Informatik 4

Mitschrift der Vorlesung von Prof. K. Jansen
nach einem Script von Prof. W. Thomas

Mitschrift von www.kuertz.name

Hinweis: Dies ist **kein offizielles Script**, sondern nur eine private Mitschrift. Die Mitschriften sind teweilse **unvollständig, falsch oder inaktuell**, da sie aus dem Zeitraum 2001–2005 stammen. Falls jemand einen Fehler entdeckt, so freue ich mich dennoch über einen kurzen Hinweis per E-Mail – vielen Dank!

Klaas Ole Kürtz (klaasole@kuertz.net)

Inhaltsverzeichnis

1 Endliche Automaten

1.1 Bezeichnungen

- Σ ist *Alphabet*
- Σ^* ist die *Menge der Wörter über* Σ (das *leere Wort* wird bezeichnet mit ε)
- u, v, w seien Wörter in Σ^*
- $|w|$ ist die *Länge* des Wortes $w \in \Sigma^*$
- $|w|_a$ ist die *Anzahl* der a in $w \in \Sigma^*$ (für $a \in \Sigma$)
- *Verkettung:* $u \cdot v = uv$. Für L, L' Wortmengen (formale Sprachen) in $\mathcal{P}(\Sigma^*)$ ist $L \cdot L' = \{\, uv \mid u \in L, v \in L' \}$, $L^0 = \{\varepsilon\}$, $L^{n+1} = L^n \cdot L$, $L^* = \bigcup_{n \in \mathbb{N}} L^n$.

  Beispiel: $L = \{ab\}$, $L^0 = \{\varepsilon\}$, $L^1 = \{ab\}$, $L^2 = \{abab\}$, $L^n = \{\underbrace{abab \ldots ab}_{n-mal}\} = \{(ab)^n\}$

1.2 Transitionssysteme, NEAs

Definitionen:

- Ein *Transitionssystem* hat die Form $\mathfrak{A} = (Q, \Sigma, I, \Delta, F)$, wobei
  - Q eine *Zustandsmenge*,
  - Σ ein endliches *Alphabet* (Q und Σ disjunkt),
  - $I, F \subseteq Q$ sind die Mengen der *Anfangs- bzw. Endzustände*,
  - $\Delta \subset Q \times \Sigma \times Q$ ist *Transitionsrelation*.

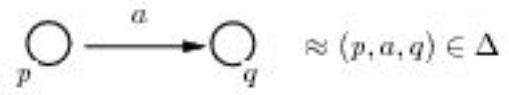
  $\mathfrak{A}$ heißt *endlich* genau dann, wenn Q endlich ist.
- Ein endliches Transitionssystem heißt *nichtdeterministischer endlicher Automat (NEA)* genau dann, wenn $I = \{q_0\}$ für $q_0 \in Q$.

  Notation: $\mathfrak{A} = (Q, \Sigma, q_0, \Delta, F)$
- Ein *Pfad durch* $\mathfrak{A}$ ist eine Folge $\pi = p_0 a_1 p_1 a_2 \ldots a_n p_n$ mit $(p_i, a_{i+1}, p_{i+1}) \in \Delta$ für $0 \leq i \leq n-1$. Die *Länge* von π sei n und die *Beschriftung* $\beta(\pi)$ sei $a_1 a_2 \ldots a_n$.

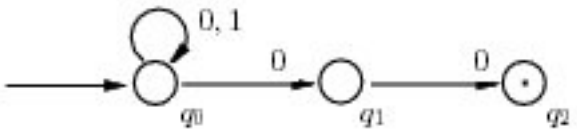
- $\mathfrak{A} : p \xrightarrow{w} q$ für $w \in \Sigma^*$ besagt, dass es Pfad π durch $\mathfrak{A}$ von p nach q mit Beschriftung $\beta(\pi) = w$ gibt.
- $\mathfrak{A}$ *akzeptiert* $w \in \Sigma^*$ genau dann, wenn $p \in I, q \in F$ existieren mit $\mathfrak{A} : p \xrightarrow{w} q$. Für NEA $\mathfrak{A}$ sei $L(\mathfrak{A}) = \{w \in \Sigma^* \mid \mathfrak{A} \text{ akzeptiert } w\}$ die *durch* $\mathfrak{A}$ *akzeptierte Sprache*.

Beispiele:

1. Darstellung des NEA $\mathfrak{A}$ durch Graphen: Zustände sind Knoten, Transitionsrelation ergeben Kanten, Kantenbeschriftung durch Buchstaben aus Σ.

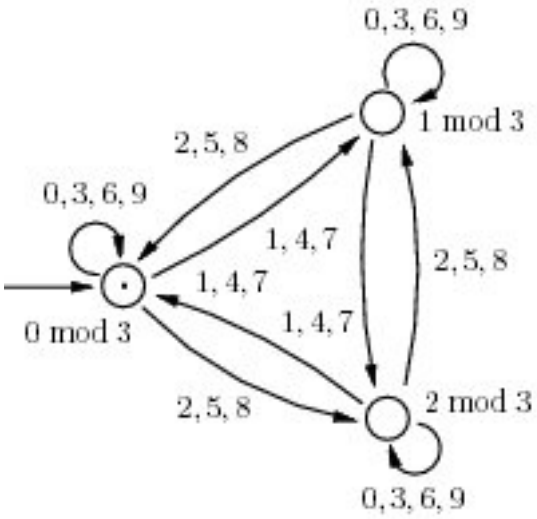


2. $\Sigma = \{0, 1\}, L = \{w \in \Sigma^* \mid w = w'00 \text{ für } w' \in \Sigma^*\}$



$$\begin{aligned} \mathfrak{A} &= (\{q_0, q_1, q_2\}, \{0, 1\}, q_0, \Delta, \{q_2\}) \\ \Delta &= \{(q_0, 0, q_0), (q_0, 1, q_0), (q_0, 0, q_1), (q_1, 0, q_2)\} \end{aligned}$$

3. $\Sigma = \{0, \ldots, 9\}$, $L = \{w \in \Sigma^* \mid w$ stellt eine durch 3 teilbare Dezimalzahl dar (Quersumme durch 3 teilbar) $\}$



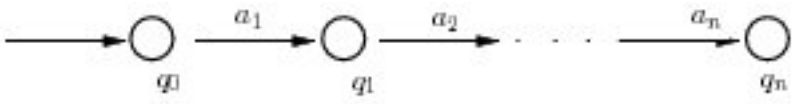
DEFINITION: $\mathfrak{A}$ ist *deterministisch*, wenn für alle $p \in Q$ und alle $a \in \Sigma$ *genau* ein Zustand $q \in Q$ existiert mit $(p, a, q) \in \Delta$. In diesem Fall verwendet man anstelle von Δ eine Funktion $\delta : Q \times \Sigma \to Q$.

Beispiele:

3. $L = \emptyset$ akzeptiert durch

4. $L = \{\varepsilon\}$ akzeptiert durch

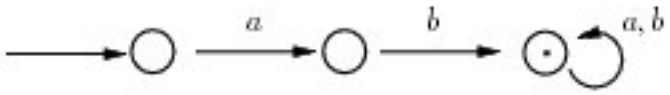
5. $L = \{w\}, w = a_1 \dots a_n \in \Sigma^*$



DEFINITIONEN: Seien $u, w \in \Sigma^*$. u heißt *Präfix* von w, wenn es $v \in \Sigma^*$ gibt mit $w = uv$. u heißt *Suffix* von w, wenn es $v \in \Sigma^*$ gibt mit $w = vu$. u heißt *Infix* von w, wenn es $v_1, v_2 \in \Sigma^*$ gibt mit $w = v_1 u v_2$.

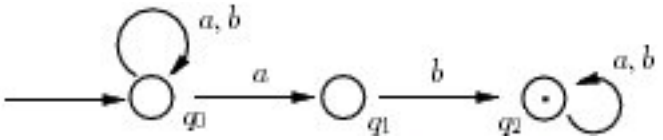
Beispiele:

6. $L_1 = \{ w \in \Sigma^* \mid ab \text{ ist Präfix von } w\}$

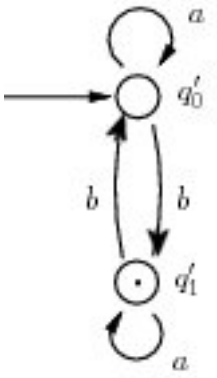


7. $L_2 = \{ w \in \Sigma^* \mid w \text{ hat Infix } ab \text{ und } |w|_b \text{ ungerade } \}$

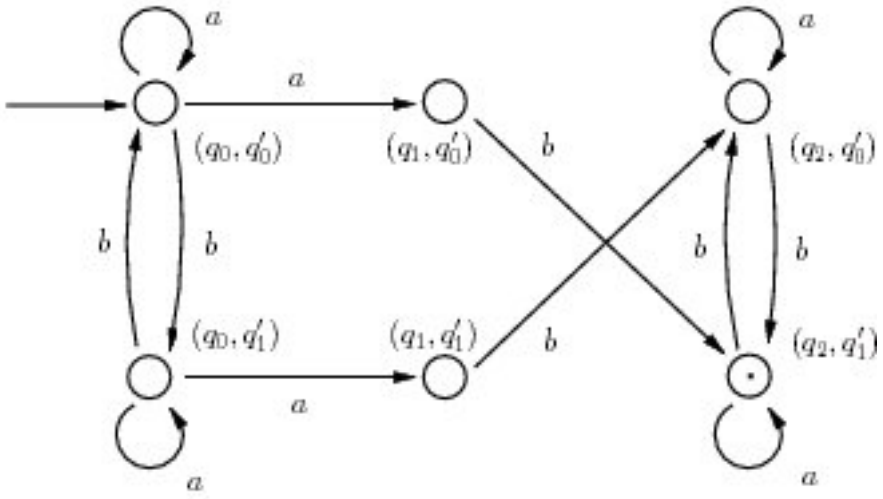
   Eigenschaft „w hat Infix ab“



   Eigenschaft „$|w|_b$ ungerade “



   beide Eigenschaften zusammen:



DEFINITION: Gegeben seien zwei NEAs $\mathfrak{A}_1 = (Q_1, \Sigma, q_1, \Delta_1, F_1)$ und $\mathfrak{A}_2 = (Q_2, \Sigma, q_2, \Delta_2, F_2)$. Der *Produktautomat* $\mathfrak{A}_1 \times \mathfrak{A}_2$ ist gegeben durch $\mathfrak{A}_1 \times \mathfrak{A}_2 = (Q_1 \times Q_2, \Sigma, (q_1, q_2), \Delta, F_1 \times F_2)$ mit $((p, r), a, (p', r')) \in \Delta$ genau dann, wenn $(p, a, p') \in \Delta_1$ und $(r, a, r') \in \Delta_2$.

Bemerkung: $L(\mathfrak{A}_1 \times \mathfrak{A}_2) = L(\mathfrak{A}_1) \cap L(\mathfrak{A}_2)$.

Beweis:

$w = a_1 \dots a_n \in L(\mathfrak{A}_1 \times \mathfrak{A}_2)$

$\iff$ es ex. Pfad $(p_0, r_0)a_1(p_1, r_1)a_2 \dots a_n(p_n, r_n)$ mit $(p_0, r_0) = (q_1, q_2), (p_n, r_n) \in F$ und $((p_i, r_i), a_{i+1}, (p_{i+1}, r_{i+1})) \in \Delta$

$\iff$ es ex. Pfade $p_0 a_1 p_1 \dots a_n p_n$ in $\mathfrak{A}_1$ und $r_0 a_1 r_1 \dots a_n r_n$ in $\mathfrak{A}_2$ mit $p_0 = q_1, r_0 = q_2, p_n \in F_1, r_n \in F_2$, $(p_i, a_{i+1}, p_{i+1}) \in \Delta_1, (r_i, a_{i+1}, r_{i+1}) \in \Delta_2$

$\iff$ $w \in L(\mathfrak{A}_1)$ und $w \in L(\mathfrak{A}_2)$

DEFINITIONEN:

1. Zwei Automaten $\mathfrak{A}_1$ und $\mathfrak{A}_2$ heißen *äquivalent*, wenn $L(\mathfrak{A}_1) = L(\mathfrak{A}_2)$.
2. Ein *NEA mit Worttransitionen* hat die Form $\mathfrak{A} = (Q, \Sigma, q_0, \Delta, F)$, wobei Q, Σ und Δ endlich sind und $\Delta \subseteq Q \times \Sigma^* \times Q$ und $q_0 \in Q, F \subset Q$.

   Sonderfall: ε-NRA mit $\Delta \subset Q \times (\Sigma \cup \{\varepsilon\}) \times Q$.

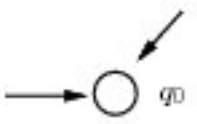
SATZ: Zu jedem NEA $\mathfrak{A}$ mit Worttransitionen gibt es einen äquivalenten NEA $\mathfrak{A}'$.

LEMMA: Zu jedem NEA $\mathfrak{A}$ mit Worttransitionen gibt es einen ε-NEA $\mathfrak{A}'$ mit $L(\mathfrak{A}) = L(\mathfrak{A}')$.

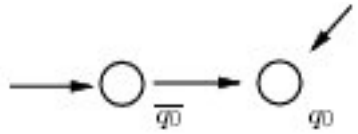
Beweis: Ersetze jede Transition $(p, a_1 \dots, a_n), q$ in $\mathfrak{A}$ mit $n \geq 2$ durch Transitionen $(p, a_1, q_1), (q_1, a_2, q_2), \dots, (q_{n-1}, a_n, q_n)$ mit jeweils neuen Zuständen $q_1, \dots, q_{n-1}$.

LEMMA: Zu jedem ε-NEA $\mathfrak{A}$ gibt es einen äquivalenten NEA $\mathfrak{A}'$.

