Formale Systeme:

1. Formale Sprachen & Grammatiken:

Plan

- Grammatiken
- Reguläre Sprachen
  - DFA und NFA
  - Umformungen, Abschlusseigenschaften, Minimalisierung
  - Pumping-Lemma
- Kontextfreie Sprachen
  - Wortproblem (CYK)
  - Pumping-Lemma
  - Kellerautomaten
- Typ 1 und Typ 0
  - Turingmaschinen
  - (Un-)Entscheidbarkeit
- Aussagenlogik
  - Syntax und Semantik
  - Algorithmen zum logischen Schließen
  - Die Komplexitätsklasse NP

- **Lexikalische Analyse:** Erzeugt Folge von Tokens (Bedeutungseinheiten)
  - Zur Weiterverarbeitung meist Mitnahme von Informationen wie Name
- **Syntaktische Analyse:** Erzeugt Strukturbaum
- **Semantische** Analyse: Coderzeugung / Optimierung

- **Formale Sprachen:** Mengen von String, wie Bezeichner, Programme, etc.
  - ➢ Eine Menge von Wörtern über $\sum$
  - Beschreibbar durch Automaten, Grammatiken, Syntaxdiagrammen

- **Alphabete & Wörter:**
  - Sei $w = a_1 \dots a_n$ ein Wort der Länge n.
    - Ein Präfix von w ist ein Wort $a_1 \dots a_i$ mit $0 \le i \le n$
    - Ein Suffix von w ist ein Wort $a_j \dots a_n$ mit $0 \le j \le n$
    - Ein Infix von w ist ein Wort $a_i \dots a_j$ mit $0 \le i \le j \le n$

    Beispiel: Das Wort `Staubecken` hat ein Präfix `Staub`, ein Suffix `ecken` und ein Infix `taube` (und viele andere mehr).
  - - Vereinigung: $L_1 \cup L_2$
    - Schnitt: $L_1 \cap L_2$
    - Komplement: $\overline{L} = \Sigma^* \setminus L$ (nur sinnvoll, wenn das Alphabet Σ klar festgelegt ist!)
    - Produkt: $L_1 \circ L_2 = \{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}$
    - Potenz: $L^0 = \{\epsilon\}$ und $L^{n+1} = L \circ L^n$
    - Kleene-Abschluss: $L^* = L^0 \cup L^1 \cup L^2 \cup \dots = \bigcup_{i \ge 0} L^i$
  - Für Sprachen L_1, L_2, K über einem gemeinsamen Alphabet Σ gilt:
    - $L_1 \cap L_2 = L_2 \cap L_1$ (Kommutativgesetz $\cap$)
    - $L_1 \cup L_2 = L_2 \cup L_1$ (Kommutativgesetz $\cup$)
    - $K \cup (L_1 \cap L_2) = (K \cup L_1) \cap (K \cup L_2)$ (Distributivgesetz $\cup$ über $\cap$)
    - $K \cap (L_1 \cup L_2) = (K \cap L_1) \cup (K \cap L_2)$ (Distributivgesetz $\cap$ über $\cup$)
    - $\overline{L_1 \cup L_2} = \overline{L_1} \cap \overline{L_2}$ (De Morgansches Gesetz 1)
    - $\overline{L_1 \cap L_2} = \overline{L_1} \cup \overline{L_2}$ (De Morgansches Gesetz 2)

  Für Sprachen L_1, L_2, K über einem gemeinsamen Alphabet Σ gilt:
  - $K \circ (L_1 \cup L_2) = (K \circ L_1) \cup (K \circ L_2)$
    $(L_1 \cup L_2) \circ K = (L_1 \circ K) \cup (L_2 \circ K)$
    } Distributivgesetze $\circ/\cup$
  - Achtung: Distributivgesetze $\circ/\cap$ gelten nicht!
  - $K = K \circ \{\epsilon\} = \{\epsilon\} \circ K$ ($\{\epsilon\}$ neutrales Element zu $\circ$)
  - $K \circ \emptyset = \emptyset \circ K = \emptyset$ ($\emptyset$ absorbierendes Element zu $\circ$)
  - $K^+ = K^* \circ K = K \circ K^*$
  - $K^* = K^+ \cup \{\epsilon\} = (K \setminus \{\epsilon\})^*$

- **Kardinalitäten:**
  - ➢ Über jedes Alphabet gibt es **abzählbar** viele Wörter und **überabzählbar** viele Sprachen
  - Fast alle Sprachen können nicht endlich beschrieben werden

- **Grammatiken:**
  - ➢ Eine Grammatik G ist ein 4-Tupel $G = \langle V, \sum, P, S \rangle$ bestehend aus:

- V: Menge von Variablennamen
- Σ: Alphabet, disjunkt zu V
- P: Menge von Produktionsregeln der Form w → v für beliebige Wörter w und v über Σ ∪ V, wobei w mindestens eine Variable enthält (d.h. w, v ∈ (Σ ∪ V) ∗ und w < Σ ∗)
- S: Startvariable aus V

Die Elemente von Σ nennt man auch Terminalsymbole, die Elemente von V analog Nicht-Terminalsymbole.

- Ableitungen:

- **1-Schritt-Ableitungsrelation**: binäre Relation ⇒ zwischen Wörtern aus $(V \cup \sum)^¿$, so dass u ⇒ v genau dann wenn:
  $u = w_1 x w_2$ und $v = w_1 y w_2$ mit Regel $x \to y \in P$
- **Ableitungsrelation** $\Rightarrow^¿$: reflexive, transitive Abschluss von ⇒, das heißt u ⇒∗ v genau dann wenn:
  $u = w_1 \Rightarrow w_2 \Rightarrow \ldots \Rightarrow w_{n-1} \Rightarrow w_n$

- Sprache einer Grammatiken:

- Die aus Grammatik erzeugte Sprache $L(G) := \left\{ w \in \sum^¿ \middle| S \Rightarrow^¿ w \right\}$ besteht aus allen Wörtern über $\sum$, die man von S ableiten kann

- Chomsky-Hierarchie:

- unterteilt Grammatiken & Sprachen in vier Stufen…
- Setzt vorraus, dass mind. Eine Variable in w existiert, sonst keine Grammatik
  - Typ 0: beliebige Grammatiken
  - Typ 1: kontextsensitive Grammatiken/Sprachen:
    Alle Regeln w → v erfüllen die Bedingung | w| ≤ |v|.
  - Typ 2: kontextfreie Grammatiken/Sprachen:
    Alle Regeln haben die Form A → v für eine Variable A.
  - Typ 3: reguläre Grammatiken/Sprachen:
    Alle Regeln haben eine der Formen $A \to cB \mid A \to c \mid A \to \varepsilon$

Typische kontextsensitive Regel:

$aBc \to avc$

„B kann im Kontext von a und c durch v ersetzt werden."

