - S \mid +S \mid \epsilon$ erzeugt.
 - Induktionsannahme: Für nicht überziehendes w gilt $w \in L(G)$
 - Induktionsbasis: Für $|w| = 0$ gilt $w = \epsilon \in L(G)$
 - Induktionsschritt:
 1. Falls $w = +u$ für ein u nicht überziehend, dann wissen wir nach Induktionsannahme, dass $u \in L(G)$. Daraus folgt $S \rightarrow +S - S \rightarrow^* +u = w$ und somit $w \in L(G)$.
 2. Falls $w = +u - v$ für u, v nicht überziehend: Nach Induktionsannahme gilt $u, v \in L(G)$, also erhalten wir $S \rightarrow +S - S \rightarrow^* +u - S \rightarrow^* +u - v = w$ und somit $w \in L(G)$.

3.5 Syntaxbaum, Mehrdeutigkeit

- Für eine CFG und ein $w \in \Sigma^*$ sind folgende Bedingungen äquivalent:
 - $A \rightarrow_G^* w$
 - $w \in L_G(A)$ (induktive Definition)
 - Es gibt einen Syntaxbaum mit Wurzel A , dessen Rand (Blätter von links nach rechts gelesen) das Wort w ist
- Syntaxbaum für eine Ableitung mit Grammatik $S \rightarrow \epsilon \mid [S] \mid SS$: $w = \epsilon$, $w = []$, kein gültiges Baum



- CFG G **mehrdeutig** gdw. es gibt $w \in L(G)$, das **zwei verschiedene Syntaxbäume** hat
- CFL L **inhärent mehrdeutig** gdw. **jede CFG** G mit $L(G) = L$ **mehrdeutig**
- Beispiel: $S \rightarrow \epsilon \mid +S-S \mid +S \mid \epsilon$ und $w = ++-$ $\Leftrightarrow L_G(S)$ ist mehrdeutig



3.6 Chomsky-Normalform (CNF)

- Eine CFG G ist in Chomsky-Normalform gdw. alle Produktionen eine der Formen haben: $A \rightarrow a$ oder $A \rightarrow BC$
- ϵ -Produktion: $A \rightarrow \epsilon$
- Kettenproduktion: $A \rightarrow B$
- Zu jeder CFG G kann man eine CFG G' in Chomsky-Normalform konstruieren, die **keine** ϵ -Produktionen und Kettenproduktionen enthält, so dass gilt $L(G') = L(G) \setminus \{\epsilon\}$

- **Konstruktion:**

1. Füge für jedes $a \in \Sigma$ mit der Länge ≥ 2 ein neues Nichtterminal X_a und eine neue Produktion $X_a \rightarrow a$ hinzu:
 - $A \rightarrow aBC$ wird zu $A \rightarrow X_aBC$
 $X_a \rightarrow a$
2. Ersetze jede Produktion der Form $A \rightarrow B_1B_2 \dots B_k$ mit der Länge $k \geq 3$:
 - $A \rightarrow BCD$ wird zu $A \rightarrow BX_{CD}$
 $X_{CD} \rightarrow CD$
3. Eliminiere alle ϵ -Produktionen, indem wir zuerst ϵ in Produktionen einsetzen und dann eliminieren:
 - $A \rightarrow XY$ wird zu $A \rightarrow X\epsilon$ wird zu $A \rightarrow X$
 $Y \rightarrow \epsilon$ $Y \rightarrow \epsilon$
4. Eliminiere alle Kettenproduktionen, indem wir zuerst Nichtterminale in Produktionen einsetzen und dann eliminieren
 - $A \rightarrow X \mid YZ$ wird zu $A \rightarrow YZ \mid x \mid DL \mid B$
 $X \rightarrow x \mid DL \mid B$ $B \rightarrow b$
 $B \rightarrow b$
 - wird zu $A \rightarrow YZ \mid x \mid DL \mid b$

3.7 Greibach-Normalform

- Eine CFG ist in Greibach-Normalform gdw. alle Produktionen die Form $A \rightarrow aA_1 \dots A_n$ haben
- Zu jeder CFG G gibt es eine CFG G' in Greibach-Normalform mit $L(G') = L(G) \setminus \{\epsilon\}$

3.8 Pumping Lemma für CFLs

- Für jede CFL L gibt es ein $n > 0$, so dass sich jedes Wort $z \in L$ mit $|z| \geq n$ zerlegen lässt in $z = uvwxy$, dass
 1. $vx \neq \epsilon$ bzw. $|vx| \geq 1$
 2. $|vwx| \leq n$
 3. $\forall i \geq 0. uv^iwx^iy \in L$
- Beispiel: $L = \{a^ib^ic^i \mid i \in \mathbb{N}\}$

Angenommen L sei kontextfrei.

Sei n eine Pumping-Lemma-Zahl für L .