Beweis: Sei $\mathfrak{A} = (Q, \Sigma, q_0, \Delta, F)$ ein ε-NEA. O.B.d.A. gebe es keine Transition nach q_0, anderenfalls ersetze



durch



Wir definieren $\mathfrak{A}' = (Q, \Sigma, q_0, \Delta', F')$ mit

$$(\star) \quad (p,a,q) \in \Delta' \iff \exists \text{ Pfad } \pi \text{ durch } \mathfrak{A} \text{ von } p \text{ nach } q \text{ mit Beschriftung } \beta(\pi) = a \quad (\mathfrak{A} : p \xrightarrow{a} q)$$

und

$$F' = \begin{cases} F \cup \{q_0\} & \text{falls } \mathfrak{A} : q_0 \xrightarrow{\varepsilon} F \\ F & \text{sonst} \end{cases}$$

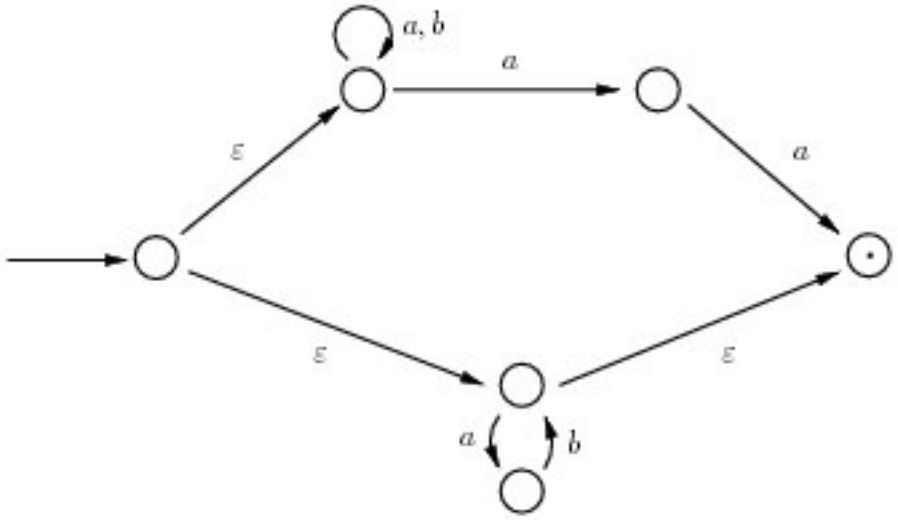
Ein **Entscheidungsverfahren** zu $(\star)$ fehlt noch und folgt später. Zeige nun: $L(\mathfrak{A}) = L(\mathfrak{A}')$.

„$\subseteq$“ $\mathfrak{A}$ akzeptiert w. Sei $p_0 a_1 p_1 \dots a_n p_n$ Pfad durch $\mathfrak{A}$ nach $F (p_n \in F)$ mit Beschriftung $w = a_1 \dots a_n$. Seien $a_{i_1} \dots a_{i_m}$ die $a_i \neq \varepsilon$, also $w = a_{i_1} \dots a_{i_m}$.

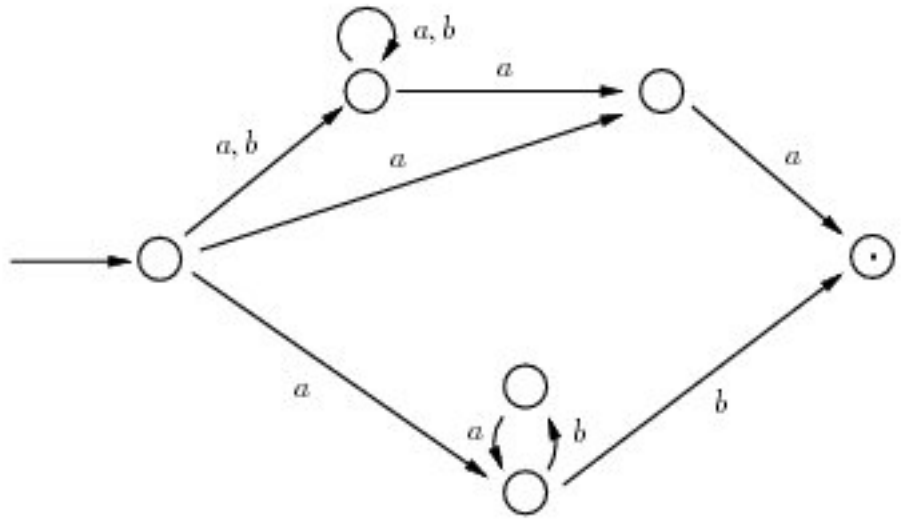
- Fall 1: $m = 0$, dann $w = \varepsilon$ und $\mathfrak{A} : q_0 \xrightarrow{\varepsilon} F$, also $q_0 \in F'$ und $\mathfrak{A}'$ akzeptiert ε.
- Fall 2: $m > 0$. Dann ist $p_0 a_{i_1} p_{i_1} a_{i_2} p_{i_2} \dots a_{i_m} p_n$ ein Pfad in $\mathfrak{A}'$, denn nach Definition von Δ' gilt $(p_0, a_{i_1}, p_{i_1}), (p_{i_1}, a_{i_2}, p_{i_2}), \dots, (p_{i_{m-1}}, a_{i_m}, p_n) \in \Delta'$. Daraus folgt: $\mathfrak{A}'$ akzeptiert w.

„$\supseteq$“ analog zu zeigen.

Beispiel:



wird zu



Verfahren zur Entscheidung, ob $\mathfrak{A} : p \xrightarrow{a} q$ $(a \in \Sigma)$. Hierzu reicht es ein Verfahren zur Entscheidung, ob $\mathfrak{A} : p' \xrightarrow{\varepsilon} q'$ für $p', q' \in Q$ durch einen ε-Pfad. Gegeben: ein ε-NEA $\mathfrak{A} = (Q, \Sigma, q_0, \Delta, F)$ mit $Q = \{q_1, \ldots, q_n\}, q_0 = q_1$. Definiere für $1 \leq i, j \leq n$

$$\varepsilon_{ij} = \begin{cases} 1, & \text{falls } (q_i, \varepsilon, q_j) \in \Delta \text{ oder } i = j \\ 0, & \text{sonst} \end{cases}$$

Gesucht: Werte

$$c_{ij} = \begin{cases} 1, & \text{falls } \mathfrak{A} : q_i \xrightarrow{\varepsilon} q_j \\ 0, & \text{sonst} \end{cases}$$

Dazu berechne rekursiv:

$$c_{ij}^{(r)} = \begin{cases} 1, & \text{falls ein } \varepsilon\text{-Pfad von } q_i \text{ nach } q_j \text{ der Länge} \leq r \text{ existiert} \\ 0, & \text{sonst} \end{cases}$$

Behauptung: $c_{ij} = c_{ij}^{(n-1)}$ für $n = |Q|$, d.h. es existiert ein ε-Pfad von q_i nach q_j genau dann, wenn es existiert ein ε-Pfad von q_j nach q_j der Länge höchstens $n-1$.

Beweis:

„$\Leftarrow$“ trivial.

„$\Rightarrow$“ Es sei π ein ε-Pfad von q_i nach q_j minimaler Länge. Zeige: Die Länge von π ist kleiner gleich $n-1$. Annahme: $\pi = p_0 \varepsilon p_1 \varepsilon \ldots \varepsilon p_m$ mit $p_0 = q_i, p_m = q_j$ und $m \geq n$. Aus der Annahme folgt: es gibt eine Zustandswiederholung mit $p_k = p_{k'}$ und $k < k'$. Also ist $\pi' = p_0 \varepsilon p_1 \ldots \varepsilon p_k \varepsilon p_{k'+1} \varepsilon \ldots \varepsilon p_m$ ein kürzerer ε-Pfad von q_i nach q_j. Dies ist ein Widerspruch zur Minimalität von π.

Wie berechnet man $c_{ij}^{(r)}$?

$$\begin{aligned} c_{ij}^{(0)} &= \begin{cases} 1, & \text{falls } i = j \\ 0, & \text{sonst} \end{cases} \\ c_{ij}^{(r+1)} &= c_{ij}^{(r)} \vee \bigvee_{k=1}^{n} \left(c_{ik}^{(r)} \wedge \varepsilon_{kj} \right) \end{aligned}$$

d.h. es gibt einen Pfad der Länge $\leq r+1$, wenn es einen Pfad der Länge $\leq r$ gibt oder es gibt einen Zwischenknoten q_k und einen Pfad der Länge $\leq r$ von q_i nach q_k und eine ε-Transition (q_k, ε, q_j). Zeitaufwand $\mathcal{O}(n^4)$, verbesserbar durch andere Ideen auf $\mathcal{O}(n^3)$.

1.3 DEAs

DEFINITIONEN:

- Ein NEA $\mathfrak{A} = (Q, \Sigma, q_0, \Delta, F)$ heißt *deterministischer endlicher Automat (DEA)*, falls zu jedem $(q, a) \in Q \times \Sigma$ genau ein $q' \in Q$ mit $(q, a, q') \in \Delta$ existiert. Dann ist Δ beschreibbar durch eine Funktion:

$$\delta : Q \times \Sigma \to Q \text{ mit } \delta(p, a) = q \Leftrightarrow (p, a, q) \in \Delta$$

- Die *Fortsetzung* von $\delta : Q \times \Sigma \to Q$ zu $\delta^* : Q \times \Sigma^* \to Q$ wird definiert durch

$$\begin{aligned} \delta^*(q, \varepsilon) &= q \\ \delta^*(q, wa) &= \delta(\delta^*(q, w), a) \end{aligned}$$

Bemerkungen:

1. $\delta^*(p, w) = q \Leftrightarrow \mathfrak{A} : p \xrightarrow{w} q$
2. $L(\mathfrak{A}) = \{ w \in \Sigma^* \mid \delta^*(q_0, w) \in F \}$

Beweis:

1. Induktion über den Aufbau der Wörter über Σ.
   - Induktionsanfang: $w = \varepsilon$.

$$\delta^*(p, \varepsilon) = q \Leftrightarrow p = q \Leftrightarrow \mathfrak{A} : p \xrightarrow{\varepsilon} q$$

   - Induktionsschritt: $w = va$, dann gilt

$$\begin{aligned} \delta^*(p, va) = q &\overset{Def.\delta^*}{\Leftrightarrow} \delta(\delta^*(p, v), a) = q \\ &\Leftrightarrow \exists p' : \ \delta^*(p, v) = p' \wedge \delta(p', a) = q \\ &\overset{IV}{\Leftrightarrow} \exists p' : \ \mathfrak{A} : p \xrightarrow{v} p' \wedge \mathfrak{A} : p' \xrightarrow{a} q \\ &\Leftrightarrow \mathfrak{A} : p \xrightarrow{va} q \end{aligned}$$

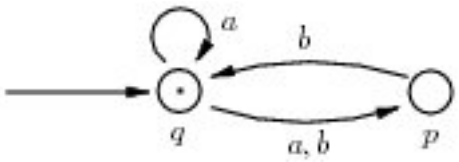
2. trivial.

Schreibweise: $\delta(q, w)$ statt $\delta^*(q, w)$.

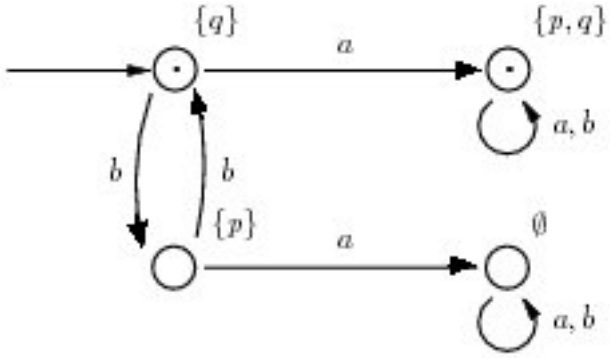
SATZ: (Rabin, Scott) Zu jedem NEA kann man einen äquivalenten DEA konstruieren (Potenzmengen-Konstruktion).

Idee: Verfolge ausgehend von $\{q\}$ längs aller Wörter über Σ die Mengen von Zuständen, die so erreichbar sind.

Beispiel:



wird zu



Formal: Sei ein NEA $\mathfrak{A} = (Q, \Sigma, q_0, \Delta, F)$ gegeben. Definiere $\mathfrak{A}' = (Q', \Sigma, q_0', \delta', F')$ wie folgt:

- $Q' = 2^Q = \{ R \mid R \subseteq Q \}$.
- $q_0' = \{q_0\}$.
- $\delta'(R, a) = \{ q \in Q \mid \exists r \in R \text{ mit } (r, a, q) \in \Delta \}, R \subseteq Q$.
- $F' = \{ R \subseteq Q \mid R \cap F \neq \emptyset \}$

Behauptung: $\mathfrak{A} : p \xrightarrow{w} q \Leftrightarrow q \in \delta'(\{p\}, w)$.

Beweis: Per Induktion über $|w|$:

- Induktionsanfang: $|w| = 0$.

$$\mathfrak{A} : p \xrightarrow{\varepsilon} q \Leftrightarrow p = q \Leftrightarrow q \in \delta'(\{p\}, \varepsilon)$$

- Induktionsschritt: Sei $w = va$ und es gelte die IV: $\mathfrak{A} : p \xrightarrow{v} r \Leftrightarrow r \in \delta'(\{p\}, v)$. Wir erhalten

$$\begin{array}{rcl}
\mathfrak{A} : p \xrightarrow{va} q & \Rightarrow & \exists r \in Q \text{ mit } \mathfrak{A} : p \xrightarrow{v} r \\
& \Leftrightarrow & \exists r \in Q \text{ mit } r \in \delta'(\{p\}, v) \\
\mathfrak{A} : p \xrightarrow{va} q & \Leftrightarrow & \exists r \in Q \text{ mit } \mathfrak{A} : p \xrightarrow{v} r,\ \mathfrak{A} : r \xrightarrow{a} q \\
& \overset{IV}{\Leftrightarrow} & \exists r \in Q \text{ mit } r \in \underbrace{\delta'(\{p\}, v)}_{R},\ \underbrace{(r, a, q) \in \Delta}_{q \in \delta'(\{r\}, a) \subseteq \delta'(R, a)} \\
& \overset{Def.\delta'}{\Leftrightarrow} & q \in \delta'(\delta'(\{p\}, v), a) \\
& \overset{Erw.}{\Leftrightarrow} & q \in \delta'(\{p\}, va)
\end{array}$$

Damit erhalten wir die Äquivalenz von $\mathfrak{A}$ und $\mathfrak{A}'$:

$$\begin{array}{rcl}
\mathfrak{A} \text{ akzeptiert } w & \Leftrightarrow & \exists q \in F\ \mathfrak{A} : q_0 \xrightarrow{w} q \\
& \overset{Beh.}{\Leftrightarrow} & \exists q \in F\ q \in \delta'(\{q_0\}, w) \\
& \Leftrightarrow & F \cap \delta'(\{q_0\}, w) \neq \emptyset \\
& \overset{DefF'}{\Leftrightarrow} & \delta'(\{q_0\}, w) \in F' \\
& \Leftrightarrow & \mathfrak{A}' \text{ akzeptiert } w
\end{array}$$

Sprechweise: „L ist von NEA (DEA) erkennbar“ bzw. „L ist regulär“.