Typische kontextfreie Regel:

$B \to v$

„B kann in jedem Kontext durch v ersetzt werden."

Erzeugung ϵ-freier Grammatiken

Der folgende Algorithmus kann ϵ-Regeln eliminieren.

Eingabe: kontextfreie Grammatik (CFG) $G = \langle V, \Sigma, P, S \rangle$
Ausgabe: ϵ-freie CFG $G' = \langle V', \Sigma, P', S' \rangle$ mit $\mathbf{L}(G') = \mathbf{L}(G)$

- Initialisiere $P' := P$ und $V' := V$
- Berechne $V_\epsilon = \{A \in V \mid A \Rightarrow^* \epsilon\}$ (Details dazu später)
- Entferne alle ϵ-Regeln aus P'
- Solange es in P' eine Regel $B \to xAy$ gibt, mit

  $A \in V_\epsilon \qquad |x| + |y| \geq 1 \qquad B \to xy \notin P'$

  wähle eine solche Regel und setze $P' := P' \cup \{B \to xy\}$
- Falls $S \in V_\epsilon$ dann definiere ein neues Startsymbol $S' \notin V$, setze $V' := V' \cup \{S'\}$ und $P' := P' \cup \{S' \to S, S' \to \epsilon\}$. Falls $S \notin V_\epsilon$, dann verwenden wir einfach $S' := S$ als Startsymbol.

(CFG steht für context-free grammar und wird oft als Abkürzung verwendet)

Berechnung von V_ϵ

Die Variablen $A \in V$ mit $A \Rightarrow^* \epsilon$ kann man leicht rekursiv finden:

Eingabe: kontextfreie Grammatik $G = \langle V, \Sigma, P, S \rangle$
Ausgabe: V_ϵ für diese Grammatik

- Intialisiere $V_\epsilon := \{A \mid A \to \epsilon \in P\}$
- Solange es eine Regel $B \to A_1 \ldots A_n \in P$ gibt, so dass gilt

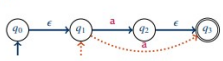
wähle

Beispiel:

Initialisieru
Schritt 1:
Schritt 2:

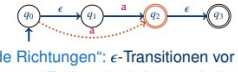
ϵ-NFA: Variationen

Die Konstruktion im Beweis „verlängert" normale Übergänge „nach rechts" durch Anhängen von ϵ-Transitionen.

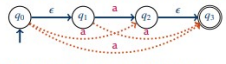


Man kann alternative Konstruktionen für den Beweis angeben:

- „Verlängerung nach links": ϵ-Transitionen vor normalen Übergängen; Anfangszustände werden beibehalten; dafür werden Endzustände mit ϵ-Transitionen erweitert.



- „Verlängerung in beide Richtungen": ϵ-Transitionen vor und nach normalen Übergängen; Anfangs- und Endzustände können beibehalten werden.*



*) Ausnahme: Anfangszustände mit ϵ-Pfad zu einem Endzustand werden Endzustand.

- **Endliche Automaten:**
  - Ein **deterministischer (DFA)** endl. Automat ist ein Tupel
    $M = \langle Q, \sum, \delta, q_0, F \rangle$ Q: Menge an Zuständen
    δ: Übergangsfunktion (partiell/total)
    $q_0 \in Q$: Startzustand
    $F \subseteq Q$: Menge Endzuständen

    Deterministisch: Von einem Zustand aus gehend müssen die Symbole der Übergänge unterschiedlich sein -> es dürfen bspw. Keine Übergänge mit bspw. „a“ auf sich und ein anderen Übergang zeigen.

  - **Sprache:** $L(M) = \{ w \in \sum ¿ \mid \delta(Q_0, w) \cap F \neq \varnothing \}$, für die M einen akzept. Lauf hat.

  - **Von Automat zum reg. Ausdruck:**

    Für $\mathcal{M} = \langle Q, \Sigma, \delta, q_0, F \rangle$ ergibt sich $G_{\mathcal{M}} = \langle V, \Sigma, P, S \rangle$ wie folgt:
    - $V := Q$
    - $S := q_0$
    - P besteht aus den folgenden Produktionsregeln:

      $q \rightarrow aq'$ falls $\delta(q, a) = q'$
      $q \rightarrow a$ falls $\delta(q, a) \in F$
      $q_0 \rightarrow \epsilon$ falls $q_0 \in F$

  - Ein **nicht-deterministischer (NFA)** endl. Automat ist ein Tupel
    $M = \langle Q, \sum, \delta, Q_0, F \rangle$
    δ: Übergangsfunktion, 2^Q... Potenzmenge von Q
    Q_0: Menge mögl. Startzustände

  - **Läufe:**

    Ein Lauf eines NFA $\mathcal{M} = \langle Q, \Sigma, \delta, Q_0, F \rangle$ für ein Wort $w = \sigma_1 \cdots \sigma_n$ ist eine Folge von Zuständen $q_0 \ldots q_m$, so dass gilt:
    - $q_0 \in Q_0$,
    - $q_{i+1} \in \delta(q_i, \sigma_{i+1})$ für alle $0 \leq i < m$,
    - (1) $m = |w| = n$ oder (2) $m < n$ und $\delta(q_m, \sigma_{m+1}) = \emptyset$.

    Ein Lauf heißt akzeptierend, falls $m = n$ und $q_n \in F$.
    Andernfalls heißt der Lauf verwerfend.