Wähle $z = a^n b^n c^n \in L$. Sei $uvwxy$ eine Zerlegung von z mit $vx \neq \epsilon$ und $|vwx| \leq n$. Wir betrachten nun 2 Fälle:

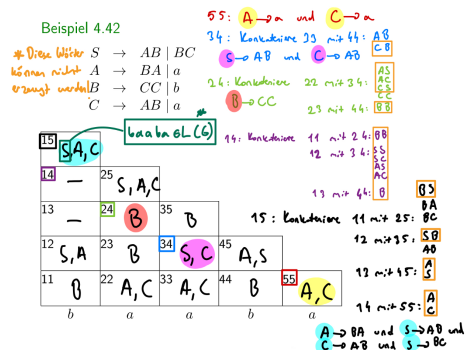
1. uvw enthält nur as oder bs oder cs : Für $i = 2$ erhalten wir $uv^2wx^2y = a^{i+2|v|+2|x|}b^ic^i$. Da $|vx| \geq 1$, gilt $uv^2wx^2y \notin L$.
2. uvw enthält nur as und bs oder bs und cs : Hier muss vx mindestens ein a und ein b enthalten. Damit gilt aber $|uv^2wx^2y|_a > |uv^2wx^2y|_c$. Also $uv^2wx^2y \notin L$.

3.9 Erzeugend, Erreichbar, Nützlich

- Sei $G = (V, \Sigma, P, S)$ CFG. Ein Symbol $X \in V \cup \Sigma$ ist
 - **erzeugend** gdw. es eine Ableitung $X \rightarrow_G^* w \in \Sigma^*$ gibt
 - **erreichbar** gdw. es eine Ableitung $S \rightarrow_G^* \alpha X \beta$ gibt
 - **nützlich** gdw. erzeugend und erreichbar
- Bekommt man eine Grammatik G' mit $L(G') = L(G)$, die nur **nützliche Symbole** enthält, durch:
 1. Elimination der **nicht erzeugenden** Symbole
 2. Elimination der **nicht erreichbaren** Symbole
- Menge der erzeugenden Symbole einer CFG ist berechenbar
- Menge der erreichbaren Symbole einer CFG ist berechenbar

3.10 Cocke-Younger-Kasami-Algorithmus (CYK)

- **Wortproblem** ist für eine CFG G mittels CYK-Algorithmus in Zeit $O(|w|^3)$ entscheidbar
- **Algorithmus:** $w \in L(G)$?
 1. Konstruiere die Treppe (Breite = Höhe = $|w|$) und beschrifte jede Spalte von unten nach oben: Für 1. Spalte 1, $1 - 1, 2 - \dots - 1, |w|$; für 3. Spalte 3, $3 - 3, 4 - \dots - 3, |w|$
 2. Fülle die erste Reihe mit Nichtterminalen ein, die die einzelnen Buchstaben des Wortes erzeugen
 3. Fülle die anderen Kästchen mit Nichtterminalen nach Beschriftungen ein: Für 1, 2 konkateniere 1, 1 2, 2; für 1, 4 konkateniere 1, 1 2, 4 und 1, 2 3, 4 und 1, 3 4, 4
 4. Wenn es im Kästchen 1, $|w|$ das Startsymbol gibt, dann gilt $w \in L(G)$, wenn nicht $w \notin L(G)$
- Beispiel: $baaba \in L(G)$?



3.11 Nicht Entscheidbare Probleme für CFGs

- **Äquivalenz:** $L(G_1) = L(G_2)$?
- **Schnittproblem:** $L(G_1) \cap L(G_2) = \emptyset$?
- **Regularität:** $L(G)$ regulär?
- **Mehrdeutigkeit:** Ist G mehrdeutig?

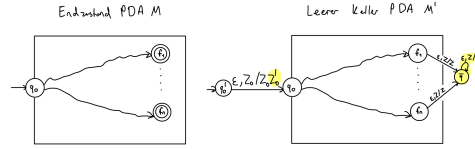
3.12 Nicht Deterministischer Kellerautomat (NPDA)

- Ein (nichtdeterministischer) Kellerautomat $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$ besteht aus
 - Q , endliche Menge von **Zuständen**
 - Σ , endliches **Eingabealphabet**
 - Γ , endliches **Kelleralphabet**
 - q_0 , **Startzustand**
 - Z_0 , **unterstes Kellerzeichen**
 - $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow \mathcal{P}_e(Q \times \Gamma^*)$, **Übergangsfunktion**
 - $F \subseteq Q$, Menge von **Endzuständen**
- **Bedeutung von $(q', \alpha) \in \delta(q, a, Z)$:** Wenn sich M in Zustand q befindet, das Eingabezeichen a liest und Z das oberste Kellerzeichen ist, so kann M im nächsten Schritt in q' übergehen und Z durch α ersetzen.
 - **POP-Operation:** $\alpha = \epsilon$, das oberste Kellerzeichen Z wird entfernt
 - **PUSH-Operation:** $\alpha = Z'Z$, Z' wird als neues oberstes Kellerzeichen gepusht
 - **ϵ -Übergang:** $a = \epsilon$, ohne Lesen eines Eingabezeichens
- Eine **Konfiguration** eines Kellerautomaten M ist ein Tripel (q, w, α) mit $q \in Q$, $w \in \Sigma^*$ und $\alpha \in \Gamma^*$