LEMMA: Ist $L \in \Sigma^*$ von DEA erkennbar, so auch $\Sigma^* \setminus L$.

Beweis: Sei $\mathfrak{A} = (Q, \Sigma, q_0, \delta, F)$ DEA, der L erkennt.

$$\begin{array}{rcl}
w \in \Sigma^* \setminus L & \Leftrightarrow & \delta(q_0, w) \in Q \setminus F \\
& \Leftrightarrow & \mathfrak{A}' = (Q, \Sigma, q_0, \delta, Q \setminus F) \text{ akzeptiert } w
\end{array}$$

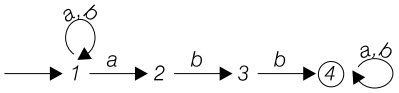
Also: $\mathfrak{A}'$ erkennt $\Sigma^* \setminus L$.

Folgerung: Die durch DEAs erkennbaren Sprachen über Σ bilden eine *Boolsche Algebra*.
Die Potenzmengenkonstruktion enthält einen Größensprung bei den Automaten (von n auf 2^n), daher stellen wir eine Methode zur *Minimierung von DEAs* vor. Wir führen zwei Schritte durch:

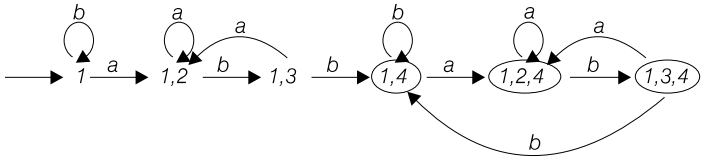
1. Bei der Potenzmengenkonstruktion Einschränkung auf erreichbare Zustände.

2. Zusammenfassen „äquivalenter Zustände“

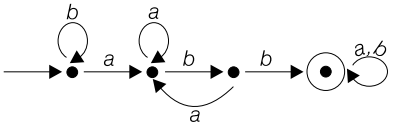
Beispiel: gegeben sei $\mathfrak{A}$:



P-Automat:



Wegen Beschränkung auf erreichbare Zustände ergeben sich 6 statt 16 Zustände. Da man von einem der drei Endzustände nur zu einem weiteren Endzustand kommt, lassen sich diese zusammenfassen. Als Ergebnis gemäß dem zweiten Schritt ergeben sich nur 4 Zustände.:



DEFINITION: Für einen DEA $\mathfrak{A} = (Q, \Sigma, q_0, \delta, F)$ gelte

$$p \sim_{\mathfrak{A}} q :\Longleftrightarrow \forall w \in \Sigma^* : (\delta(p, w) \in F \Leftrightarrow \delta(q, w) \in F).$$

Bemerkungen:

- $\sim_{\mathfrak{A}}$ ist Äquivalenzrelation auf Q, $\tilde{q}$ sei die Äquivalenzklasse von q.
- $a \in \Sigma, p \sim_{\mathfrak{A}} q \Rightarrow \delta(p, a) \sim_{\mathfrak{A}} \delta(q, a)$.

  Beweis: Kontraposition ist: $\neg\delta(p, a) \sim_{\mathfrak{A}} \delta(q, a) \Rightarrow \neg p \sim_{\mathfrak{A}} q$. Wähle also w mit $\delta(\delta(p, a), w) \in F$, $\delta(\delta(q, a), w) \notin F$. Dann ist $\delta(p, aw) \in F$, $\delta(q, aw) \notin F$, also $\neg p \sim_{\mathfrak{A}} q$.

DEFINITION: Nenne p, q *trennbar* über Länge l, falls für ein w mit $|w| = l$ die Äquivalenz $\delta(p, w) \in F \Leftrightarrow \delta(q, w) \in F$ nicht zutrifft.

Nach obiger Definition gilt

($\star$) Sind $\delta(p, a), \delta(q, a)$ trennbar über Länge l, so p, q über Länge $l + 1$.

DEFINITION: Der *reduzierte* DEA $\tilde{\mathfrak{A}}$ zu $\mathfrak{A}$ sei $\tilde{\mathfrak{A}} := (\tilde{Q}, \Sigma, \tilde{q_0}, \tilde{\delta}, \tilde{F})$ mit $\tilde{Q} = \{\tilde{q} \mid q \in Q\}$, $\tilde{\delta}(\tilde{q}, a) = \widetilde{\delta(q, a)}$ (wohldefiniert nach Bemerkung oben), $\tilde{F} = \{\tilde{q} \mid q \in F\}$.

Es gilt:

1. $\tilde{\delta}(\tilde{q}, w) = \widetilde{\delta(q, w)}$ (mit Induktion über $|w|$).
2. $\tilde{\mathfrak{A}}$ äquivalent zu $\mathfrak{A}$, denn:

$$\begin{aligned} \mathfrak{A} \text{ akzeptiert } w &\Leftrightarrow \delta(q_0, w) \in F \\ &\Leftrightarrow \widetilde{\delta(q_0, w)} \in \tilde{F} \\ &\Leftrightarrow \tilde{\delta}(\tilde{q_0}, w) \in \tilde{F} \\ &\Leftrightarrow \tilde{\mathfrak{A}} \text{ akzeptiert } w. \end{aligned}$$

Um den Übergang $\mathfrak{A} \to \tilde{\mathfrak{A}}$ algorithmisch durchführen zu können, brauchen wir ein Verfahren zum Test, ob $p \sim_{\mathfrak{A}} q$. **Idee:** Bestimme für $l = 0, 1, 2, \ldots$ die (p, q), die über Länge l trennbar sind.

Markierungsalgorithmus

(0) Markiere alle Paare (p, q), die über Länge 0 trennbar sind (d.h. $\neg(p \in F \Leftrightarrow q \in F)$).

(1) Solange im letzten Schritt ein zusätzliches Paar markiert wurde: Markiere alle noch nicht markierten Paare (p, q), für die mit geeignetem $a \in \Sigma$ das Paar $(\delta(p, a), \delta(q, a))$ schon markiert ist.

Beispiel:

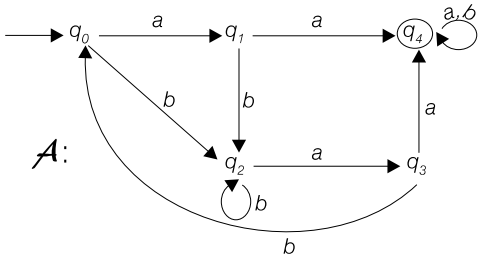
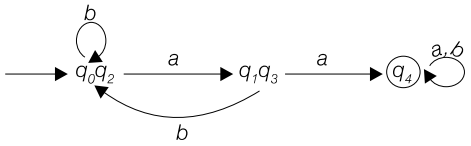


Tabelle der Paare:

	q_0	q_1	q_2	q_3
q_1	*			
q_2		*		
q_3	*		*	
q_4	X	X	X	X

X in (0) markiert
* in (1) markiert

Ergebnis: $q_0 \sim_{\mathfrak{A}} q_2, q_1 \sim_{\mathfrak{A}} q_3$. Es ergibt sich der Automat $\tilde{\mathfrak{A}}$:



Korrektheitsbehauptung: Der Markierungsalgorithmus terminiert, und genau dann sind die trennbaren Paare markiert.

Beweis:

1. Es werden nur trennbare Paare markiert,
2. der Algorithmus terminiert,
3. bei Termination sind alle trennbaren Paare markiert.

zu 1: klar nach Bemerkung, siehe oben ($\star$)

zu 2: klar, da nur endlich viele Paare markierbar, und jede Durchführung von (1) markiert ein neues Paar

zu 3: mit folgenden Punkten (a) und (b) sind wir offensichtlich fertig:

(a) nach l Schritten (1) sind alle Paare markiert, die über Länge l trennbar sind (der Beweis ist einfach, mit Induktion über l).

(b) Die Termination erfolge nach m Markierungsschritten (1). Dann sind alle überhaupt trennbaren Paare schon über Länge $\leq m$ trennbar (d.h. nach $\leq m$ Schritten (1)).

Beweis:

- Gemäß Terminationsbedingung liegt vor dem $(m+1)$-ten Schritt (1) kein Paar vor, das über Länge $m+1$ trennbar ist und noch nicht über Länge m. (Sonst gäbe es nämlich noch ein zu markierendes Paar.)
- Wir schließen noch die Trennbarkeit von Paaren über Längen $l > m+1$ aus. Annahme: Es gibt (p, q), über Länge $l > m+1$ trennbar, aber nicht über kleinere Längen. In einem trennbaren Wort sei v das Suffix der letzten $m+1$ Buchstaben:

$$\mathfrak{A} : \begin{cases} p \xrightarrow{u} p' \xrightarrow{v} F \\ q \xrightarrow{u} q' \xrightarrow{v} \not\in F \end{cases} \qquad |uv| = l, |v| = m+1.$$

Dann ist (p', q') über Länge $m+1$ trennbar, aber nicht über kleinere Länge (sonst wäre (p, q) über Länge $< l$ trennbar). Widerspruch zu oben.

1.3.1 Charakterisierung der regulären Sprachen

Abschließend geben wir eine *Charakterisierung der regulären Sprachen* an, mit der man auch nachweisen kann, daß gewisse Sprachen nicht DEA-erkennbar sind. Schlüssel ist folgende Definition:

DEFINITION: Sei $L \subseteq \Sigma^*$ (beliebig!) und $u, v \in \Sigma^*$. Wir definieren

$$u \widetilde{=}_L v :\Longleftrightarrow \forall\, w \in \Sigma^* (uw \in L \Leftrightarrow vw \in L)$$

Dann ist $\widetilde{=}_L$ eine Äquivalenzrelation, $[u]_L$ sei die $\widetilde{=}_L$-Äquivalenzklasse von u.

Beispiel: Sei $L = \{ w \in \{a, b\}^* \mid w \text{ hat } abb \text{ als Infix}\}$

- $\varepsilon \widetilde{\neq}_L a$, denn $\varepsilon bb \notin L$, aber $abb \in L$.
- $\varepsilon \widetilde{=}_L b$, denn $\varepsilon w \in L \Leftrightarrow w$ hat Infix $abb \Leftrightarrow bw \in L$.
- $a \widetilde{=}_L aa$, denn $aw \in L \Leftrightarrow w$ hat Infix abb oder beginnt mit $bb \Leftrightarrow aaw \in L$.
- $a \widetilde{\neq}_L ab$, denn $ab \notin L$, aber $abb \in L$.
- $ab \widetilde{\neq}_L abb$, denn $ab\varepsilon \notin L$, aber $abb\varepsilon \in L$.

Nach genauerer Analyse ergeben sich insgesamt vier $\widetilde{=}_L$-Klassen: $[\varepsilon]_L, [a]_L, [ab]_L, [abb]_L$.

DEFINITION: Es sei Index($\widetilde{=}_L$) definiert als die Anzahl der Äquivalenzklassen.

SATZ: (Nerode) $L \subseteq \Sigma^*$ ist regulär genau dann, wenn Index($\widetilde{=}_L$) endlich ist.

Beweis:

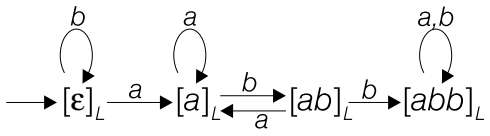
„⇒“ Sei $\mathfrak{A} = (Q, \Sigma, q_0, \delta, F)$ ein DEA, der L erkennt. *Zeige:* Verschiedene $\widetilde{=}_L$-Klassen der Σ^* bestimmen verschiedene $\widetilde{=}_{\mathfrak{A}}$-Klassen in Q. Dann ist $|Q| \geq \text{Index}(\widetilde{=}_{\mathfrak{A}}) \geq \text{Index}(\widetilde{=}_L)$, und die Behauptung ist gezeigt, da Q endlich. Gilt $\neg u \widetilde{=}_L v$, d.h. $[u]_L \neq [v]_L$, so bilde $\delta(q_0, u)$ und $\delta(q_0, v)$. Zeige dann $\delta(q_0, u) \widetilde{\neq}_{\mathfrak{A}} \delta(q_0, v)$. Wäre $\delta(q_0, u) \widetilde{=}_{\mathfrak{A}} \delta(q_0, v)$, so wäre

$$\forall\, w \in \Sigma^* : \underbrace{\delta(\delta(q_0, u), q)}_{uw \in L} \in F \Leftrightarrow \underbrace{\delta(\delta(q_0, v), q)}_{vw \in L} \in F$$

also ist $uw \in L \Leftrightarrow vw \in L$, d.h. $u \widetilde{=}_L v$. Widerspruch!

„⇐“ (kommt eventuell noch)

Im oben genannten Beispiel der Sprache $L = \{ w \in \{a,b\}^* \mid w \text{ hat } abb \text{ als Infix}\}$ erhalten wir den folgenden Automaten $\mathfrak{A}_L$:



1.3.2 Angabe nichtregulärer Sprachen

Aus dem Satz von Nerode (Richtung „⇒“) folgt: L ist nicht regulär, falls Index($\widetilde{=}_L$) unendlich ist, d.h. falls es Wörter $u_0, u_1, \ldots$ gibt, so daß für $i \neq j$ jeweils ein w existiert mit $u_i w \in L, u_j w \notin L$.