  - **Verallgemeinerte Übergangsfunktion:**

Zuerst erweitern wir δ auf Mengen von Zuständen:

Für eine Zustandsmenge $R \subseteq Q$ und ein Terminalsymbol a sei

$$\delta^{\cup}(R, a) = \bigcup_{q \in R} \delta(q, a).$$

Dann erweitern wir $\delta^{\cup}$ von einzelnen Symbolen zu beliebigen Wörtern:

Für eine Zustandsmenge $R \subseteq Q$ und ein Wort $w \in \Sigma^*$ sei $\delta^*(R, w)$ die Menge aller Zustände, die man erreichen kann, wenn man in einem Zustand aus R beginnt und das Wort w einliest. Formal:

- $\delta^*(R, \epsilon) = R$
- $\delta^*(R, av) = \delta^*(\delta^{\cup}(R, a), v)$

- **Potenzmengenkonstruktion:**
  - Potenzmengen DFS $M_{DFA} := \langle Q_{DFA}, \sum, \delta_{DFA}, q_0, F_{DFA} \rangle \; Q_{DFA} = 2^Q$
    $\delta_{DFA}(R, a) := ¿\, q \in R ¿\, \delta(q, a)$
    $q_0 = Q_0$
    $F_{DFA} := \{ R \in 2^Q \mid R \cap F \neq \varnothing \}$

- **Abschlusseigenschaften:**
  - **Vereinigungsautomat:**
    - Automat mit den Unterschieden beider
    - **Schnittautomat (∩):** wie Quotientenautomat

      $M \oplus M := \{ Q_1 \cup Q_2, \sum, \delta_{12}, Q_{0,1} \cup Q_{0,2}, F_1 \cup F_2 \}$ mit
      $q_{12}(q, a) := \begin{cases} q_1(q, a), falls\, q \in Q_1 \\ q_2(q, a), falls\, q \in Q_2 \end{cases}$

  - **Produktautomat:**
    $M \otimes M := \{ Q_1 \times Q_2, \sum, \delta, Q_{0,1} \times Q_{0,2}, F_1 \times F_2 \}$ mit
    $\delta(\langle q_1, q_2 \rangle, a) = \delta_1(q_1 a) \times \delta_2(q_2 a)$

  - **Komplementoperator:** $\overline{M} := \langle Q, \sum, \delta, q_0, Q ¿ \rangle$

  - **Konkatenation:**
    - Endzustände werden mit Startzuständen des anderen verbunden

      $M \odot M := \langle Q_1 \cup Q_2, \sum, \delta, Q_{0,1}, F_2 \rangle$ mit
      $\delta(q, a) := \begin{cases} \delta_1(q, a), falls\, q \in Q_1 \\ \delta_2(q, a), falls\, q \in Q_2 \end{cases}$ und
      $\delta(q, e) := \begin{cases} Q_{0,2}, falls\, q \in F_1 \\ \varnothing, sonst \end{cases}$

  - **Kleene Stern:**
    - Von allen Finalzuständen gehen ε-Übergänge zum Startzustand

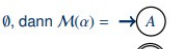
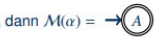
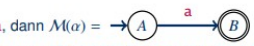
Gegeben sei ein NFA $\mathcal{M} = \langle Q, \Sigma, \delta, Q_0, F \rangle$. Der Automat $\mathcal{M}^*$ ist ein ϵ-NFA gegeben durch $\langle Q', \Sigma, \delta', Q_0', F' \rangle$ wobei

- $Q' = Q \cup \{q_\epsilon\}$ (wobei $q_\epsilon \notin Q$ o.B.d.A.)
- $\delta'(q, a) = \delta(q, a)$ für alle $q \in Q, a \in \Sigma$ und $\delta'(q_f, \epsilon) = Q_0$ für alle $q_f \in F$
- $Q_0' = Q_0 \cup \{q_\epsilon\}$
- $F' = F \cup \{q_\epsilon\}$

- **Reguläre Ausdrücke:**
  - Menge ist induktiv definiert…
    - $\varnothing, \varepsilon$
    - $\forall a \in \sum$
    - Sei $a, b \in \sum$ reguläre Ausdrücke, dann sind es auch: (ab) , $(a|b)$,$(a)^{¿}$
  - **Semantik:**
    - Menge (Sprache) ist induktiv definiert…
      - $L(\varnothing) = \varnothing, L(\varepsilon) = \{\varepsilon\}, L(a) = \{a\}$ für $\forall a \in \sum$
      - $L((ab)) = L(a) \circ L(b)$
      - $L((a \vee b)) = L(a) \cup L(b)$
      - $L((a)^{¿}) = L(a)^{¿}$
  - **Regulärer Ausdruck -> NFA:**
    - **Rekursive Komposition $\varepsilon - NFA$ zu NFA:**

Für einen Ausdruck α definieren wir induktiv den ϵ-NFA $\mathcal{M}(\alpha)$:

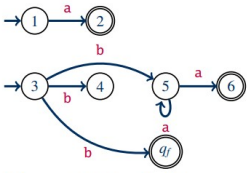
Grundfälle:

- Wenn $\alpha = \emptyset$, dann $\mathcal{M}(\alpha) =$ 
- Wenn $\alpha = \epsilon$, dann $\mathcal{M}(\alpha) =$ 
- Wenn $\alpha = a$, dann $\mathcal{M}(\alpha) =$ 

Rekursive Fälle: Wir bezeichnen mit $\text{elim}_\epsilon(\mathcal{M})$ den NFA, der aus einem ϵ-NFA $\mathcal{M}$ durch Eliminierung der ϵ-Übergänge entsteht.

- Wenn $\alpha = (\beta\gamma)$, dann $\mathcal{M}(\alpha) = \text{elim}_\epsilon(\mathcal{M}(\beta) \odot \mathcal{M}(\gamma))$.
- Wenn $\alpha = (\beta \mid \gamma)$, dann $\mathcal{M}(\alpha) = \mathcal{M}(\beta) \oplus \mathcal{M}(\gamma)$.
- Wenn $\alpha = (\beta)^*$, dann $\mathcal{M}(\alpha) = \text{elim}_\epsilon(\mathcal{M}(\beta)^*)$.