- q , der **momentane Zustand**
- w , **noch zu lesende Teil der Eingabe**
- α , der **aktuelle Inhalt des Kellers**
- **Anfangskonfiguration** von M für die Eingabe $w \in \Sigma^*$ ist (q_0, w, Z_0)
- Auf der Menge aller Konfigurationen definieren wir binäre Relation \rightarrow_M :
$$(q, aw, Z\alpha) \rightarrow_M \begin{cases} (q', w, \beta\alpha) & \text{falls } (q', \beta) \in \delta(q, a, Z) \\ (q', aw, \beta\alpha) & \text{falls } (q', \beta) \in \delta(q, \epsilon, Z) \end{cases}$$
- **Bedeutung von $(q, w, \alpha) \rightarrow_M (q', w', \alpha')$** : Wenn M sich in der Konfiguration (q, w, α) befindet, dann kann er in einen Schritt in die Nachfolgerkonfiguration (q', w', α') übergehen.
- Eine Konfiguration kann **mehrere Nachfolgerkonfigurationen** haben \rightarrow **Nichtdeterminismus**
- PDA M **akzeptiert** $w \in \Sigma^*$ **mit Endzustand** gdw. $(q_0, w, Z_0) \rightarrow_M^* (f, \epsilon, \gamma)$ für $f \in F, \gamma \in \Gamma^*$
- PDA M **akzeptiert** $w \in \Sigma^*$ **mit leeren Keller** gdw. $(q_0, w, Z_0) \rightarrow_M^* (q, \epsilon, \epsilon)$ für $q \in F$
- Akzeptanz durch Endzustände und leeren Keller **gleich mächtig**

3.13 Endzustand PDA $M \rightarrow$ Leerer Keller PDA M'

- **Idee:**
 1. Sobald M einen Endzustand erreicht, darf er den Keller leeren \rightarrow M' **geht in den neuen Nicht-Endzustand \bar{q} und leert dort den Keller**, es gibt **keine Endzustände mehr**
 2. Verhindern, dass der Keller von M leer wird, ohne dass M in einem Endzustand ist $\rightarrow M'$ hat ein **neues Symbol Z' ganz unten im Keller**
- $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$
- $M' = (Q', \Sigma, \Gamma', q'_0, Z'_0, \delta')$ mit $Q' = Q \uplus \{q'_0, \bar{q}\}$, $\Gamma' = \Gamma \uplus \{Z'_0\}$ und
 - $\delta'(q'_0, \epsilon, Z'_0) = \{(q_0, Z_0 Z'_0)\} \rightarrow$ **Übergang von q'_0 zu q_0**
 - $\delta'(q, a, Z) = \delta(q, a, Z)$ für $q \in Q \setminus F, a \in \Sigma \cup \{\epsilon\}, Z \in \Gamma \rightarrow$ **Alle Übergänge zwischen qs**
 - $\delta'(f, a, Z) = \delta(f, a, Z)$ für $f \in F, a \in \Sigma, Z \in \Gamma \rightarrow$ **Alle Übergänge von Endzuständen zu qs**
 - $\delta'(f, \epsilon, Z) = \delta(f, \epsilon, Z) \cup \{(\bar{q}, Z)\}$ für $Z \in \Gamma \rightarrow$ **Alle ϵ -Übergänge von Endzuständen zu qs und \bar{q}**
 - $\delta'(\bar{q}, \epsilon, Z) = \{(\bar{q}, \epsilon)\}$ für $Z \in \Gamma' \rightarrow$ **Übergang von \bar{q} zu \bar{q} zum Leeren des Kellers**

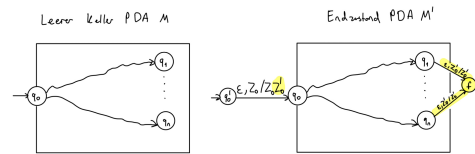


3.14 Leerer Keller PDA $M \rightarrow$ Endzustand PDA M'

- **Idee:**

1. Es wird am Anfang Z'_0 auf Keller geschrieben, damit der Keller nicht geleert wird
2. Sobald M auf dem Keller Z'_0 findet, muss er in den Endzustand gehen, somit ist der Keller am Ende nicht leer $\rightarrow M'$ **geht in den neuen Endzustand f**