Beispiele:

- $L_1 := \{ a^i b^i \mid i \geq 0\}$. Wähle $u_i := a^i$. Für $i \neq j$ gilt mit $w := b^i$, daß $u_i w = a^i b^i \in L_1$ während $u_j w = a^j b^i \notin L_1$
- $L_2 := \{ a^i b^j \mid \text{ggT}(i,j) = 1\}$. Wähle $u_p := 0^p$ für p Primzahl. Dann ist mit $w = 1^q$ (wobei q Primzahl) $u_p w \in L$, aber $u_q w \notin L$.

1.3.3 Pumping-Lemma

LEMMA: Sei L eine reguläre Sprache. Dann gibt es ein $n \in \mathbb{N}$, so daß sich alle Wörter $x \in L$ mit $|x| \geq n$ zerlegen lassen in $x = uvw$ mit

1. $|v| \geq 1$
2. $|uv| \leq n$
3. für alle $i \in \mathbb{N}_0$ gilt: $uv^i w \in L$ (also auch $uw \in L$)

Beweis: Sei $\mathfrak{A}$ ein DEA, der L akzeptiert. Wir wählen $n = |Q|$. Sei $x \in L$ mit $|x| = m \geq n$ und $x = a_1 \ldots a_m$. Damit existiert ein Pfad $\pi : p_0 a_1 p_1 a_2 \ldots a_m p_m$ und $p_m \in F$. Die Anzahl der Zustände im Pfad ist $m + 1 \geq |Q| + 1$, d.h. ein Zustand kommt mindestens zwei mal vor. Unter den ersten $n + 1$ Zuständen $p_0, \ldots, p_n$ existiert ein Paar k, k' mit $0 \leq k < k' \leq n$ und $p_k = p_{k'}$. Wähle nun $u = a_1 \ldots a_k$ und $v = a_{k+1} \ldots a_{k'}$ sowie $w = a_{k'+1} \ldots a_n$. Damit sind die Bedingungen aus dem Lemma erfüllt und es gilt $uv^i \in L$.

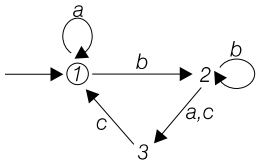
Beispiele:

1. Sei $L = \{a^i b^i \mid i \geq 1\}$. Annahme, L wäre regulär. Dann existiert ein n wie im Pumping-Lemma. Wähle nun $x = a^n b^n$. Wegen der ersten beiden Eigenschaften im Lemma ist $v \neq \varepsilon$ und $uv = a^m$ mit $m \leq n$, also $v = a^{m'}$ mit $1 \leq m' \leq n$. Nun müßte aber auch $uv^0 w = uw = a^{n-m'} b^n \in L$ sein, Widerspruch!

2. Sei $L = \{0^m \mid m \text{ ist Quadratzahl}\}$. Annahme, L wäre regulär. Dann existiert ein n wie im Pumping-Lemma. Wähle $x = 0^{n^2} \in L$ und betrachte Zerlegung $x = uvw$ mit $1 \leq |v| \leq |uv| \leq n$. Es folgt $uv^2 w \in L$, aber
$$n^2 = |x| = |uvw| < \left|uv^2 w\right| \leq n^2 + n < (n+1)^2$$
Damit liegt das Wort $uv^2 w$ zwischen zwei aufeinanerfolgenden Quadratzahlen, kann also selbst keine solche sein.

1.4 Reguläre Ausdrücke

Idee: Sei ein Automat $\mathfrak{A}$ gegeben:



Die durch den Automaten $\mathfrak{A}$ erkannte Sprache wird kompakter durch den folgenden Ausdruck beschrieben:

$$(a^* b b^* (a + c) c)^* a^*$$

DEFINITION: Die Menge der *regulären Ausdrücke* über einem Alphabet Σ (mit $\varepsilon, \emptyset \notin \Sigma$) ist induktiv definiert durch:

- ε, $\emptyset$, alle $a \in \Sigma$ sind reguläre Ausdrücke
- sind r und s reguläre Ausdrücke, so auch $(r + s)$, $(r \cdot s)$ und r^*

DEFINITION: Die durch den regulären Ausdruck r definierte Sprache $L(r)$ wird induktiv festgelegt durch:

- $L(\emptyset) = \emptyset$, $L(\varepsilon) = \{\varepsilon\}$ $L(a) = \{a\}$
- $L(r + s) = L(r) \cup L(s)$, $L(r \cdot s) = L(r) \cdot L(s)$ $L(r^*) = L(r)^*$

Konventionen: Außenklammern fallen weg, $\cdot$ bindet stärker als $+$, * stärker als $\cdot$, und $\cdot$ darf wegfallen. Statt $((a \circ b^*) + b)$ schreiben wir also auch $ab^* + b$. Häufig schreiben wir auch r statt $L(r)$.

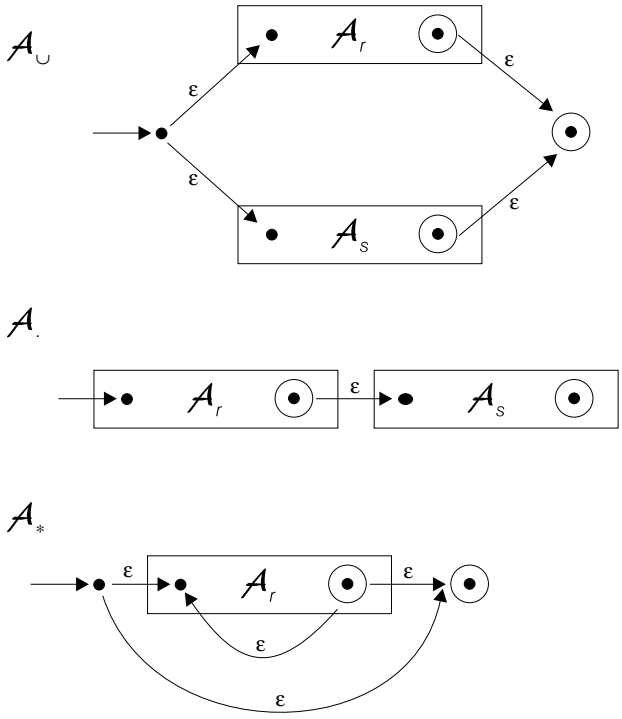
SATZ: (Kleene) Eine Sprache $L \subseteq \Sigma^*$ ist durch einen NEA erkennbar genau dann, wenn sie durch einen regulären Ausdruck definierbar ist.

Beweis:

„$\Leftarrow$“ Durch Induktion über den Aufbau der regulären Ausdrücke: Finde für jeden regulären Ausdruck r einen ε-NEA $\mathfrak{A}_r$ mit nur einem Endzustand, so daß $L(r) = L(\mathfrak{A}_r)$.

- Induktionsanfang: Die Fälle $r = 0, 1, a \in \Sigma$ sind trivial: Entsprechende ε-NEAs:

- Induktionsvoraussetzung: Zu r, s mögen ε-NEAs mit $\mathfrak{A}_r, \mathfrak{A}_s$ mit nur einem Endzustand existieren, so daß $L(r) = L(\mathfrak{A}_r), L(s) = L(\mathfrak{A}_s)$.
- Induktionsbehauptung: Zu $(r + s)$, $(r \cdot s)$, r^* existieren ε-NEAs $\mathfrak{A}_\cup$, $\mathfrak{A}_\cdot$, $\mathfrak{A}_*$ mit nur einem Endzustand, so daß $L(r + s) = L(\mathfrak{A}_\cup), L(r \cdot s) = L(\mathfrak{A}_\cdot), L(r^*) = L(\mathfrak{A}_*)$. Wir wählen



„⇒“ Sei ein DEA $\mathfrak{A} = (Q = \{q_0, \ldots, q_n\}, \Sigma, q_0, \Delta, F)$ gegeben. Wir definieren für $0 \leq i, j \leq n$:

$$L_{ij} = \left\{ w \in \Sigma^* \;\middle|\; \mathfrak{A} : q_i \xrightarrow{w} q_j \right\}$$

Dann ist $L(\mathfrak{A}) = \bigcup_{q_j \in F} L_{0j}$. Es genügt, r_{ij} für L_{ij} zu finden. Definiere Index(q) für $q \in Q$ als denjenigen Index i mit $q_i = q$. Betrachte folgende Einschänkungen von L_{ij}:[1]

$$\begin{aligned} L_{ij}^k = \{ w \in \Sigma^* \quad | \quad & \exists\, \pi = p_0 a_1 p_1 \ldots p_{m-1} a_m p_m, \\ & p_0 = q_i, p_m = q_j, w = a_1 \cdots a_m, \\ & \text{Index}(p_l) < k \;\forall\; 1 \leq l < m \} \end{aligned}$$

Zeige: Für alle $k = 0, \ldots, n+1$ und für alle i, j ist L_{ij}^k durch einen regulären Ausdruck definierbar. Vollständige Induktion:

- Induktionsanfang: Für $k = 0$ ist

$$L_{ij}^0 = \begin{cases} \{a \mid (q_i, a, q_j) \in \Delta\} \cup \{\varepsilon\} & \text{falls} \quad i = j \\ \{a \mid (q_i, a, q_j) \in \Delta\} & \text{falls} \quad i \neq j \end{cases}$$

  Da L_{ij}^0 endlich ist, existiert ein regulärer Ausdruck, der L_{ij}^0 definiert.

[1] alle Wege von i nach j nur mit Zwischenknoten vom Index höchstens k

– Induktionsschritt $k \Rightarrow k+1$: Es sei L_{ij}^k durch den regulären Ausdruck r_{ij}^k definierbar.

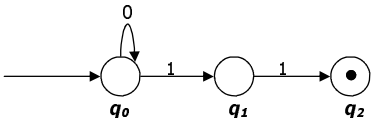
$$w \in L_{ij}^{k+1} \Leftrightarrow (w \in L_{ij}^k) \quad \vee \quad (\exists m \ : \ w = w_0 \cdots w_m \\ \text{mit } w_0 \in L_{ik}^k, w_1, \ldots, w_{m-1} \in L_{kk}^k, w_m \in L_{kj}^k)$$

Somit ist

$$L_{ij}^{k+1} = L_{ij}^k \cup L_{ik}^k (L_{kk}^k)^* L_{kj}^k$$

Der reguläre Ausdruck r_{ij}^{k+1} ergibt sich als $r_{ij}^k + r_{ik}^k (r_{kk}^k)^* r_{kj}^k$

Beispiel:



Es ist $L_{ij} = \emptyset$ für $i > j$. Für $k = 0$ ist

r_{00}^0	r_{01}^0	r_{02}^0	r_{11}^0	r_{12}^0	r_{22}^0
$\varepsilon + 0$	1	$\emptyset$	ε	1	ε

Für $k = 1$ ist

$$\begin{aligned}
r_{00}^1 &= r_{00}^0 + r_{00}^0 (r_{00}^0)^* r_{00}^0 \\
&= (\varepsilon + 0) + (\varepsilon + 0)(\varepsilon + 0)^* (\varepsilon + 0) = 0^* \\
r_{01}^1 &= r_{01}^0 + r_{00}^0 (r_{00}^0)^* r_{01}^0 \\
&= 1 + (\varepsilon + 0)(\varepsilon + 0)^* 1 = 1 + 0^* 1 = 0^* 1 \\
r_{02}^1 &= r_{02}^0 + r_{00}^0 (r_{00}^0)^* r_{02}^0 \\
&= \emptyset + (\varepsilon + 0)(\varepsilon + 0)^* \emptyset = \emptyset + 0^* \emptyset = \emptyset \\
r_{11}^1 &= r_{11}^0 + r_{10}^0 (r_{00}^0)^* r_{01}^0 \\
&= \varepsilon + \emptyset (\varepsilon + 0)^* 1 = \varepsilon \\
r_{02}^2 &= r_{02}^2 + r_{01}^1 (r_{11}^1)^* r_{12}^1 \\
&= \emptyset + 0^* 1 \cdot \varepsilon \cdot 1 = 0^* 11 \\
r_{02}^3 &= r_{02}^2 + r_{02}^2 (r_{22}^2)^* r_{22}^2 \\
&= 0^* 11 + 0^* 11 \cdot \varepsilon = 0^* 11 \hat{=} L(\mathfrak{A})
\end{aligned}$$

1.5 Gleichungssysteme

Sei $\mathfrak{A} = (Q = \{q_0, \ldots, q_n\}, \Sigma = \{a_1, \ldots, a_m\}, q_0, \Delta, F)$ gegeben. Setze $\mathfrak{A}_i = (Q, \Sigma, q_i, \Delta, F)$ mit $L_i = L(\mathfrak{A}_i)$ für $0 \leq i \leq n$. **Nummeriere** die Transitionen aus q_i durch

$$(q_i, a_{(i,1)}, q_{\langle i,1\rangle}), \ldots, (q_i, a_{(i,m_i)}, q_{\langle i,m_i\rangle})$$

Nun bekommt man für jedes i eine Gleichung[2]

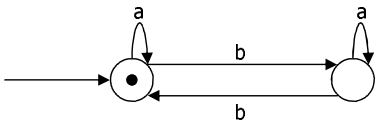
$$L_i = \{a_{(i,1)}\} \cdot L_{\langle i,1\rangle} \cup \ldots \cup \{a_{(i,m_i)}\} \cdot L_{\langle i,m_i\rangle} \cup E_i \qquad (\star)_i$$

Vereinfachung der Schreibweise: $\{a\}L_j \cup \{b\}L_j = \{a, b\}L_j$.