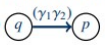
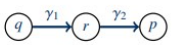
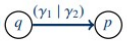
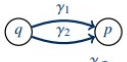
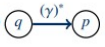
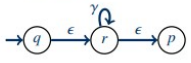
Kompositioneller Ansatz



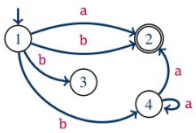
    - **Explizite Konstruktion von NFA:**

Initialisierung: $\mathcal{M}_\alpha =$ 

Solange es in $\mathcal{M}_\alpha$ Übergänge $q \xrightarrow{\beta} p$ gibt, die mit einem Ausdruck $\beta \notin \{\epsilon\} \cup \Sigma$ beschriftet sind, wende eine der folgenden Regeln an:

- Ersetze  durch 
- Ersetze  durch 
- Ersetze  durch 

Expliziter Ansatz



  - **NFA/DFA -> regulärer Ausdruck:**
    - **Ersetzungsmethode:**
      1. Vereinfache den Automaten *(entferne offensichtlich unnötige/unerreichbare Zustände)*
      2. Bestimme das Gleichungssystem

Für einen NFA $\mathcal{M} = \langle Q, \Sigma, \delta, Q_0, F \rangle$ betrachten wir die folgenden Gleichungen für Ausdrücke α_q mit $q \in Q$.

- Für jeden Zustand $q \in Q \setminus F$:

$$\alpha_q \equiv \sum_{a \in \Sigma} \sum_{p \in \delta(q,a)} a\alpha_p$$

- Für jeden Zustand $q \in F$:

$$\alpha_q \equiv \epsilon \mid \sum_{a \in \Sigma} \sum_{p \in \delta(q,a)} a\alpha_p$$

(eine Gleichung pro Zustand)

3. Löse das Gleichungssystem
   (durch Einsetzen und Ardens Lemma)
4. Gib den Ausdruck für die Sprache des NFA/DFA an
   (Alternative der Ausdrücke für alle Anfangszustände)

Lemma (Arden): Aus $\alpha \equiv \beta\alpha \mid \gamma$ mit $\epsilon \notin \mathbf{L}(\beta)$ folgt $\alpha \equiv \beta^*\gamma$.

- **Dynamisch Programmierung:**
  - Sei $i,j \in \{1,\ldots,n\}$ und $k \in \{0,\ldots,n\}$ mit $L^k[i,j]$ und $a^{k+1}[i,j] := a^k[i,j] \vee \left(a^k[i,k+1]\left(a^k[k+1,k+1]\right)^{¿} a^k[k+1,j]\right)$
    - i. $a^n[i,j] := a_{qi,qj}$
    - ii. $a^0[i,j] := \begin{cases} a_1 \vee \ldots \vee a_m, i \neq j \\ a_1 \vee \ldots \mid a_m \mid \varepsilon, i = j \end{cases}$

- Wortprobleme:

Für reguläre Sprachen haben wir eine Reihe von Problemstellungen kennengelernt:

- Leerheitsproblem: Ist die beschriebene Sprache gleich der leeren Menge ∅?
- Inklusionsproblem: Ist eine beschriebene Sprache Teilmenge einer anderen?
- Äquivalenzproblem: Wird durch zwei Beschreibungen die selbe Sprache gegeben?
- Endlichkeitsproblem: Ist die beschriebene Sprache endlich?
- Universalitätsproblem: Ist die beschriebene Sprache Σ^*?

- Äquivalenz-Automaten:

- $q \;{}_M\; p$, wenn gilt: $L(M_p) = L(M_q)$
- 2 Zustände bilden (ggf. über Umwege) auf sich selbst ab

- **Quotientenautomat:**
  - $M_{¿} := \langle Q_{¿}, \sum, \delta, [q_0]_M, F_{¿} \rangle$
    - $Q_{¿} := \{[q_0] \mid q \in Q\}$
    - $\delta([q_0], a) := \delta([[q_0], a])$
    - $F_{¿} := \{[q_0] \mid q \in F\}$

    mit...

    - **Äquivalenzklasse:** $[q] := \{p \in Q \mid q \;\; p\}$
    - **Quotienten:** $P_{¿} := \{[p] \mid p \in P\}$

  - **Ermittlung M :**
    1. Aus $q_1 \;\; q_2$ folgt stets: $q_1 \in F$ gdw. $q_2 \in F$
    2. Wenn $q_1 \;\; q_2$, dann auch $\delta(q1,a) \sim \delta(q2,a)$

    Umgekehrt gilt...

    1. Aus $q_1 \in F$ und $q_2 < F$ folgt immer $q_1 \;{}_{NOT}\; q_2$
    2. Wenn $\delta(q_1,a) \;{}_{NOT}\; \delta(q_2,a)$, dann $q_1 \;{}_{NOT}\; q_2$

- Reduzierter Automat M_r:

- Minimaler DFA mit totalen Übergangsfunktionen

1. Entferne alle unerreichbare Zustände
2. Berechne den Quotientenautomat

- Nerode Rechtskongruenz $\simeq_R$:

- Für Wörter $u, v \in \sum ¿$ sei $u \simeq_R v$, wenn gilt: $\forall e \in \sum ¿$ gilt $ew \in L$, wenn $ve \in L$

- Satz: Myhill & Nerode:

- Eine Sprache L ist genau dann regulär, wenn $\simeq_L$ endlich viele Äquivalenzklassen hat
- DFA $M_L := \langle Q, \sum, \delta, q_0, F \rangle$ mit...

  $Q = \{ [w]_\simeq \mid w \in \sum ¿ \}$

  $q_0 = [\varepsilon]_\simeq$

  $F = \{ [w]_\simeq \mid w \in L \}$

  $\delta([w]_\simeq, a) = [wa]_\simeq$

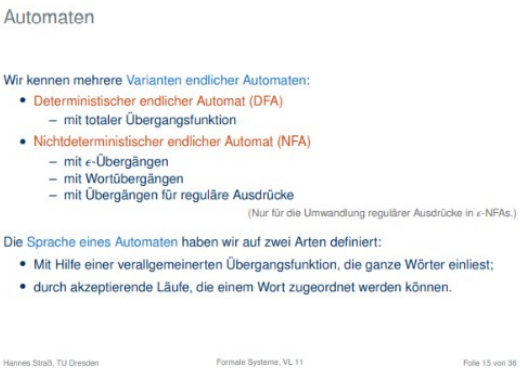
- Lemma - Pumping:

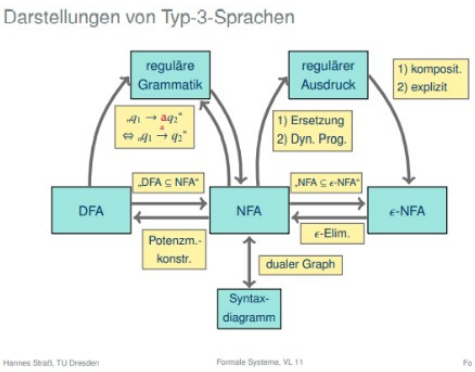
- Beweis für Regularität & Nicht-Regularität
- Für jede reguläre Sprache $\exists n \geq 0$, so dass gilt:

  $\forall z \in L$ mit $|z| \geq n$ ... $\exists z = uvw$ *(Zerlegung)* mit $|v| \geq 1$ und $|uv| \leq n$, so dass:

  - Für jede Zahl $k \geq 0$ gilt: $(uv^k w) \in L$

-



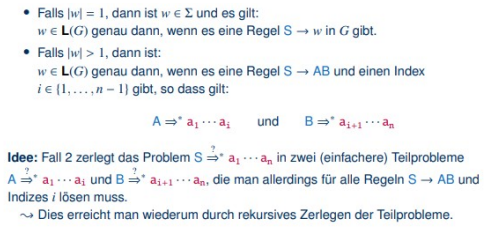


Chomsky Normalform:

- $A \to BC$ (mit $B, C \in V$) oder $A \to c$ (mit $c \in \Sigma$)
- $\varepsilon \notin L(G)$, aber es exist. Höchst. 1 Regel der Form $S \to \in$

  - **Umwandlung:**
    1. Eliminierung von ϵ-Regeln
    2. Eliminierung von Kettenregeln *(Form: $A \to B$)*
    3. Extrahiere Regeln der Form A → c, so dass alle anderen Regeln B → w keine Terminale mehr in w enthalten
    4. Zerlege Regeln der Form A → B1 · · · Bn für n > 2, so dass solche Regeln nur noch mit n = 2 auftauchen

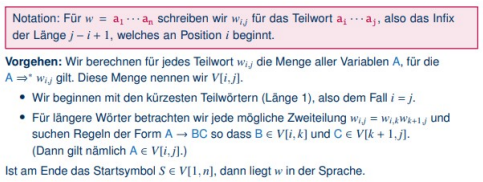
- **CYK Algorithmus:**
  - Zum Prüfen ob ein Wort zu einer bestimmten kontextfreien Sprache gehört



- Falls $|w| = 1$, dann ist $w \in \Sigma$ und es gilt:
  $w \in \mathbf{L}(G)$ genau dann, wenn es eine Regel $S \to w$ in G gibt.
- Falls $|w| > 1$, dann ist:
  $w \in \mathbf{L}(G)$ genau dann, wenn es eine Regel $S \to AB$ und einen Index $i \in \{1, \dots, n-1\}$ gibt, so dass gilt:

$$A \Rightarrow^* a_1 \cdots a_i \quad \text{und} \quad B \Rightarrow^* a_{i+1} \cdots a_n$$

Idee: Fall 2 zerlegt das Problem $S \overset{?}{\Rightarrow}^* a_1 \cdots a_n$ in zwei (einfachere) Teilprobleme $A \overset{?}{\Rightarrow}^* a_1 \cdots a_i$ und $B \overset{?}{\Rightarrow}^* a_{i+1} \cdots a_n$, die man allerdings für alle Regeln $S \to AB$ und Indizes i lösen muss.
$\rightsquigarrow$ Dies erreicht man wiederum durch rekursives Zerlegen der Teilprobleme.



Notation: Für $w = a_1 \cdots a_n$ schreiben wir $w_{i,j}$ für das Teilwort $a_i \cdots a_j$, also das Infix der Länge $j - i + 1$, welches an Position i beginnt.

Vorgehen: Wir berechnen für jedes Teilwort $w_{i,j}$ die Menge aller Variablen A, für die $A \Rightarrow^* w_{i,j}$ gilt. Diese Menge nennen wir $V[i,j]$.

- Wir beginnen mit den kürzesten Teilwörtern (Länge 1), also dem Fall $i = j$.
- Für längere Wörter betrachten wir jede mögliche Zweiteilung $w_{i,j} = w_{i,k} w_{k+1,j}$ und suchen Regeln der Form $A \to BC$ so dass $B \in V[i,k]$ und $C \in V[k+1,j]$.
  (Dann gilt nämlich $A \in V[i,j]$.)

Ist am Ende das Startsymbol $S \in V[1,n]$, dann liegt w in der Sprache.

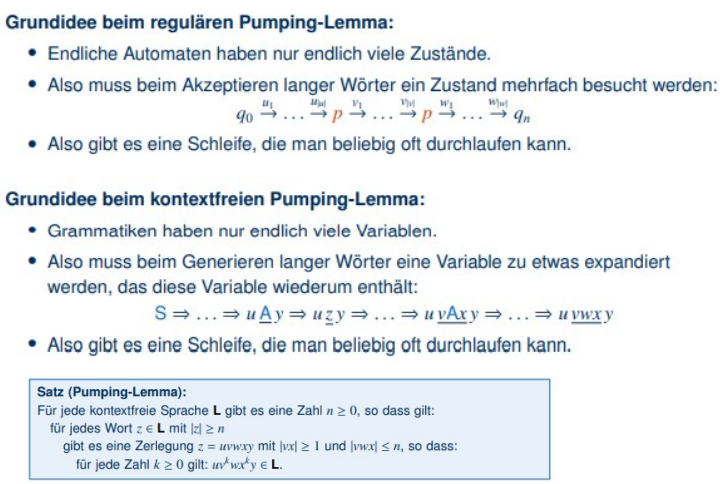
-

Kontextfreie Sprachen:

Eine kontextfreie Grammatik (oder Typ-2-Grammatik oder CFG) enthält nur Regeln der Form $A \to v$, wobei A eine Variable ist.

Eine Sprache ist kontextfrei (oder Typ 2), wenn sie durch eine kontextfreie Grammatik dargestellt werden kann.