- $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta)$
- $M' = (Q', \Sigma, \Gamma', q'_0, Z'_0, \delta', F')$ mit $Q' = Q \uplus \{q'_0, f\}$, $\Gamma' = \Gamma \uplus \{Z'_0\}$, $F' = \{f\}$ und
 - $\delta'(q'_0, \epsilon, Z'_0) = \{(q_0, Z_0 Z'_0)\} \rightarrow$ **Übergang von q'_0 zu q_0**
 - $\delta'(q, a, Z) = \delta(q, a, Z)$ für $q \in Q$, $a \in \Sigma \cup \{\epsilon\}$, $Z \in \Gamma \rightarrow$ **Alle Übergänge zwischen qs**
 - $\delta'(q, \epsilon, Z'_0) = \{(f, Z'_0)\}$ für $q \in Q$, $f \in F' \rightarrow$ **Übergang von q zu f beim Sehen von Z'_0**



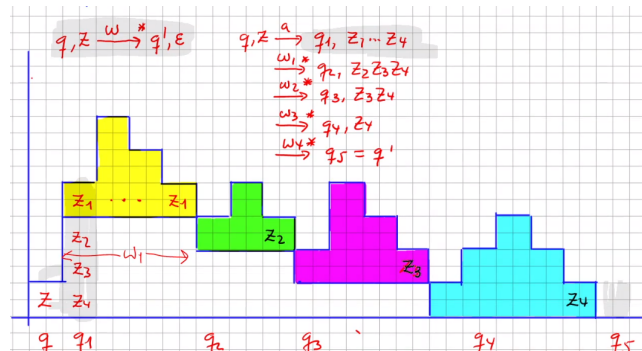
3.15 Erweiterungslemma

- $(q, u, \alpha) \rightarrow_M^n (q', u', \alpha') \implies (q, uv, \alpha\beta) \rightarrow_M^n (q', u'v, \alpha'\beta)$
 - $q' \in Q$ ist der neue Zustand, den wir in n Schritten erreichen
 - $u' \in \Sigma^*$ ist der noch zu lesende Teil von der Eingabe u
 - $\alpha' \in \Gamma^*$ ist der veränderte Kellerinhalt durch das Lesen eines Teils vom Wort u , der ganz oben im Keller steht
 - $v \in \Sigma^*$ ist eine neue Eingabe, die mit u konkateniert wird
 - $\beta \in \Gamma^*$ ist der neue Kellerzeichenkette, die ganz unten im Keller steht und während der ganzen Berechnungen unverändert erhalten bleibt

- **Erweiterungslemma** sagt, dass man **den Kellerinhalt sowie die Eingabe erweitern** kann, indem man ganz unten im Keller oder ganz hinten in der Eingabe etwas hinzufügt, ohne die Eigenschaft zu zerstören, dass man **wieder in n gleichen Schritten die gleiche Konfiguration (plus das extra unberührte Kellerinhalt und die Eingabe) erreichen** kann.

3.16 Zerlegungssatz

- **Zerlegungssatz** sagt, dass die Berechnung von einem Wort $(q, w, Z_{1...k}) \rightarrow_M^n (q', \epsilon, \epsilon)$ mit $w = u_1 \dots u_k$, $Z_i \in \Gamma^*$ **sich in k Teile zerlegen lässt**, indem man bei jeder Zerlegung **nur ein Teil** von w liest und zum Lesen **beliebig viele** PUSH- und POP-Operationen macht **ohne den Kellerteil von den folgenden Zerlegungen zu berühren** (Erweiterungslemma).



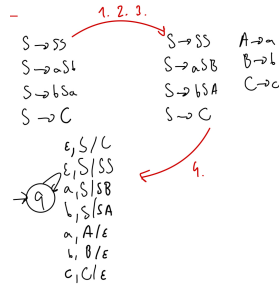
- Z_1 hat die Höhe 4, d.h. die Kellerzeichen Z_2, Z_3, Z_4 unten werden nicht berührt.

3.17 CFG \rightarrow PDA

- Zu jeder CFG G kann man einen PDA M konstruieren, der mit leerem Keller akzeptiert, so dass $L_\epsilon(M) = L(G)$
- **Konstruktion:** Bringe alle Produktionen in die Form: $A \rightarrow bB_1 \dots B_k$ mit $b \in \Sigma \cup \{\epsilon\}$
 1. Füge für jedes Terminal $a \in \Sigma$ ein neues Nichtterminal X_a hinzu
 2. Ersetze alle a s ausser dem ersten Terminalzeichen durch X_a in den Produktionen auf der rechten Seite
 3. Füge eine neue Produktion $X_a \rightarrow a$ hinzu
 4. Übersetze die Produktionen in Transitionen von PDA, die Schleifen über einen einzigen Zustand sind:
 - $S \rightarrow G$ zu $\epsilon, S/G$