Sei nun $A_{ij} = \{\, a \in \Sigma \mid \{a\}L_j$ kommt in Zeile $(\star)_i$ vor$\}$. Dann lautet die umgeformte Gleichung (mit $A_{ij} \subseteq \Sigma$, $A_{ij} = \emptyset$ möglich; $E_i \subseteq \{\varepsilon\}$):

$$L_i = A_{i0}L_0 \cup \ldots \cup A_{in}L_n \cup E_i$$

Beispiel:



Das Gleichungssystem $GS(\mathfrak{A})$ ist dann

$$\begin{aligned} L_0 &= \{a\}L_0 + \{b\}L_1 + \{\varepsilon\} \\ L_1 &= \{b\}L_0 + \{a\}L_1 \end{aligned}$$

Satz: Das zum NEA $\mathfrak{A}$ gehörige Gleichungssystem $GS(\mathfrak{A})$ $X_i = \bigcup_{j=0}^{n} A_{ij}X_j \cup E_i$ für $i = 0, \ldots, n$ (mit A_{ij} und E_i wie oben beschrieben) ist eindeutig lösbar und $(L_0, \ldots, L_n)$ mit $L_i = L(\mathfrak{A}_i)$ ist Lösung.

Beweis:

1. $(L_0, \ldots, L_n)$ ist Lösung wegen der Definition
2. zur Eindeutigkeit: Seien $(K_0, \ldots, K_n), (K'_0, \ldots, K'_n)$ Lösungen von $GS(\mathfrak{A})$. Wir zeigen nun, daß $\{w\} \cap K_i = \{w\} \cap K'_i \forall w \in \Sigma^*$ (und somit $K_i = K'_i$) ist. Induktion über $|w|$:

[2] mit $E_i = \begin{cases} \{\varepsilon\} & \text{falls} \quad q_i \in F \\ \emptyset & \text{sonst} \end{cases}$

- Induktionsanfang: Für $|w| = 0$ ist

$$\begin{aligned} \{\varepsilon\} \cap K_i &= \bigcup_{j=0}^{n} (\{\varepsilon\} \cap A_{ij}K_j) \cup (\{\varepsilon\} \cap E_i) \\ &= \emptyset \cup (\{\varepsilon\} \cap E_i) = \{\varepsilon\} \cap E_i \end{aligned}$$

  Analog: für K_i' ist $\{\varepsilon\} \cap K_i' = \{\varepsilon\} \cap E_i$

- Induktionsschritt: Sei $|w| = n + 1$ und $w = xy$ mit $|x| = 1$.

$$\begin{aligned} \{w\} \cap K_i &= \bigcup_{j=0}^{n} (\{w\} \cap A_{ij}K_j) \cup (\{w\} \cap E_i) \\ &= \bigcup_{j=0}^{n} ((\{x\} \cap A_{ij})(\{y\} \cap K_j)) \\ &\overset{\text{IV}}{=} \bigcup_{j=0}^{n} ((\{x\} \cap A_{ij})(\{y\} \cap K_j')) \\ &= \{w\} \cap K_i' \end{aligned}$$

Folgerung: Ein regulärer Ausdruck r für einen gegebenen NEA $\mathfrak{A}$ ergibt sich aus einer Lösung $(r_0, \ldots, r_n)$ des Gleichungssystems, indem wir $r = r_0$ setzen[3]:

$$X_i = \sum_{j=0}^{n} a_{ij}X_j + e_i \quad (i = 0, \ldots, n)$$

Frage: Wie lösen wir diese Gleichungssysteme?

Bemerkungen:

(1) $r \cdot \varepsilon = \varepsilon \cdot r = r$ (denn $\{\varepsilon\}L = L$)

(2) $r + \emptyset = \emptyset + r = r$

(3) $\varepsilon^* = \emptyset^* = \varepsilon$ (denn $\emptyset^* = \{\varepsilon\} \cup \emptyset \cup \emptyset \cdot \emptyset \cup \ldots$)

(4) $r(sr)^* = (rs)^*r$

(5) $(r + s)^* = (r^*s^*)^*$

(6) $r(s + s') = rs + rs'$ und $(s + s')r = sr + s'r$

Beweis (exemplarisch) von (d):

[3] wobei a_{ij}, e_i reguläre Ausdrücke für Sprachen A_{ij}, E_i im $GS(\mathfrak{A})$ sind

„$\subseteq$“ Zu zeigen ist $r(sr)^* \subseteq (rs)^*r$, sei also $w \in r(sr)^*$. Zeige, daß $w \in (rs)^*r$ ist. Es folgt: $w = uv$ mit $u \in r$ und $v = \varepsilon$ oder $v = v_1 \cdot \ldots \cdot v_n$ mit $v_i \in sr$ und $n \geq 1$ Fallunterscheidung:

1. $w = u\varepsilon$ mit $u \in r$, dann ist $e = \varepsilon u \in (rs)^*r$.
2. $w = uv_1'v_1''v_2'v_2'' \ldots v_n'v_n''$ mit $v_i' \in s$ und $v_i'' \in r$, dann ist

$$w = \underbrace{uv_1'}_{\in rs}\underbrace{v_1''v_2'}_{\in rs}\underbrace{v_2'' \ldots \ldots \ldots v_n'}\underbrace{v_n''}_{\in r}$$

LEMMA: (Resolutionsregel) Die Gleichung $X = rX + s$ (mit r, s reguläre Ausdrücke) wird gelöst durch $X = r^*s$.

Beweis: $X = r^*s = (rr^* + \varepsilon)s = rr^*s + s = r(r^*s) + s = rX + s$

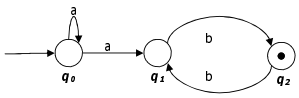
Beispiele:

1. Sei $X_0 = aX_0 + bX_1 + \varepsilon$ und $X_1 = bX_0 + aX_1$. Wende auf die zweite Gleichung die Resolutionsregel an, man erhält: $X_1 = a^*bX_0$. Eingesetzt in die erste Gleichung ergibt sich

$$X_0 = aX_0 + ba^*bX_0 + \varepsilon = (a + ba^*b)X_0 + \varepsilon$$

   Nochmalige Anwendung der Resolutionsregel liefert $X_0 = (a + ba^*b)^* \cdot \varepsilon$, somit ist $r = (a + ba^*b)^*$ und $r_1 = a^*b(a + ba^*b)^*$.

2. Für den folgenden Automaten lautet das Gleichungssystem:



$$\begin{array}{lrcl} (1) & X_0 & = & aX_0 + aX_1 \\ (2) & X_1 & = & bX_2 \\ (3) & X_2 & = & bX_1 + \varepsilon \end{array}$$

   Gleichung (3) in (2) einsetzen ergibt $X_1 = bbX_1 + b$, Resolutionsregel liefert Gleichung (4): $X_1 = (bb)^*b$. Setze nun (4) in (1) ein, erhalte $X_0 = aX_0 + a(bb)^*b$, damit ist $X_0 = a^*a(bb)^*b$, für X_2 erhält man $b(bb)^*b + \varepsilon = (bb)^*$.

Allgmeines Verfahren der Auflösung von $GS(\mathfrak{A})$:

- für eine einzige Gleichung in $GS(\mathfrak{A})$ wende das Resolutionslemma an.
- für $m+1$ Gleichungen in $\leq m+1$ Variablen nehme Reduktion auf m Gleichungen in $\leq m$ Variablen vor: Betrachte

$$X_{m+1} = a_{m+1,m+1}X_{m+1} + \sum_{j=1}^{m} a_{m+1,j}X_j + e_{m+1}.$$

  Resolutionslemma liefert

$$X_{m+1} = a^*_{m+1,m+1}\left(\sum_{j=1}^{m} a_{m+1,j}X_j + e_{m+1}\right).$$

  Einsetzen in die ersten m Gleichungen liefert die gewünschte Reduktion.
- Iterierte Anwendung liefert regulären Ausdruck für X_1.

Zusatz: Ist (wie bei NEAs) in keinem A_{ij} das leere Wort enthalten, sind die Lösungen eindeutig.

SATZ: Jedes Gleichungssystem für reguläre Ausdrücke $X_i = \sum_{j=0}^{n} a_{ij}X_j + e_i$ (mit $a_{ij} \subseteq \Sigma$ und $e_i \subseteq \{\varepsilon\}$) ist durch iterierte Anwendung der Resolutionsregel lösbar.

1.6 Abschlußeigenschaften und Entscheidbarkeit

SATZ: Die regulären Sprachen sind abgeschlossen unter Vereinigung, Produkt, Durchschnitt, Komplement und Stern.

Beweis:

1. Abschluß unter Vereinigung, Produkt und Stern folgt über reguläre Ausdrücke. Seien α, β reguläre Ausdrücke für reguläre Sprachen L und L'. Dann folgt, daß $\alpha + \beta$, $\alpha \cdot \beta$ und $(\alpha)^*$ die regulären Ausdrücke für die Sprachen $L \cup L'$, $L \cdot L'$ und L^*.
2. Abschluß unter Komplement und Durchschnitt folgt über die Konstruktion von Automaten für $\Sigma^* \setminus L$ und $L \cap L'$ (als Lemma vorher schon bewiesen)

Leerheitsproblem: Gegeben sei ein Automat $\mathfrak{A}$, **Frage:** Ist $L(\mathfrak{A}) = \emptyset$?

Bemerkung: $L(\mathfrak{A}) = \emptyset \Leftrightarrow \nexists\, w \in \Sigma^*$ mit $\mathfrak{A} : q_0 \xrightarrow{w} F$

LEMMA: $L(\mathfrak{A}) \neq \emptyset \Leftrightarrow \exists\, w \in L(\mathfrak{A})$ mit $|w| < |Q|$.

Beweis:

„$\Leftarrow$“ klar

„$\Rightarrow$“ $\mathfrak{A}$ habe n Zustände. Wähle $w \in L(\mathfrak{A})$ mit minimaler Länge. *Behauptung:* die minimale Länge ist kleiner n. *Annahme*, die minimale Länge sei größergleich n. Mit dem Pumping-Lemma folgt eine Zerlegung $w = xyz$ mit $y \neq \varepsilon$ und $xy^0z = xz \in L(\mathfrak{A})$. Dann gilt aber $|xz| < |w|$, Widerspruch!

Endlichkeitsproblem: Gegeben sei $\mathfrak{A}$ mit n Zuständen. **Frage:** Ist $|L(\mathfrak{A})| < \infty$?

LEMMA: $|L(\mathfrak{A})|$ ist unendlich genau dann, wenn ein $w \in L(\mathfrak{A})$ existiert mit $n \leq |w| < 2n$.

Beweis: als Übung

Äquivalenzproblem: Gegeben seien Automaten $\mathfrak{A}$, $\mathfrak{B}$. **Frage:** Ist $L(\mathfrak{A}) = L(\mathfrak{B})$?

LEMMA: $L(\mathfrak{A}) \neq L(\mathfrak{B})$ genau dann, wenn gilt:

$$(L(\mathfrak{A}) \cap \overline{L(\mathfrak{B})}) \cup (\overline{L(\mathfrak{A})} \cap L(\mathfrak{B})) \neq \emptyset$$

Bemerkung: Reguläre Sprachen und Automaten sind abgeschlossen bezüglich Vereinigung, Durchschnitt und Komplement, d.h. ein DEA $\mathfrak{C}$ mit $L(\mathfrak{C}) = (L(\mathfrak{A}) \cap \overline{L(\mathfrak{B})}) \cup (\overline{L(\mathfrak{A})} \cap L(\mathfrak{B}))$ ist effektiv konstruierbar.

SATZ: Für DEAs (und damit für NEAs und reguläre Sprachen) sind Leerheitsproblem, Endlichkeitsproblem und Äquivalenzproblem effektiv entscheidbar.

Beweis: am Beispiel vom Leerheitsproblem: $L(\mathfrak{A}) \neq \emptyset \Leftrightarrow \exists\, w \in L(\mathfrak{A})$ mit $|w| < |Q|$. Teste also alle Wörter der Länge $< |Q| = n$: Für jedes Wort

verfolge Zeichen für Zeichen die Zustandsübergänge im Automaten, die durch w hervorgerufen werden. Falls ein Endzustand erreicht wird, ist $w \in L(\mathfrak{A})$.

Effizienzaspekt: die Algorithmen sind hoffnungslos ineffizient!

1.7 Sequentielle Maschinen

Idee: Wir verwenden verallgemeinerte Transitionen $q \xrightarrow{a/v} q'$ mit *Eingabebuchstabe* a, *Ausgabewort* v.

DEFINITIONEN:

- Eine *verallgemeinerte sequentielle Maschine* (kurz *GSM*: generalized sequential machine) hat die Form $\mathfrak{A} = (Q, \Sigma, \Gamma, q_0, \delta, \lambda)$ mit Q endlich, Σ, Γ Eingabe- bzw. Ausgabealphabet, $q_0 \in Q$, $\delta : Q \times \Sigma \to Q$, $\lambda : Q \times \Sigma \to \Gamma^*$.
- Variante: *Sequentielle Maschine* oder *Mealy-Automat*, falls $\lambda : Q \times \Sigma \to \Gamma$. Erweitere δ auf $Q \times \Sigma^*$ wie zuvor, entsprechend definiere $\lambda^* : Q \times \Sigma^* \to \Gamma^*$ wie folgt:

$$\lambda^*(q, \varepsilon) = \varepsilon, \quad \lambda^*(q, wa) = \lambda^*(q, w)\lambda(\delta(q, w), a)$$

- Weiter sei $f_{\mathfrak{A}} : \Sigma^* \to \Gamma^*$ definiert durch $f_{\mathfrak{A}}(w) := \lambda^*(q_0, w)$. Es ist also $f_{\mathfrak{A}}(w)$ das durch $\mathfrak{A}$ bei Lesen von w gelieferte Ausgabewort.

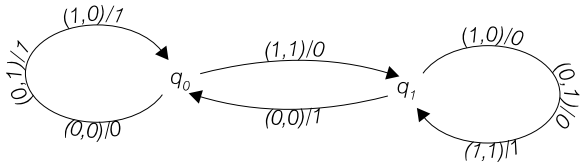
Beispiel: Addition gespiegelter Binärzahlen mit wenigstens einer führenden 0. Ein- und Ausgabealphabet sind

$$\Sigma = \left\{ \binom{0}{0}, \binom{0}{1}, \binom{1}{0}, \binom{1}{1} \right\} \quad \Gamma = \{0, 1\}$$

Beispiel (führende Nullen stehen rechts!):

$$\begin{array}{r} 101100 \\ \underline{100010} \\ 011110 \end{array}$$

Die entsprechende Funktion $f_+ : \Sigma^* \to \Gamma^*$ wird berechnet durch einen Automaten mit zwei Zuständen q_0 und q_1, die Überträge 0 und 1 darstellen:



DEFINITION: $f : \Sigma^* \to \Gamma^*$ sei eine *(verallgemeinert) sequentielle Funktion* genau dann, wenn eine (verallgemeinerte) sequentielle Maschine $\mathfrak{A}$ existiert mit $f = f_{\mathfrak{A}}$.