- **Pumping:**



Grundidee beim regulären Pumping-Lemma:

- Endliche Automaten haben nur endlich viele Zustände.
- Also muss beim Akzeptieren langer Wörter ein Zustand mehrfach besucht werden:
  $q_0 \xrightarrow{u_1} \dots \xrightarrow{u_{|u|}} p \xrightarrow{v_1} \dots \xrightarrow{v_{|v|}} p \xrightarrow{w_1} \dots \xrightarrow{w_{|w|}} q_n$
- Also gibt es eine Schleife, die man beliebig oft durchlaufen kann.

Grundidee beim kontextfreien Pumping-Lemma:

- Grammatiken haben nur endlich viele Variablen.
- Also muss beim Generieren langer Wörter eine Variable zu etwas expandiert werden, das diese Variable wiederum enthält:
  $S \Rightarrow \dots \Rightarrow u\underline{A}y \Rightarrow u\underline{z}y \Rightarrow \dots \Rightarrow u\underline{vAx}y \Rightarrow \dots \Rightarrow u\underline{vwx}y$
- Also gibt es eine Schleife, die man beliebig oft durchlaufen kann.

Satz (Pumping-Lemma):
Für jede kontextfreie Sprache $\mathbf{L}$ gibt es eine Zahl $n \geq 0$, so dass gilt:
für jedes Wort $z \in \mathbf{L}$ mit $|z| \geq n$
gibt es eine Zerlegung $z = uvwxy$ mit $|vx| \geq 1$ und $|vwx| \leq n$, so dass:
für jede Zahl $k \geq 0$ gilt: $uv^k wx^k y \in \mathbf{L}$.

- **Abschluss** nur unter $\cup, \circ, *$¿
- **Vereinigungsgrammatik:**
  $G_1 \dot{\cup} G_2 := \langle V_1 \cup V_2 \cup \{S\}, \sum, P_1 \cup P_2 \cup \{S \to S_1 | S_2\}, S \rangle$
- **Konkatenatiosgrammatik:**
  $G_1 \circ G_2 := \langle V_1 \cup V_2 \cup \{S\}, \sum, P_1 \cup P_2 \cup \{S \to S_1 S_2\}, S \rangle$
- **Kleenegrammatik:** $G^{\dot{*}} := \langle V \cup \{S\}, \sum, P_1 \cup P_2 \cup \{S \to \epsilon | S_1 S\}, S \rangle$

Keller *(Stack)* sind Stapel:

- Als zusätzliche Speicher in endlichen Automaten
- Ist eine Datenstruktur, die eine Liste von Einträgen speichert

- **Zugriffsoperationen:**
  i. push: Ein Element wird oben auf dem Stapel abgelegt
  ii. pop: oberste Element wird vom Stapel entfernt und zurückgeliefert

$\Longrightarrow$ *Keller sind demnach LIFO-Speicher (last-in/first-out)*

Zur Vereinfachung der folgenden Definition schreiben wir Σ_ϵ für $\Sigma \cup \{\epsilon\}$ (analog für Γ_ϵ).

- **Kellerautomat:**

Ein Kellerautomat (international: „PDA“, „Pushdown Automaton“) $\mathcal{M}$ ist ein Tupel $\mathcal{M} = \langle Q, \Sigma, \Gamma, \delta, Q_0, F \rangle$ mit den folgenden Bestandteilen:

- Q: endliche Menge von Zuständen
- Σ: Eingabealphabet
- Γ: Kelleralphabet
- δ: Übergangsfunktion, eine totale Funktion
  $$Q \times \Sigma_\epsilon \times \Gamma_\epsilon \to 2^{Q \times \Gamma_\epsilon},$$
  wobei $2^{Q \times \Gamma_\epsilon}$ die Potenzmenge von $Q \times \Gamma_\epsilon$ ist.
- Q_0: Menge möglicher Startzustände $Q_0 \subseteq Q$
- F: Menge von Endzuständen $F \subseteq Q$

Der PDA $\mathcal{M}$ akzeptiert ein Wort w genau dann, wenn es eine Folge von Übergängen $\langle q_s, w, \epsilon \rangle \vdash^* \langle q_f, \epsilon, \gamma \rangle$ für ein $q_s \in Q_0$ und $q_f \in F$ gibt.

Die von $\mathcal{M}$ akzeptierte Sprache $\mathbf{L}(\mathcal{M})$ ist die Menge aller von $\mathcal{M}$ akzeptierten Wörter.

- **(Typ 2) Grammatik → PDA:**

$Q = \{q_s, q_l, q_f\}$ $Q_0 = \{q_s\}$ $F = \{q_f\}$ $\Gamma = \Sigma \cup V \cup \{\#\}$

Durch Entfernen der Wort-push-Operationen wie zuvor gezeigt entsteht der gesuchte PDA $\mathcal{P}_G$. □

- **PDA → (Typ 2) Grammatik:**

$w \in L_{q,r}$ genau dann, wenn $\langle q, w, \epsilon\rangle \vdash^* \langle r, \epsilon, \epsilon\rangle$ **(VL16 – Video!)**

- **Deterministischer – DPDA:**
  - δ:Übergangsfunktionen, sodass $\forall\, q \in Q, a \in \sum, A \in \Gamma$ jeweils nur eines der
    Folgenden definiert ist…
    $\delta(q,a,A)$ oder $\delta(q,a,\epsilon)$ oder $\delta(q,\varepsilon,A)$ oder $\delta(q,\varepsilon,\varepsilon)$
  - q_0: ein Startzustand $q_0 \in Q$

- Klasse der deterministisch kontextfreien Sprachen unter Komplement abgeschlossen
- **Erweiterung:**
  - Mehr Stapel: *Statt 1 mehrere Speicherstapel*
  - Warteschlangenautomaten: *Warteschlange statt Stapelspeicher verwenden*
  - Zählerautomaten: *Endl. Viele Speicherplätze für Zahlen statt Stapelspeicher*
  - While – bzw. GoTo Programme

Turing Maschine:

Eine (deterministische) Turingmaschine (DTM) ist ein Tupel $\mathcal{M} = \langle Q, \Sigma, \Gamma, \delta, q_0, F\rangle$ mit den folgenden Bestandteilen:

- Q: endliche Menge von Zuständen
- Σ: Eingabealphabet
- Γ: Arbeitsalphabet mit $\Gamma \supseteq \Sigma \cup \{\sqcup\}$
- δ: Übergangsfunktion, eine partielle Funktion
  $$Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$$
- q_0: Startzustand $q_0 \in Q$
- F: Menge von akzeptierenden Endzuständen $F \subseteq Q$

Dabei bedeutet $\delta(q,a) = \langle p, b, D\rangle$:
„Liest die TM in Zustand q unter dem Lese-/Schreibkopf ein a, dann wechselt sie zu Zustand p, überschreibt das a mit b und verschiebt den Lese-/Schreibkopf gemäß $D \in \{L, R, N\}$ (nach links, nach rechts, gar nicht)."