- $B \rightarrow bSS$ zu $b, B/SS$
- $A \rightarrow a$ zu $a, A/\epsilon$

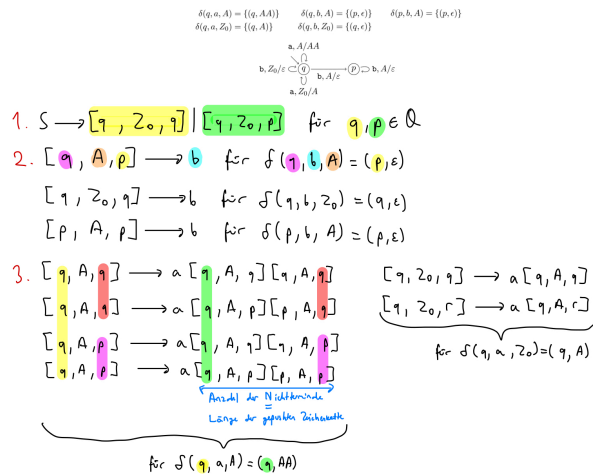
• Beispiel:



3.18 PDA \rightarrow CFG

- Zu jedem PDA M , der mit leerem Keller akzeptiert, kann man eine CFG G konstruieren mit $L(G) = L_\epsilon(M)$
- **Konstruktion:**
 1. Für alle $q \in Q$ füge die Produktion $S \rightarrow [q_0, Z_0, q]$
 2. Für alle POP-Operationen $\delta(q, a, Z) = (q', \epsilon)$ füge die Produktion $[q, Z, q'] \rightarrow a$
 3. Für alle PUSH-Operationen $\delta(q, a, Z) = (r_0, Z_1 \dots Z_k)$ und für alle $r_1 \dots r_k$ füge die Produktion $[q, Z, r_k] \rightarrow a[r_0, Z_1, r_1][r_1, Z_2, r_2] \dots [r_{k-1}, Z_k, r_k]$

• Beispiel:



3.19 Deterministisches Kellerautomat (DPDA)

- Ein PDA ist **deterministisch** gdw. für alle $q \in Q$, $a \in \Sigma$ und $Z \in \Gamma$ gilt $|\delta(q, a, Z)| + |\delta(q, \epsilon, Z)| \leq 1$
- NPDA ist **mächtiger** als DPDA
- CFL ist **deterministisch** (DCFL) gdw. sie **von einem DPDA akzeptiert** wird
- Jede **reguläre Sprache** ist eine **CFL**
- Die vom leeren Keller akzeptierten DCFLs sind die Sprachen, die **vom Endzustand akzeptiert** werden und **präfixfrei** sind: Kein Wort in der Sprache ist die Präfix eines anderen Wortes in der Sprache
- Jede DCFL ist **nicht inhärent mehrdeutig**, d.h. sie wird von einer **nicht-mehrdeutigen Grammatik** erzeugt

3.20 Abschlusseigenschaften der CFLs

Seien $L, L_1, L_2 \subseteq \Sigma^*$ kontextfreie Sprachen, dann sind auch

- $L_1 \cup L_2$
- $L_1 L_2$
- L^*
- L^R
- $L_1 \times L_2$

3.21 Abschlusseigenschaften der DCFLs

Sei $L \subseteq \Sigma^*$ eine deterministische kontextfreie Sprache, dann ist auch

- \bar{L}

4 Abschlusseigenschaften und Entscheidbarkeit

4.1 Abschlusseigenschaften

	$L_1 \cap L_2$	$L_1 \cup L_2$	\bar{L}	$L_1 \times L_2$	L^*	$L_1 L_2$	L^R
Regulär	ja	ja	ja	ja	ja	ja	ja
CFL	nein	ja	nein	ja	ja	ja	ja
DCFL	nein	nein	ja	nein	nein	nein	nein

4.2 Entscheidbarkeit

	Wort	Leerheit	Äquivalenz	Schnitt	Endlichkeit
DFA	ja	ja	ja	ja	ja
NFA	ja	ja	ja	ja	ja
CFG	ja	ja	ja	nein	ja
DPDA	ja	ja	nein	nein	ja