LEMMA: Es sei $f : \Sigma^* \to \Gamma^*$ eine *(verallgemeinert) sequentielle Funktion.* Dann gilt:

1. f ist *präfixtreu*, d.h. $f(uv)$ hat jeweils $f(u)$ als Präfix
2. f ist *längenbeschränkt*, d.h. es existiert $k \geq 0$ mit $|f(w)| \leq k \cdot |w|$ für alle $w \in \Sigma^*$.
3. $L \subseteq \Sigma^*$ regulär $\Rightarrow f(L)$ regulär.

Anwendung: Die *Multiplikation gespiegelter Binärzahlen* liefert *keine* verallgemeinert sequentielle Funktion $f.$ mit Σ, Γ wie bei der Addition. Wegen der dritten Eigenschaft im Lemma genügt die Angabe von $L \subseteq \Sigma^*$ regulär mit $f.(L)$ nicht regulär. Betrachte die reguläre Sprache $L = \left\{ \binom{0}{1}^n \binom{1}{0} \mid n \geq 0 \right\}$.

Dabei ist die obere Zahl 0^n1 (also 2^n), das untere Wort 1^n0 (also $1 + 2 + \ldots + 2^{n-1}$). Das Produkt ist demnach $2^n + 2^{n+1} + \ldots + 2^{2n-1}$ (was 0^n1^n entspricht). Damit ist $f.(L) = \{0^n1^n \mid n > 0\} \cup \{0\}$, also ist $f.(L)$ nicht regulär (wie schon gesehen u.a. beim Pumping-Lemma).

2 Grammatiken

DEFINITION: Eine *Grammatik* hat die Form $G = (V, \Sigma, P, S)$ mit der Menge der Variablen (Nicht-Terminalsymbole) V, Terminalalphabet Σ und einem Startsymbol $S \in V$, dabei sind V, Σ endlich und $V \cap \Sigma = \emptyset$; sei $P \subset (V \cup \Sigma)^* V (V \cup \Sigma)^* \times (V \cup \Sigma)^*$.

Beispiele:

- BNF-Regeln:

  $$G_0: \quad \langle \text{Term} \rangle ::= 0 \mid 1 \mid (\langle \text{Term} \rangle + \langle \text{Term} \rangle) \mid (\langle \text{Term} \rangle * \langle \text{Term} \rangle)$$

  Beispiel für eine *Ableitung*:

  $$\begin{aligned} \langle \text{Term} \rangle &\vdash (\langle \text{Term} \rangle + \langle \text{Term} \rangle) \\ &\vdash (0 + \langle \text{Term} \rangle) \\ &\vdash (0 + (\langle \text{Term} \rangle * \langle \text{Term} \rangle)) \\ &\vdash (0 + (1 * \langle \text{Term} \rangle)) \\ &\vdash (0 + (1 * 1)) \end{aligned}$$

Schreibweisen:

- $\alpha \to \beta$ in P statt $(\alpha, \beta) \in P$ (in BNF $\alpha ::= \beta$)
- Symbole in V: $A, B, C, \ldots$; Symbole in Σ: $a, b, c, \ldots, 0, 1$; Wörter über $V \cup \Sigma$: $\alpha, \beta, \gamma, \ldots$ (Satzformen); Wörter über Σ : $u, v, w, \ldots$
- $\alpha \vdash_G \beta$ besagt: Es existieren $\gamma, \delta, \alpha_1, \beta_1$ mit $\alpha = \gamma\alpha_1\delta$, $\alpha_1 \to \beta_1 \in P$ und $\beta = \gamma\beta_1\delta$

  Beispiel:

  $$\underbrace{(}_{\gamma}\underbrace{\langle \text{Term} \rangle}_{\alpha_1}\underbrace{+\langle \text{Term} \rangle)}_{\delta} \vdash \underbrace{(}_{\gamma}\underbrace{0}_{\beta_1}\underbrace{+\langle \text{Term} \rangle)}_{\delta}$$

- $\alpha \vdash_G^n \beta$ bedeutet: es existieren $\alpha_0, \ldots, \alpha_n$ mit $\alpha_0 = \alpha, \alpha_i \vdash_G \alpha_{i+1} (i < n), \alpha_n = \beta$
- $\alpha \vdash_G^* \beta$ bedeutet: es existiert $n \geq 0$ mit $\alpha \vdash_G^n \beta$
- α heißt *ableitbar* in G genau dann, wenn $S \vdash_G^* \alpha$ gilt.

Die von G erzeugte Sprache ist

$$L(G) = \{ w \in \Sigma^* \mid S \vdash^* w \}$$

Beispielsweise ist $(0 + (1 * 1)) \in L(G_0)$.

Notatation: Regeln mit gleicher linker Seite werden zusammengefaßt und auf der rechten Seite durch „|“ getrennt (wie in BNF oben).

Beispiele:

- Sei folgende Grammatik gegeben:

$$\begin{array}{lrcl} G_1 & S & \to & 0|1|1A \\ & A & \to & 0A|1A|0|1 \end{array}$$

  Beispielhafte Ableitung: $S \vdash 1A \vdash 10A \vdash 101A \vdash 1011$, damit ist

$$L(G_1) = \{0\} \cup \{ w \in \{0,1\}^* \mid w \text{ beginnt mit } 1 \}$$

  Beweis:

  „$\supseteq$“ $w = 0, 1$ ableitbar in G_1; Induktion über $|w|$: jedes Wort $1wA$ mit $w \in \{0,1\}^*$ ist ableitbar in G_1:

  * $|w| = 0$: $S \vdash 1A$
  * $|w| = k + 1$: Die Induktionsvoraussetzung gelte für alle w' mit $|w'| \leq k$. Sei $|w| = k + 1$ gegeben mit $w = w'a$ wobei $a \in \{0,1\}$. Dann gilt $S \vdash^* 1w'A \vdash 1w'aA = 1wA$, also ist $1wA$ ableitbar.

  Damit folgt

$$\left.\begin{array}{l} S \vdash^*_{G_1} 1wA \; \forall \; w \in \{0,1\}^* \\ S \vdash^*_{G_1} 1 \end{array}\right\} \Rightarrow S \vdash^*_{G_1} 1w \; \forall \; w \in \{0,1\}^*$$

  „$\subseteq$“ Fallunterscheidung:

  * Länge der Ableitung ist 1: Dann ist $w = 0$ oder $w = 1$
  * Länge der Ableitung größer 1: Dann ist die letzte angewandte Regel $A \to 0$ oder $A \to 1$ und die erste angewandte Regel ist $S \to 1A$. Die Ableitung hat dann die Form

$$S \vdash 1A \underbrace{\vdash}_{(\star)} \ldots \underbrace{\vdash}_{(\star)} 1uA \vdash \left\{ \begin{array}{l} 1u0 \\ 1u1 \end{array} \right. = w$$

- Mit $\Sigma = \{a, b\}$ betrachte

$$\begin{array}{lrcl} G_2 & S & \to & aB|bA \\ & A & \to & a|aS|bAA \\ & B & \to & b|bS|aBB \end{array}$$

Nun ist $L(G_2) = \{\, w \in \Sigma^+ \mid |w|_a = |w|_b\}$. *Beweis:* Per Induktion über die Länge von $w \in \{a, b\}^*$ zeigen wir

(1) $S \vdash^* w \Leftrightarrow |w|_a = |w|_b$

(2) $A \vdash^* w \Leftrightarrow |w|_a = |w|_b + 1$

(3) $B \vdash^* w \Leftrightarrow |w|_a + 1 = |w|_b$

Beachte: Jeder (nicht-triviale) Ableitungsschritt vergrößert die Länge der Satzformen.

– $|w| = 1$:

(1) gilt, da beide Seiten der Äquivalenz falsch sind

(2) für $w = a$ gelten beide Seiten

(3) für $w = b$ gelten beide Seiten

– $|w| = k + 1$: die Aussagen (1) bis (3) mögen für alle Wörter der Länge kleinergleich k gelten.

(1) „⇒“ Sei $S \vdash^* w$. Dann beginnt die Ableitung mit $S \to aB$ oder $S \to bA$. Fallunterscheidung:

· $S \vdash aB \vdash \ldots \vdash aw_1 = w$, also $B \vdash^* w_1$ und aus $|w_1| = k$ folgt mit der Induktionsvoraussetzung (3), daß $|w_1|_a + 1 = |w_1|_b$ ist, also gilt $|w|_a = |w_1|_a + 1 = |w_1|_b = |w|_b$.

· $S \vdash bA \vdash \ldots \vdash bw_1 = w$, analog mit (2)

„⇐“ Sei $|w|_a = |w|_b$ und $|w| = k + 1$. Fallunterscheidung:

· $w = aw_1 \Rightarrow |w_1| = k$ und $|w_1|_a + 1 = |w_1|_b$. Dann folgt mit Induktionsvoraussetzung (3), daß $B \vdash^* w_1$, also $S \vdash aB \vdash^* aw_1 = w$

· $w = bw_1$ analog mit (2)

(2) analog

(3) analog

- Mit $\Sigma = \{a, b, c\}$

$$\begin{array}{rrcl} G_3 & S & \to & abc|aAbc \\ & Ab & \to & bA \\ & Ac & \to & Bbcc \\ & bB & \to & Bb \\ & aB & \to & aaA|aa \end{array}$$

Beispielhafte Ableitung:

$$S \vdash \left\{ \begin{array}{l} abc \\ aAbc \vdash abAc \vdash abBbcc \vdash aBbbcc \vdash \left\{ \begin{array}{l} aabbcc \\ aaAbbcc \vdash^2 aabbAcc \vdash^3 aaBbbbccc \vdash \left\{ \begin{array}{l} aaabbbccc \\ \dots \end{array} \right. \end{array} \right. \end{array} \right.$$

Damit ist $L(G_3) = \{\, a^n b^n c^n \mid n \geq 1\}$. *Beweis:*

„$\supseteq$“ zeige per Induktion $S \vdash^* a^n B b^{n+1} c^{n+1}$:

- $n = 1$ ist klar
- $n > 1$:

$$S \vdash^* a^{n-1} B b^n c^n \vdash a^n A b^n c^n \vdash^n a^n b^n A c^n \vdash a^n b^n B b c^{n+1} \vdash^n a^n B b^{n+1} c^{n+1}$$

Durch Anwendung von $aB \to aa$ folgt $S \vdash^* a^{n+1} b^{n+1} c^{n+1}$.

„$\subseteq$“ Sei $S \vdash^* w$. Zeige: $w = a^n b^n c^n$ für ein $0 < n \in \mathbb{N}$. Fallunterscheidung:

- Aus $S \vdash w$ folgt $w = abc$.
- Sei $S \vdash aAbc \vdash^* w$. Von $a^n A b^n c^n$ sehen die nächsten (eindeutigen) $2n + 1$ wie folgt aus:

$$a^n A b^n c^n \quad \underbrace{\vdash^{2n+1}}_{\substack{n\text{-mal } Ab \to bA \\ \text{einmal } Ac \to Bbcc \\ n\text{-mal } bB \to Bb}} \quad a^n B b^{n+1} c^{n+1} \vdash \left\{ \begin{array}{l} a^{n+1} b^{n+1} c^{n+1} \\ a^{n+1} A b^{n+1} c^{n+1} \end{array} \right.$$

Wegen der Eindeutigkeit folgt daraus, daß die Ableitung nur Worte $a^n b^n c^n$ erzeugt.

2.1 Chomsky-Hierarchie

DEFINITION:

- Eine Grammatik $G = (V, \Sigma, P, S)$ ist
  - vom *Typ 0 (rekursiv aufzählbar)* im allgemeinen Fall,
  - vom *Typ 1 (kontextsensitiv)*, falls für jede Regel $\alpha \to \beta$ gilt: $|\alpha| \leq |\beta|$ (außer eventuell $S \to \varepsilon$, siehe unten),
  - vom *Typ 2 (kontextfrei)*, falls jede Regel die Form $A \to \alpha$ hat mit $A \in V$ und $\alpha \in (V \cup \Sigma)^*$,
  - vom *Typ 3 (rechtslinear)*, falls jede Regel die Form $A \to w$ oder $A \to wB$ hat mit $w \in \Sigma^+$ und $A, B \in V$.

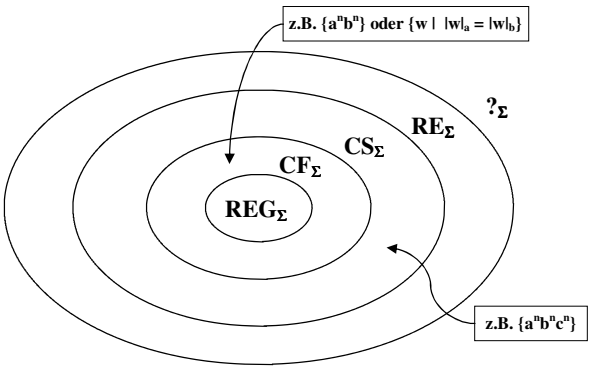
  ε-Sonderbedingung: Bei Typ 1 bis 3 ist ε auf der rechten Seite nur bei $S \to \varepsilon$ erlaubt, dann darf S in keiner rechten Regelseite auftreten.

- Eine Sprache $L \subset \Sigma^*$ heißt *rechtslinear* (*kontextfrei, kontextsensitiv* oder *rekursiv aufzählbar*), falls es eine Grammatik mit Alphabet Σ gibt, die rechtslinear (kontextfrei, kontextsensitiv oder rekursiv aufzählbar) ist und L erzeugt.