- Lese-&Schreiboperation gleichzeitig möglich & an jeder belieb. Adresse

- **Übergänge:**

Eine Konfiguration von $\mathcal{M}$ ist ein Wort $wqv \in \Gamma^* \circ Q \circ \Gamma^*$, wobei w den Bandinhalt vor dem Lese-/Schreibkopf, q den aktuellen Zustand und v den Bandinhalt ab dem Lese-/Schreibkopf darstellt.

Sei $wqav$ eine Konfiguration, mit $w, v \in \Gamma^*$, $a \in \Gamma$ und $q \in Q$.
Die Übergangsrelation $\vdash$ ist wie folgt definiert:

- Falls $\delta(q,a) = \langle r, b, N\rangle$, dann $wqav \vdash wrbv$. $wqav \vdash wrbv$
- Falls $\delta(q,a) = \langle r, b, R\rangle$
  - falls $v \neq \epsilon$, dann $wqav \vdash wbrv$; $wqav \vdash wbrv$
  - falls $v = \epsilon$, dann $wqav \vdash wbr\sqcup$. $wqa \vdash wbr\sqcup$
- Falls $\delta(q,a) = \langle r, b, L\rangle$
  - falls $w = xc$ mit $c \in \Gamma$, dann $wqav \vdash xrcbv$; $xcqav \vdash xrcbv$
  - falls $w = \epsilon$, dann $wqav \vdash rbv$. $qav \vdash rbv$

Mit $\vdash^*$ bezeichnen wir den reflexiven, transitiven Abschluss von $\vdash$.

- Ein Lauf ist eine max. Folge von Konfigurationen, die durch die Übergangsrelation in Beziehung stehen
- TM-Sprache wird akzeptiert, wenn sie einen endlichen Lauf hat und somit in einem Endzustand hält
- **Turing-Mächtig** ist ein Formalismus, wenn er jedes Ein-&Ausgabeverhalten jeder TM simulieren kann
- **Sprache - Satz von Rice:**
  - Sei E eine Eigenschaft von Sprachen, die für manche Turing-erkennbare Sprachen gilt und für manche Turing-erkennbare Sprachen nicht gilt.
    Dann ist das folgende Problem unentscheidbar:
    - Eingabe: Turingmaschine M
    - Ausgabe: Hat L(M) die Eigenschaft E?
  - TM Sprache = Typ 0 Sprache

- **Typ 1 Automaten:**

Lösung: Beschränkung des Arbeitsspeichers einer TM:

Ein linear beschränkte Turingmaschine (linear bounded automaton, LBA) ist eine nichtdeterministische Turingmaschine, die den Lese-/Schreibkopf nicht über das letzte Eingabezeichen hinaus bewegen kann. Versucht sie das, so bleibt der Kopf stattdessen an der letzten Bandstelle stehen.

Ein LBA kann also nur die Menge an Speicher nutzen, die durch die Eingabe belegt wird.

Die Definition der Übergangsrelation $\vdash$ für TMs wird für LBAs also wie folgt geändert:
Sei $wqav$ eine Konfiguration, mit $w, v \in \Gamma^*$, $a \in \Gamma$ und $q \in Q$; sei $\delta(q,a) = \langle r, b, R\rangle$.

- falls $v \neq \epsilon$, dann $wqav \vdash wbrv$ (wie zuvor); $wqav \vdash wbrv$
- falls $v = \epsilon$, dann $wqav \vdash wrb$ (neu). $wqa \vdash wrb$

Bsp: q 1 mit Übergang L/R -> q bleibt an Stelle

2. Aussagenlogik:

- **Atomare Aussagen** = Behauptungen, die wahr/falsch sein können

- **Syntax:**

Die Menge der aussagenlogischen Formeln ist induktiv[a] definiert:

- Jedes Atom $p \in \mathbf{P}$ ist eine aussagenlogische Formel.
- Wenn F und G aussagenlogische Formeln sind, so auch:

Syntax	Name	intuitive Bedeutung
$\neg F$	Negation	„nicht F“
$(F \wedge G)$	Konjunktion	„F und G“
$(F \vee G)$	Disjunktion	„F oder G“
$(F \to G)$	Implikation	„F impliziert G“
$(F \leftrightarrow G)$	Äquivalenz	„F ist äquivalent zu G“

[a] Das bedeutet: Die Definition ist selbstbezüglich und soll die $\subseteq$-kleinste Menge an Formeln beschreiben, die alle Bedingungen erfüllen.

- **Teilformeln:**
  - ist ein Teilwort (Infix) einer Formel, welches selbst eine Formel ist

Die Menge $\mathrm{Sub}(F)$ der Teilformeln einer Formel F ist definiert als:

$$\mathrm{Sub}(F) = \begin{cases} \{F\} & \text{falls } F \in \mathbf{P} \\ \{\neg G\} \cup \mathrm{Sub}(G) & \text{falls } F = \neg G \\ \{(G_1 \wedge G_2)\} \cup \mathrm{Sub}(G_1) \cup \mathrm{Sub}(G_2) & \text{falls } F = (G_1 \wedge G_2) \\ \{(G_1 \vee G_2)\} \cup \mathrm{Sub}(G_1) \cup \mathrm{Sub}(G_2) & \text{falls } F = (G_1 \vee G_2) \\ \{(G_1 \to G_2)\} \cup \mathrm{Sub}(G_1) \cup \mathrm{Sub}(G_2) & \text{falls } F = (G_1 \to G_2) \\ \{(G_1 \leftrightarrow G_2)\} \cup \mathrm{Sub}(G_1) \cup \mathrm{Sub}(G_2) & \text{falls } F = (G_1 \leftrightarrow G_2) \end{cases}$$

-

Eine Wertzuweisung w erfüllt genau dann eine Formel F, in Symbolen $w \models F$, wenn eine der folgenden induktiv definierten Bedingungen gilt:

- $F \in \mathbf{P}$ mit $w(F) = 1$,
- $F = \neg G$ mit $w \not\models G$,
- $F = (G_1 \wedge G_2)$ mit $w \models G_1$ und $w \models G_2$,
- $F = (G_1 \vee G_2)$ mit $w \models G_1$ oder $w \models G_2$ (oder beides),
- $F = (G_1 \to G_2)$ mit $w \not\models G_1$ oder $w \models G_2$,
- $F = (G_1 \leftrightarrow G_2)$ mit: $w \models G_1$ genau dann, wenn $w \models G_2$.