	Regularität	Mehrdeutigkeit
CFG	nein	nein
DPDA	ja	ja

5 Berechenbarkeit, Entscheidbarkeit

5.1 Grundbegriffe der Berechenbarkeit

- **Intuitiver Begriff:** Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ ist **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 $f(n_1, \dots, n_k)$ **definiert** ist
 - **nicht terminiert**, falls $f(n_1, \dots, n_k)$ **nicht definiert** ist
- Funktion $f : A \rightarrow B$ ist
 - **total** gdw. $f(a)$ für alle $a \in A$ definiert ist
 - **partiell** gdw. $f(a)$ für beliebige oder keine $a \in A$ definiert ist
 - **echt partiell** gdw. sie nicht total ist
- **Jeder Algorithmus** berechnet eine **partielle Funktion**
 - Beispiel 1: Algorithmus *input(n); while true do;* berechnet die überall undefinierte partielle Funktion, d.h. $\emptyset \subset \mathbb{N} \rightarrow \mathbb{N}$
 - Beispiel 2: $f(n) \begin{cases} 1 & \text{falls } n \text{ als Ziffernfolge Anfangsstück} \\ & \text{der Dezimalbruchentwicklung von } \pi \text{ ist} \\ 0 & \text{sonst} \end{cases}$
 f ist berechenbar ($f(314) = 1, f(315) = 0$), denn es Algorithmen gibt, die π iterativ auf beliebig viele Dezimalstellen genau berechnet werden.
 - Beispiel 3: $f(n) \begin{cases} 1 & \text{falls } n \text{ als Ziffernfolge irgendwo in} \\ & \text{der Dezimalbruchentwicklung von } \pi \text{ ist} \\ 0 & \text{sonst} \end{cases}$
 Es ist unbekannt, ob f berechenbar ist. Wie stellt man fest, dass n nicht vorkommt, wenn der Dezimalbruchentwicklung nie terminiert?

- Beispiel 4: $f(n) = \begin{cases} 1 & \text{falls 10 mal hintereinander Nullen in} \\ & \text{der Dezimalbruchentwicklung von } \pi \text{ ist} \\ 0 & \text{sonst} \end{cases}$

f ist berechenbar, denn f ist entweder die konstante Funktion, die 1 für alle $n \geq 0$ zurückgibt, oder die Funktion, die 0 für alle $n \geq 0$ zurückgibt. Beide sind berechenbar.

Das ist ein **nicht-konstruktiver Beweis**: Wir wissen, es gibt einen Algorithmus, der f berechnet, aber **wissen nicht, welcher**.

- Es gibt **nicht berechenbare** Funktionen $\mathbb{N} \rightarrow \{0, 1\}$
 - Es gibt **abzählbar** viele Algorithmen, aber **überabzählbar** viele Funktionen in $\mathbb{N} \rightarrow \{0, 1\}$
 - Menge M ist **abzählbar** falls es eine injektive Funktion $M \rightarrow \mathbb{N}$ gibt, Äquivalente Definitionen:
 - * Entweder gibt es eine Bijektion $M \rightarrow \{0, \dots, n\}$ für ein $n \in \mathbb{N}$, oder eine Bijektion $M \rightarrow \mathbb{N}$
 - * Es gibt eine Nummerierung der Elemente von M
 - Menge M ist **überabzählbar** wenn sie nicht abzählbar ist
 - **Zeige: Es gibt abzählbar viele Algorithmen**
 - * Da Σ^* (falls Σ endlich) durch Nummerierung **abzählbar** ist und die **Algorithmen** nur **Wörter** sind, ist die **Menge der Algorithmen abzählbar**
 - **Zeige: Es gibt überabzählbar viele Funktionen $\mathbb{N} \rightarrow \{0, 1\}$**
 - * Wir nehmen an, dass die Menge der Funktionen abzählbar ist.
 - * **Diagonalargument:**

	0	1	2	3	...
f_0	0	1	1	0	...
f_1	1	0	0	0	...
f_2	0	0	1	0	...
f_3	0	0	1	0	...
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots

3 Eingabewerte über \mathbb{N}

Ausgabewerte der Funktionen über $\{0, 1\}$

Eine Funktion f , die nicht in der Tabelle ist: **Diagonale!**

f	1	1	0	0	...
-----	---	---	---	---	-----

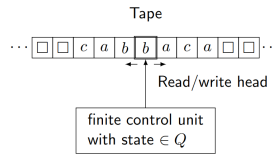
Widerspruch! \rightarrow Diese Funktion vergisst man immer beim Zählen!

Formal: Sei $f(i) := 1 - f_i(i)$. Dann $f \neq f_i$ für alle i , da $f(i) \neq f_i(i)$. □

- Es gibt verschiedene Formalisierungen des Begriffs der Berechenbarkeit, wie Turing-Maschinen, λ -Kalkül, while-/goto-Programme usw. Sie sind alle **gleichmächtig**.
- **Church-Turing These:** Der formale Begriff der Berechenbarkeit mit Turing-Maschinen stimmt mit dem intuitiven Begriff der Berechenbarkeit überein.