Bezeichnungen: REG_ε, CF_Σ, CS_Σ, RE_Σ bezeichnen die Mengen der rechtslinearen, kontextfreien, kontextsensitiven oder rekursiv aufzählbaren Sprachen.
Bemerkung: $REG_\varepsilon \subset CF_\Sigma \subset CS_\Sigma \subset RE_\Sigma$.

Kernfragen:

1. Bilden diesse Sprachklassen eine echte Hierarchie? Ja, die *Chomsky-Hierarchie*!



2. Gibt es andere Charakterisierungen („Automaten“)?

3. Bezüglich welcher Operationen $\cup, \cap, /, \cdot, ^*$ sind diese Klassen abgeschlossen?

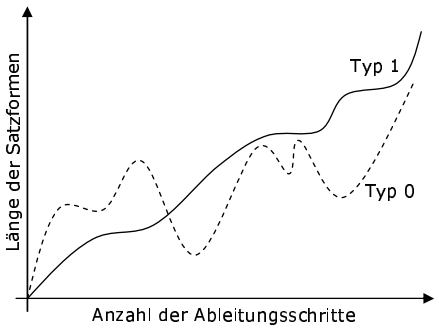
4. Reduktion auf einfachere Regelformen (bzw. Normalformen)

5. Entscheidungsprobleme: Leerheitsproblem, Äquivalenzproblem, Endlichkeitsproblem, Wortproblem...

Wortproblem: Gegeben sei eine Grammatik $G = (V, \Sigma, P, S)$ und $w \in \Sigma^*$,
Frage: Ist $w \in L(G)$?

SATZ: Das Wortproblem für kontextsensitive Sprachen ist entscheidbar (d.h. es existiert ein Algorithmus).

Bemerkung: Für Typ-0-Grammatiken ist das Wortproblem i.a. nicht entscheidbar, beachte dazu, daß die Länge der Wortformen bei einer Typ-0-Grammatik schwanken kann:



Beweis: Definiere zunächst Mengen

$$T_m^n = \left\{ w \in (V \cup \Sigma)^* \mid |w| \leq n \ \wedge \ S \vdash^{\leq m} w \right\}$$

Die Mengen T_m^n lassen sich rekurisv über m berechnen: $T_0^n = \{S\}$ und $T_{m+1}^n = \mathrm{Abl}_n(T_m^n)$ mit

$$\mathrm{Abl}_n(X) = X \cup \{ w \in (V \cup \Sigma)^* \mid |w| \leq n \ \wedge \ \exists w' \in X \ : \ w' \vdash w \}$$

Es gibt nur endlich viele (exponentiell in m) Wörter der Länge $\leq n$ in $(V \cup \Sigma)^*$. Damit ist $\bigcup_{m \geq 0} T_m^n$ eine endliche Menge für jedes n und es existiert $\bar{m}$ mit $T_{\bar{m}}^n = T_{\bar{m}+1}^n$.

Bemerkungen:

1. $\bar{m} \leq \left| \bigcup_{m \geq 0} T_m^n \right| \leq \sum_{l=0}^{n} (|V| + |\Sigma|)^l \leq (|V| + |\Sigma|)^{n+1}$

2. $T^n_{\bar{m}} = T^n_{\bar{m}+1} \Rightarrow T^n_{\bar{m}} = T^n_{\bar{m}+i}$

Algorithmus:

```
T := { S };
repeat T' = T; T = Abl[n](T')
until (w in T) or (T = T');
if (w in T) then write(w + ' ist in L(G)')
            else write(w + ' ist nicht in L(G)');
```

Bemerkung: Der Algorithmus hat exponentielle Laufzeit (beachte: Wortproblem für kontextsensitive Sprachen ist NP-schwer, siehe Komplexitätstheorie später).

SATZ: Eine Sprache ist rechtslinear genau dann, wenn sie durch einen NEA erkannt wird.

Beweis:

„⇒“ Sei $L = L(G)$ rechtslinear bei gegebener Grammatik $G = (V, \Sigma, P, S)$. Es ist $w \in L(G)$ genau dann, wenn eine Ableitung folgender Gestalt existiert:

$$S = B_0 \vdash w_1 B_1 \vdash w_1 w_2 B_2 \vdash \ldots \vdash w_1 \ldots w_{n-1} B_{n-1} \vdash w_1 \ldots w_n = w$$

wobei die $w_i \in \Sigma^+$ und die $B_i \in V$ sind. Es gilt $B_i \to w_{i+1} B_{i+1} \in P$ für $i < n-1$ und $B_{n-1} \to w_n \in P$. Konstruiere nun einen NEA mit Worttransitionen: Definiere

$$\mathfrak{A} = ((V \cup \{\Omega\}) = Q, \Sigma^+, S = q_0, \Delta, \{\Omega\} = F)$$

mit einem neuen Zustand $\Omega \notin V$; dabei ist

$$\begin{aligned} (A, w, B) \in \Delta &\Leftrightarrow A \to wB \in P \\ (A, w, \Omega) \in \Delta &\Leftrightarrow A \to w \in P \end{aligned}$$

Es gilt nun $w \in L(G)$ genau dann, wenn ein Pfad $s = B_0 w_1 B_1 w_2 \ldots w_{n-1} B_{n-1} w_n \Omega$ durch $\mathfrak{A}$ mit Beschriftung $w = w_1 \ldots w_n$. Also akzeptiert $\mathfrak{A}$ auch w.

„⇐“ Sei NEA $\mathfrak{A} = (Q, \Sigma, q_0, \Delta, F)$ gegeben. O.B.d.A. ohne Transistionen nach q_0 (führe sonst einen neuen Hilszustand ein). Definiere $G = (V, \Sigma, P, S)$ durch $V = Q$, $S = q_0$ und

$$\begin{aligned} A \to aB \in P &\Leftrightarrow (A, a, B) \in \Delta \\ A \to a \in P &\Leftrightarrow (A, a, B) \in \Delta \text{ für } B \in F \end{aligned}$$

Falls $q_0 \in F$, nehme auch Regel $S \to \varepsilon$ hinzu. Wegen der Annahme oben gilt die ε-Bedingung. Es gilt für $w = a_1 \dots q_n$ mit $n > 0$:

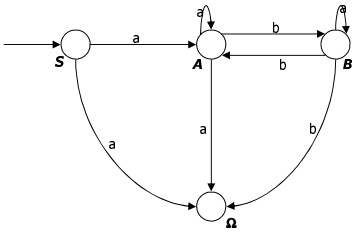
$$\begin{aligned} & \mathfrak{A} \text{ akzeptiert } w \\ \iff & \exists \text{ Pfad } q_0 = B_0 a_1 B_1 \dots B_{n-1} a_n B_n \text{ in } \mathfrak{A} \text{ mit } B_n \in F \\ \iff & \exists\, G - \text{Abl.} S = B_0 \vdash a_1 B_1 \vdash \dots \vdash a_1 \dots a_n = w \end{aligned}$$

Beispiele:

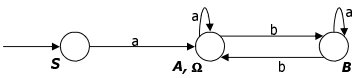
1. Sei folgende Grammatik gegeben:

$$\begin{array}{llll} G & S & \to & aA|a \\ & A & \to & aA|a|bB \\ & B & \to & aB|bA|b \end{array}$$

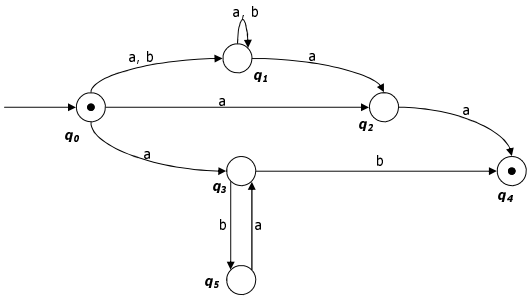
Der entsprechende NEA ist



Er läßt sich reduzieren zu



2. Sei folgender NEA gegeben:



Dann ist die zugehörige Grammatik $G = (V, \Sigma, P, S)$ zu $\mathfrak{A}$: $V = \{q_0, \ldots, q_5\}$, $S = q_0$,

$$\begin{aligned} q_0 &\to \varepsilon | aq_1 | bq_1 | aq_2 | aq_3 \\ q_1 &\to aq_1 | bq_1 | aq_2 \\ q_2 &\to aq_4 | a \\ q_3 &\to b | bq_4 | bq_5 \\ q_5 &\to aq_3 \end{aligned}$$

2.2 Kontextfreie Sprachen

DEFINITIONEN:

1. Eine kontextfreie Grammatik ist im *Chomsky-Normalform (CNF)*, falls sie nur Regeln $A \to BC$, $A \to a$ (und eventuell $S \to \varepsilon$) hat.
2. Eine kontextfreie Grammatik ist im *Greibach-Normalform (GNF)*, falls sie nur Regeln $A \to aB_1 \ldots B_n$ mit $n \geq 0$ (und eventuell $S \to \varepsilon$) hat.

LEMMA: Zu jeder kontextfreien Grammatik G kann man eine äquivalente kontextfreie Grammatik G' ohne einfache Regeln $A \to B$ angeben.

Beweis: Sei $G = (V, \Sigma, P, S)$ gegeben. Zu $A, B \in V$ können wir effektiv entscheiden, ob $A \vdash_G^* B$ (denn Ableitung ohne Wiederholungen von Variablen haben Länge $\leq |V|$). Sei P_1 die Menge der P-Regeln ohne einfache Regeln und

$$P_2 = \{ A \to \alpha \mid \exists\, B \in V \,:\, A \vdash_G^* B \,\wedge\, B \to \alpha \in P \,\wedge\, \alpha \notin V \}$$

Definiere $G' = (V, \Sigma, P_1 \cup P_2, S)$, *Behauptung:* $L(G) = L(G')$.

„$\supseteq$“ klar nach Definition

„$\subseteq$“ betrachte G-Ableitung $S = \alpha_0 \vdash_G \alpha_1 \vdash_G \ldots \vdash_G \alpha_n = w \in \Sigma^*$ mit Anwendung von mindestens einer einfachen Regel. Betrachte das erste Vorkommen einer einfachen Regel:

$$S \vdash_G^* \beta A_1 \gamma \vdash_G \beta A_2 \gamma \vdash_G \ldots \vdash_G \beta' A_k \gamma' \vdash_G \beta' \alpha \gamma' \vdash_G^* w$$

mit $A_1, \ldots, A_k \in V$, $\alpha \notin V$, $\beta \vdash_G^* \beta'$, $\gamma \vdash_G^* \gamma'$, $A_1 \vdash_G^* A_k$ und $A_k \vdash_G \alpha$. Dann gilt:

$$S \vdash_{G'}^* \beta A_1 \gamma \vdash_{G'} \beta \alpha \gamma \vdash_G^* \beta' \alpha \gamma' \vdash_G^* w$$

Rekursiv eliminieren wir einfache Regeln in $\beta \vdash_G^* \beta'$, $\gamma \vdash_G^* \gamma'$ und $\beta' \alpha \gamma' \vdash_G^* w$.

SATZ: (Chomsky-Normalform) Jede kontextfreie Sprache wird durch eine Grammatik in Chomsky-Normalform erzeugt, d.h. mit Regeln der Form $A \to BC$, $A \to a$ (und eventuell $S \to \varepsilon$).

Beweis:

1. Elimination der einfachen Regeln $A \to B$: Für alle Regeln $A \to \alpha$ mit $|\alpha| = 1$ gilt: $\alpha = a \in \Sigma$

2. Für alle Regeln mit $|\alpha| \geq 2$ soll gelten: $\alpha \in V^*$. Betrachte $A \to X_1 \dots X_m$ (mit $m \geq 2$), $X_i \in V \cup \Sigma$, verwende für jedes $a \in \Sigma$ eine Variable X_a und ersetze die $X_i \in \Sigma$ entsprechend. Füge für diese $a \in \Sigma$ die Regeln $X_a \to a$ ein.

   Behauptung: Für die so gebildete Grammatik G' gilt $L(G) = L(G')$.

3. Reduktion der Regeln $A \to B_1 \dots B_m$ auf $m = 2$: Sei $A \to B_1 \dots B_m$ mit $m \geq 3$ gegeben. Verwende neue Variablen $D_1, \dots, D_{m-2}$ und ersetze $A \to B_1 \dots B_m$ durch $A \to B_1 D_1$, $D_1 \to B_2 D_2$, $\dots$, $D_{m-3} \to B_{m-2} D_{m-2}$ und $D_{m-2} \to B_{m-1} B_m$.

Dann entsteht eine äquivalente Grammatik in Chomsky-Normalform.

Beispiel: Sei folgende Grammatik G gegeben.

$$\begin{aligned} S &\to bA|aB \\ A &\to bAA|aS|a \\ B &\to aBB|bS|b \end{aligned}$$

Die einzelnen Schritte:

1. nichts zu tun, da keine einfachen Regeln

2. neuer Regelsatz:

$$\begin{aligned} S &\to X_bA|X_aB \\ A &\to X_bAA|X_aS|a \\ B &\to X_aBB|X_bS|b \\ X_a &\to a \\ X_b &\to b \end{aligned}$$

3. neue Regeln:

$$\begin{aligned} S &\rightarrow X_bA|X_aB \\ A &\rightarrow X_bD_1|X_aS|a \\ D_1 &\rightarrow AA \\ B &\rightarrow X_aE_1|X_bS|b \\ E_1 &\rightarrow BB \\ X_a &\rightarrow a \\ X_b &\rightarrow b \end{aligned}$$

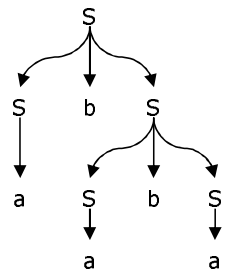
SATZ: (Greibach-Normalform)Jede kontextfreie Sprache wird durch eine Grammatik in Greibach-Normalform erzeugt.