Wertzuweisung:

- **Junktoren anders ausgedrückt:**

$$F \leftrightarrow G \equiv (F \to G) \wedge (G \to F) \equiv (F \wedge G) \vee (\neg F \wedge \neg G)$$
$$F \wedge G \equiv \neg(\neg F \vee \neg G) \quad \text{(De Morgansches Gesetz)}$$
$$F \vee G \equiv \neg(\neg F \wedge \neg G) \quad \text{(De Morgansches Gesetz)}$$

$$F \uparrow G \equiv \neg(F \wedge G) \equiv \neg F \vee \neg G \quad \text{(NAND)}$$
$$F \downarrow G \equiv \neg(F \vee G) \equiv \neg F \wedge \neg G \quad \text{(NOR)}$$
$$F \oplus G \equiv \neg(F \leftrightarrow G) \quad \text{(XOR, Antivalenz)}$$
$$\top \equiv \neg p \vee p \equiv p \to p \quad \text{(Wahrheit; } p \in \mathbf{P} \text{ beliebig)}$$
$$\bot \equiv \neg p \wedge p \quad \text{(Falschheit; } p \in \mathbf{P} \text{ beliebig)}$$

Eine Formel F ist:

- unerfüllbar (oder inkonsistent), wenn sie keine Modelle hat;
- erfüllbar (oder konsistent), wenn sie mindestens ein Modell hat;
- allgemeingültig (oder eine Tautologie), wenn alle Wertzuweisungen Modelle für die Formel sind;
- widerlegbar, wenn sie nicht allgemeingültig ist.

Diese Begriffe kann man für Mengen von Formeln genauso definieren.

Zwei Formeln F und G sind genau dann semantisch äquivalent, in Symbolen $F \equiv G$, wenn sie genau dieselben Modelle haben, d.h. gdw.

für alle Wertzuweisungen w gilt: $w(F) = w(G)$

- **Ableitungen mittels Resolution:**

Ein einzelner Resolutionschritt funktioniert wie folgt:

Gegeben seien zwei Klauseln K_1 und K_2, für die es ein Atom $p \in \mathbf{P}$ gibt, sodass $p \in K_1$ und $\neg p \in K_2$. Die Resolvente von K_1 und K_2 bezüglich p ist die Klausel

$$(K_1 \setminus \{p\}) \cup (K_2 \setminus \{\neg p\}).$$

Eine Klausel R ist eine Resolvente einer Klauselmenge $\mathcal{K}$ wenn R Resolvente zweier Klauseln $K_1, K_2 \in \mathcal{K}$ ist.

Beispiel: Resolventen für die Menge $\{\{\neg A, \neg B\}, \{A, B\}, \{\neg B, \neg C\}, \{B, C\}, \{\neg C, \neg A\}, \{A, B, C\}, \{\neg B\}\}$ sind z.B.

- $\{B, \neg A\}$ aus den Klauseln $\{B, C\}$ und $\{\neg C, \neg A\}$;
- $\{A\}$ aus den Klauseln $\{A, B\}$ und $\{\neg B\}$;
- $\{B, \neg B\}$ aus den Klauseln $\{B, C\}$ und $\{\neg B, \neg C\}$.

➢ **Ziel:** Verfahren, um (Un-)Erfüllbarkeit einer KNF zu bestimmen

p bzw. -p wird aus Klauseln entfernt und die Klauseln danach

- **Resolutionskalkül:**

Resolution
Gegeben: Eine Formel $\mathcal{F}$ in Klauselform.
Gesucht: Ist $\mathcal{F}$ unerfüllbar?

(1) Finde ein Klauselpaar $K_1, K_2 \in \mathcal{F}$ mit Resolvente $R \notin \mathcal{F}$.
(2) Setze $\mathcal{F} := \mathcal{F} \cup \{R\}$.
(3) Wiederhole Schritt (1) und (2) bis keine neuen Resolventen mehr gefunden werden können.
(4) Falls $\bot \in \mathcal{F}$, dann gib „unerfüllbar" aus; andernfalls gib „erfüllbar" aus.

Beobachtung: Unerfüllbarkeit steht fest, sobald $\bot$ abgeleitet wurde.
$\rightsquigarrow$ Dann kann man das Verfahren frühzeitig abbrechen.

Erfüllbarkeit kann dagegen erst erkannt werden, wenn alle Resolventen erschöpfend gebildet worden sind.

- **Widerspruchsfreie Resolution zu Modell**

Für jedes Atom p, das in der Formel vorkommt, bestimmen wir einen Wahrheitswert $w(p)$:

Gegeben: widerspruchsfreie Klauselmenge $\mathcal{F}$ nach Resolution.

Für jedes Atom p in $\mathcal{F}$ (in beliebiger Reihenfolge):

(a) Wenn $\{p\} \in \mathcal{F}$, definiere $w(p) := 1$
(b) Wenn $\{\neg p\} \in \mathcal{F}$, definiere $w(p) := 0$
(c) Wenn weder $\{p\} \in \mathcal{F}$ noch $\{\neg p\} \in \mathcal{F}$, dann setze $w(p) := 1$ und $\mathcal{F} := \mathcal{F} \cup \{\{p\}\}$ und führe nochmals Resolution durch, bis keine weiteren Klauseln abgeleitet werden.

Die so bestimmte Wertzuweisung w ist ein Modell für $\mathcal{F}$.

- **Horn-Logik:**
  - Horn-Klausel = Klausel, mit höchstens ein nichtnegiertes Literal
  - Horn-Formel = Formel in KNF, welche nur Horn-Klauseln enthält