5.2 Turingmaschine

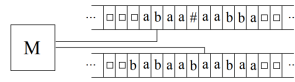
- Eine **Turingmaschine (TM)** ist ein 7-Tupel $M = (Q, \Sigma, \Gamma, \delta, \square, F)$ so dass
 - Q , endliche Menge von **Zuständen**
 - Σ , endliches **Eingabealphabet**
 - Γ , endliches **Bandalphabet**, mit $\Sigma \subset \Gamma$
 - $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, N\}$, (**partielle**) **Übergangsfunktion**
 - $q_0 \in Q$, **Startzustand**
 - $\square \in \Gamma \setminus \Sigma$, **Leerzeichen**
 - $F \subseteq Q$, Menge der **akzeptierenden** oder **Endzustände**
- **Annahme:** $\delta(q, a)$ ist **nicht definiert** für alle $q \in F$ und $a \in \Gamma$
- Eine **nichtdeterministische Turingmaschine** hat eine Übergangsfunktion $\delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R, N\})$, e.g. $\delta(q, first(\beta)) \ni (q', c, D)$
- **Bedeutung von $\delta(q, a) = (q', b, D)$:** Wenn sich M im Zustand q befindet und auf dem Band a liest, so geht M im nächsten Schritt in den Zustand q' über, überschreibt a mit b und bewegt danach den Schreib-/Lesekopf nach **rechts** (falls $D = R$), nach **links** (falls $D = L$), oder **nicht** (falls $D = N$).



- Eine **Konfiguration** einer Turingmaschine ist ein Tripel $(\alpha, q, \beta) \in \Gamma^* \times Q \times \Gamma^*$, dies modelliert
 - **Bandinhalt:** $\dots \square \alpha \beta \square \dots$
 - **Zustand:** q
 - **Kopf auf dem ersten Zeichen von β**
- **Startkonfiguration** der Turingmaschine bei Eingabe $w \in \Sigma^*$ ist (ϵ, q_0, w)
- Die **Berechnung der TM M** wird als Relation \rightarrow_M auf Konfigurationen formalisiert. Falls $\delta(q, first(\beta)) = (q', c, D)$:

$$(\alpha, q, \beta) \rightarrow_M \begin{cases} (\alpha, q', c \text{ rest}(\beta)) & \text{falls } D = N \\ (\alpha c, q', \text{rest}(\beta)) & \text{falls } D = R \\ (\text{butlast}(\alpha), q', \text{last}(\alpha) c \text{ rest}(\beta)) & \text{falls } D = L \end{cases}$$

- Eine Turingmaschine M **akzeptiert die Sprache** $L(M) = \{w \in \Sigma^* \mid \exists q \in F, \alpha, \beta \in \Gamma^*. (\epsilon, q_0, w) \rightarrow_M^* (\alpha, q, \beta)\}$
- Eine Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ heisst **Turing-berechenbar** gdw. es eine Turingmaschine M gibt, so dass für alle $n_1, \dots, n_k, m \in \mathbb{N}$ gilt
 - $f(n_1, \dots, n_k) = m \Leftrightarrow \exists r \in F. (\epsilon, q_0, \text{bin}(n_1) \# \text{bin}(n_2) \# \dots \# \text{bin}(n_k)) \rightarrow_M^* (\square \dots \square, r, \text{bin}(m) \square \dots \square)$
 - $\text{bin}(n)$ ist die Binärdarstellung der Zahl n
- Eine Funktion $f : \Sigma^* \rightarrow \Sigma^*$ heisst **Turing-berechenbar** gdw. es eine Turingmaschine M gibt, so dass für alle $u, v \in \Sigma^*$ gilt
 - $f(u) = v \Leftrightarrow \exists r \in F. (\epsilon, q_0, u) \rightarrow_M^* (\square \dots \square, r, v \square \dots \square)$
- **Halten/Terminieren von TM:**
 - Eine TM **hält** wenn sie eine Konfiguration $(\alpha, q, a\beta)$ erreicht und $\delta(q, a)$ nicht definiert oder (bei nichtdeterministischer TM) $\delta(q, a) = \emptyset$
 - Nach Annahme **hält eine TM immer, wenn sie einen Endzustand erreicht.**
 - Eine TM kann aber auch halten, **bevor sie einen Endzustand erreicht.**
- Zu jeder nichtdeterministischen TM N gibt es eine deterministische TM M mit $L(N) = L(M)$
- Die von Turingmaschinen akzeptierten Sprachen sind genau die Typ-0-Sprachen der Chomsky Hierarchie.
- **k-Band-Turingmaschine:**



Die k Köpfe sind völlig unabhängig voneinander:

$$\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, N\}^k$$

- Jede k -Band-TM kann effektiv durch eine 1-Band-TM simuliert werden
 $\Leftrightarrow n$ Schritte von M lassen sich durch $O(n^2)$ Schritte von M' simulieren
- Seien $M_i = (Q_i, \Sigma, \Gamma, \delta_i, q_i, \square, F_i)$, $i = 1, 2$
 - **Sequentielle Komposition:**
 - * $M := M_1 \rightarrow M_2$
 - * $M = (Q_1 \cup Q_2, \Sigma, \Gamma_1 \cup \Gamma_2, \delta, q_1, \square, F_2)$ mit $Q_1 \cap Q_2 = \emptyset$

- * $\delta = \delta_1 \cup \delta_2 \cup \{(f_1, a) \mapsto (q_2, a, N) \mid f_1 \in F_1, a \in \Gamma_1\} \rightarrow$ Wenn man im Zustand f_1 ist und a liest dann kommt man zum Anfangszustand von M_2 und macht weiter wie M_2
- **Fallunterscheidung:** Sind f_1 und f_2 Endzustände von M
 - * $M \xrightarrow{f_1} M_1 \rightarrow$
 $\downarrow f_2$
 M_2
 \downarrow
 - * M macht eine Fallunterscheidung, indem sie entweder von f_1 zu M_1 oder von f_2 zu M_2 übergeht
 - * "Band $i = 0$?"-TM: ja und $nein$ sind Endzustände
 - $\delta(q_0, 0) = (q_0, 0, R)$
 - $\delta(q_0, \square) = (ja, \square, L)$
 - $\delta(q_0, a) = (nein, a, N)$ für $a \neq 0, \square$
- **Schleife:** Analog zur Fallunterscheidung
 - * Band $i = 0$? \xrightarrow{ja}
 $\uparrow \downarrow nein$
 M
 - * Diese TM verhält sich wie *while* Band $i \neq 0$ *do* M

5.3 WHILE-Programme

- Strukturierte Programme mit *while*-Schleifen
- **Syntax von WHILE-Programmen:**
 - $x_i = x_j + n$
 - $x_i = x_j - n$
 - $x_i = x_j$
 - $x_i = n$
 - $x_i = x_j + x_k$ mit $i \neq k$
 - $x_i = x_j * x_k$ mit $i \neq j, k$
 - DIV, MOD
 - $IF\ x = n\ THEN\ ELSE\ END$
 - $WHILE\ x_k \neq 0\ DO\ END$
- Eine **totale Funktion** $f : \mathbb{N}^k \rightarrow \mathbb{N}$ ist **WHILE-berechenbar** gdw. es ein **WHILE-Programm** P gibt, so dass für alle $n_1, \dots, n_k \in \mathbb{N}$:

P , gestartet mit n_1, \dots, n_k in x_1, \dots, x_k terminiert mit $f(n_1, \dots, n_k)$ in x_0 (Endergebnis)

- Eine **partielle Funktion** $f : \mathbb{N}^k \rightarrow \mathbb{N}$ ist **WHILE-berechenbar** gdw. es ein **WHILE-Programm** P gibt, so dass für alle $n_1, \dots, n_k \in \mathbb{N}$:

P , gestartet mit n_1, \dots, n_k in x_1, \dots, x_k :

- **terminiert** mit $f(n_1, \dots, n_k)$ in x_0 , falls $f(n_1, \dots, n_k)$ **definiert** ist
- **terminiert nicht**, falls $f(n_1, \dots, n_k)$ **undefiniert** ist
- Jede WHILE-berechenbare Funktion ist auch Turing-berechenbar \Leftrightarrow Jede Programmvariable wird auf einem eigenen Band gespeichert, diese mehrband-TM wird dann von 1-Band-TM simuliert werden
- Jedes WHILE-Programm ist zu einem WHILE-Programm mit genau einer WHILE-Schleife äquivalent
- Beispiel:

```

while  $x_1 \neq 0$  do
   $x_2 := x_1 \text{ MOD } 2$ 
   $x_1 := x_1 \text{ DIV } 2$ 
  if  $x_1 = 0$  then
     $x_0 := 1$ 
  else
    if  $x_2 = 0$  then
       $x_0 = x_0^2$ 
    else
       $x_0 := 0$ 
       $x_1 := 0$ 
    end
  end
end
end

```

5.4 GOTO-Programme

- Assembler
- Eine Sequenz von markierten Anweisungen: $M_1 : A_1, \dots, M_k : A_k$
- **Syntax (Anweisungen) von GOTO-Programmen:**
 - $x_i = x_j + n$
 - $x_i = x_j - n$
 - *GOTO* M_i
 - *IF* $x_i = n$ *GOTO* M_j
 - *HALT*
- Jedes WHILE-Programm kann durch ein GOTO-Programm simuliert werden
- Jedes GOTO-Programm kann durch ein WHILE-Programm simuliert werden

- WHILE- und GOTO-Berechenbarkeit sind äquivalent
- Jede TM kann durch ein GOTO-Programm simuliert werden
- Beispiel:

```

M1:  x0 := x0 + 1
M2:  if x1 = 1 goto M16
M3:  x2 := x1
M4:  x3 := 0
M5:  x4 := 4
M6:  if x2 = 1 goto M12
M7:  if x2 = 0 goto M14
M8:  x2 := x2 - 2
M9:  x3 := x3 + 1
M10: x4 := x4 + 6
M11: goto M6
M12: x1 := x4
M13: goto M1
M14: x1 := x3
M15: goto M1
M16: HALT

```

5.5 Unterscheidbarkeit des Halteproblems