Beispiel: Sei eine Grammatik gegeben mit $S \rightarrow SbS|a$, mögliche Ableitungen des Wortes *ababa* sind z.B.:

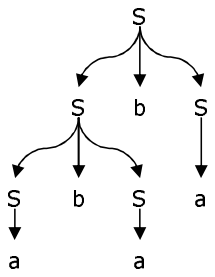
1. $S \vdash \underline{S}bS \vdash ab\underline{S} \vdash ab\underline{S}bS \vdash abab\underline{S} \vdash ababa$
2. $S \vdash Sb\underline{S} \vdash SbSb\underline{S} \vdash Sb\underline{S}ba \vdash \underline{S}baba \vdash ababa$
3. $S \vdash \underline{S}bS \vdash \underline{S}bSbS \vdash ab\underline{S}bS \vdash abab\underline{S} \vdash ababa$
4. $S \vdash Sb\underline{S} \vdash \underline{S}ba \vdash Sb\underline{S}ba \vdash \underline{S}baba \vdash ababa$

Bemerkungen:

1. Die Ableitungen 1 und 3 sind *Linksableitungen*, d.h. in jedem Schritt wird die am weitesten links stehende Variable ersetzt. Entsprechend sind 2 und 4 *Rechtsableitungen*
2. Jede Ableitungen induziert einen *Ableitungsbaum*, einem Ableitungsbaum entsprechen i.a. aber mehrere Ableitungen, im Beispiel:



für 1 und 3

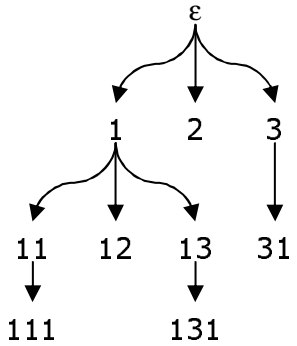


für 2 und 4

DEFINITION:

1. Ein *unbewerteter, höchstens-k-verzweigter Baum* ist eine Teilmenge $D \subseteq \{1, \ldots, k\}^*$, die
   - unter Präfixbildung abgeschlossen ist, d.h. $xi \in D \Rightarrow x \in D$, und
   - jeweils für xj auch xi (mit $1 \leq i < j$) enthält.
2. Ein Σ-*bewerteter, höchstens-k-verzweigeter Baum* ist eine Abbildung $t : \mathrm{dom}(t) \to \Sigma$, wobei $\mathrm{dom}(t)$ ein unbewerteter, höchstens-k-verzweigter Baum ist.
3. Ein *Ableitungsbaum* t zur kontextfreien Grammatik $G = (V, \Sigma, P, S)$ ist ein $(V \cup \Sigma)$-bewerteter Baum mit:
   (a) die Wurzel ist mit S beschriftet (d.h. $t(\varepsilon) = S$)
   (b) hat einen Knoten x die Nachfolger $x1, \ldots, xn$, so ist $t(x) \to t(x1)t(x2)\ldots t(xn) \in P$.

Beispiel: Der oben verwendete Ableitungsbaum als unbewerteter, höchstens 3 verzweigter Baum ist $D = \{\varepsilon, 1, 2, 3, 11, 12, 13, 31, 111, 131\}$, als Ableitungsbaum formal ist $t(\varepsilon) = S$, $t(1) = S$, $t(2) = b$, $t(3) = S$, $\ldots$, der Regel $S \to SbS$ entspricht $t(\varepsilon) \to t(1)t(2)t(3) \in P$. Der Baum D ist:



Sprechweisen: ε ist die *Wurzel* des Baums, Knoten xi ist *Nachfolger* von x, *Blätter* sind Knoten ohne Nachfolger, die *Front* eines Baumes ist die Folge der Blätter in lexikographischer Reihenfolge. Ein *Pfad* der Länge n ist eine Folge $y_0, y_1, \ldots, y_n$, wobei y_0 die Wurzel, y_n ein Blatt und y_{i+1} ein Nachfolger von y_i $(0 \leq i < n)$ ist. Yield(t) ist die Folge der Beschriftungen der Knoten in der Front.

Bemerkung: Es sei G eine kontextfreie Grammatik und $A \in V$. Dann gilt:

$A \vdash_G^* w$ genau dann, wenn ein Ableitungsbaum t für G existiert, dessen Wurzel mit A beschriftet ist ($t(\varepsilon) = A$) und Yield(t) $= w$.

Beweis: als Übung, Ideen:

„$\Leftarrow$“ Induktion über die Anzahl der inneren Knoten von t

„$\Rightarrow$“ Induktion über die Länge der Ableitung

2.3 Wortproblem, Leerheitsproblem

SATZ: Es sei G eine kontextfreie Grammatik über Σ in CNF. Dann entscheidet der COCKE-YOUNGER-KASAMI-Algorithmus (CYK) mit Zeitaufwand $\mathcal{O}(|w|^3)$, ob $w \in L(G)$ ist.

Beweis: Es seien $G = (V, \Sigma, P, S)$ und $w = a_1, \ldots, a_n \in \Sigma^+$ gegeben ($w = \varepsilon$ ist trivial wegen ε-Sonderbedingung). **Idee:** bestimme für $1 \leq i \leq j \leq n$ die Menge $V_{ij} = \{A \in V \mid A \vdash_G^* a_i a_{i+1} \ldots a_j\}$. Dann ist $w \in L(G)$ genau dann, wenn $S \in V_{1n}$. Die Berechnung der V_{ij} erfolgt nach wachsenden Wortlängen $j - i + 1$:

- Für $j - 1 + 1 = 1$, also $i = j$, ist $A \in V_{ii}$ genau dann, wenn $A \to a_i \in P$ ist.
- Es gilt $A \in V_{ij}$ genau dann, wenn eine Regel $A \to BC$ in P existiert und k (mit $i \leq k < j$) existiert, so daß $B \in V_{ik}$ und $C \in V_{k+1\,j}$ ist.

CYK-Algorithmus: Eingabe $w = a_1 \ldots a_n \in \Sigma^+$, $G = (V, \Sigma, P, S)$ kontextfrei

```
for i = 1 to n
        V_ii = { A | A → a_i ∈ P };
for d = 2 to n
        for i = 1 to n + 1 − d
        {
                j = i + d − 1;
                V_ij = ∅;
                for k = i to j − 1
                        V_ij = V_ij ∪ { A | ∃A → BC ∈ P, B ∈ V_ik, C ∈ V_{k+i j} };
        }
```

Beispiel: Sei eine Grammatik G gegeben mit

$$\begin{array}{llll} G & S & \to & AB \\ & A & \to & ab|aAb \\ & B & \to & c|cB \end{array}$$

Dann ist $L(G) = \{a^n b^n c^m \mid n, m \geq 1\}$. Die Regeln in Chomsky-Normalform:

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow X_a X_b | X_a D \\ D &\rightarrow A X_b \\ B &\rightarrow c | X_c B \\ X_a &\rightarrow a \\ X_b &\rightarrow b \\ X_c &\rightarrow c \end{aligned}$$

Der Algorithmus bestimmt für $w = aaabbbcc$ Mengen V_{ij} mit:

	1	2	3	4	5	6	7	8
1	$\{X_a\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{A\}$	$\{S\}$	$\{S\}$
2		$\{X_a\}$	$\emptyset$	$\emptyset$	$\{A\}$	$\{D\}$	$\emptyset$	$\emptyset$
3			$\{X_a\}$	$\{A\}$	$\{D\}$	$\emptyset$	$\emptyset$	$\emptyset$
4				$\{X_b\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
5					$\{X_b\}$	$\emptyset$	$\emptyset$	$\emptyset$
6						$\{X_b\}$	$\emptyset$	$\emptyset$
7							$\{B, X_c\}$	$\{B\}$
8								$\{B, X_c\}$

SATZ: Es gibt einen Algorithmus, der zu gegebener kontextfreier Grammatik G entscheidet, ob $L(G) = \emptyset$ ist.

Beweis: Berechne $T = \{A \in V \mid \exists w \in \Sigma^* : A \vdash^* w\}$ der terminierenden Nichtterminale. Dann gilt: $L(G) \neq \emptyset \Leftrightarrow S \in T$. Folgender Algorithmus berechnet T:

- markiere (durch Unterstreichen) auf rechten Regelseiten alle Terminalsymbole $a \in \Sigma$
- solange eine Regel mit unmarkierter linker Seite A und vollständig markierter rechter Seite existiert, markiere A in allen Regeln (links und rechts)

Dann ist T die Menge der markierten Variablen.

Beispiele:

1. Sei folgende Grammatik gegeben:

$$\begin{aligned} S &\to \underline{b}A|\underline{a}B \\ A &\to \underline{b}AA|\underline{a}S|\underline{a} \\ B &\to \underline{a}BB|\underline{b}S|\underline{b} \end{aligned}$$

$$\begin{aligned} \text{wird zu} \quad S &\to \underline{bA}|\underline{aB} \\ \underline{A} &\to \underline{bAA}|\underline{a}S|\underline{a} \\ \underline{B} &\to \underline{aBB}|\underline{b}S|\underline{b} \end{aligned}$$

$$\begin{aligned} \text{wird zu} \quad \underline{S} &\to \underline{bA}|\underline{aB} \\ \underline{A} &\to \underline{bAA}|\underline{aS}|\underline{a} \\ \underline{B} &\to \underline{aBB}|\underline{bS}|\underline{b} \end{aligned}$$

   Damit ist $T = \{S, A, B\}$, also $L(G) \neq \emptyset$.

2. Sei folgende Grammatik gegeben:

$$\begin{aligned} S &\to AB|AB\underline{a} \\ A &\to \underline{a} \end{aligned}$$

$$\begin{aligned} \text{wird zu} \quad S &\to \underline{A}B|\underline{A}B\underline{a} \\ \underline{A} &\to \underline{a} \end{aligned}$$

   Somit ist $T = \{A\}$, also $L(G) = \emptyset$.

Korrektheit:

1. Der Algorithmus terminiert, da V endlich ist und jeder Durchlauf mindestens eine Variable markiert.

2. Es werden genau die terminierenden Variablen markiert:

   (a) Es werden nur terminierende Variablen markiert: Für alle markierten A gilt: Es existiert $w \in \Sigma^*$ mit $A \vdash^* w$ (Induktion über die Anzahl der Schleifendurchläufe)

   (b) Alle terminierenden Variablen werden auch markiert: Für alle Variablen A gilt: Falls $A \vdash^* w \in \Sigma^*$, so wird A markiert.

Zeitaufwand: abhängig von der Eingabegröße der Grammatik (Gesamtlänge aller Wörter in allen Regeln mit zusätzlichem Zählen von „$\to$“, „$|$“, „,“). Bei Gesamtlänge N genügt der Zeitaufwand $\mathcal{O}(n^2)$.

DEFINITION: Ist t ein Baum, $x \in \mathrm{dom}(t)$, so ist der durch x gegebene *Unterbaum* $t \restriction x$ von t festgelegt durch: $\mathrm{dom}(t \restriction x) = \{y \mid xy \in \mathrm{dom}(t)\}$ und für die Markierung gilt $(t \restriction x)(y) = t(xy)$.

2.4 Pumping-Lemma

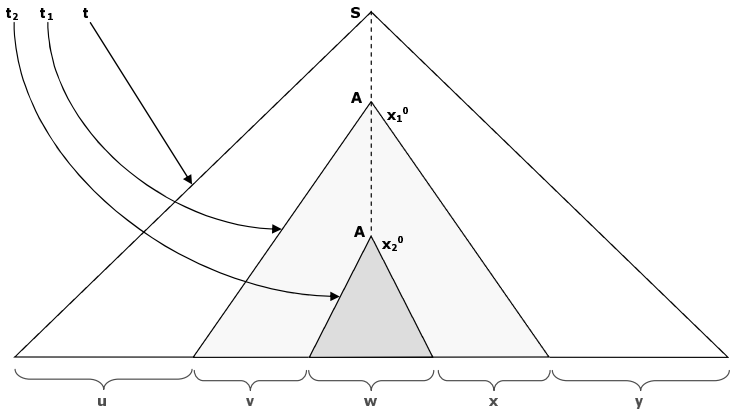
SATZ: (*Pumping-Lemma* oder *uvwxy-Theorem*) Sei $G = (V, \Sigma, P, S)$ eine kontextfreie Grammatik ohne einfache Regeln, mit k Variablen und rechten Regelseiten der Länge höchstens $l \geq 2$. Dann gilt für $n = l^{k+1}$: Für alle $z \in L(G)$ mit $|z| \geq n$ existieren u, v, w, x, y mit $z = uvwxy$ und

1. $vx \neq \varepsilon$
2. $|vwx| \leq n$
3. $uv^i wx^i y \in L(G)$ für alle $i \geq 0$

Bemerkung: Sei t ein höchstens-l-verzweigter Baum mit $|\mathrm{Yield}(t)| > l^k$, dann existiert ein Pfad der Länge mindestens k durch t. Anders formuliert: Haben alle Pfade durch t eine Länge von höchstens k, dann muß $|\mathrm{Yield}(t)| \leq l^k$. Beweise dies durch Induktion über k:

- Induktionsanfang für $k = 1$: klar, da es nur höchstens l Verzweigungen gibt
- Induktionsschritt Für $k + 1$: Alle Pfade in den Unterbäumen haben Länge höchstens k,

Beweis: Sei $z \in L(G)$ mit $|z| \geq n = l^{k+1} > l^k$ und t ein Ableitungsbaum zu z mit $z = \mathrm{Yield}(t)$. Aus der Bemerkung folgt, daß ein Pfad der Länge größer als k in t existiert. Dann gibt es verschiedene Knoten $x_1, x_2 \in t$ mit $t(x_1) = t(x_2) = A$ (Variablenwiederholung). Betrachte auf jedem Pfad von der Front aus die erste Variablenwiederholung (bei Knoten x_1, x_2). Wähle unter diesen Knotenpaaren ein Paar x_1^0, x_2^0 mit $\mathrm{Hhe}\, t \restriction x_1^0$ (für den oberen Knoten) minimal. Setze $t_1 = t \restriction x_1^0$, $t_2 = t \restriction x_2^0$. Dann ist $\mathrm{Hhe}(t_1) \leq k + 1$. Der Ableitungsbaum sieht damit aus wie folgt:



Dabei ist $w = \text{Yield}(t_2)$, $vwx = \text{Yield}(t)$ und $uvwxy = \text{Yield}(t)$. Dann folgt $A \vdash^* w$, $A \vdash_G^* vAx$ und $S \vdash_G^* uAy$. Dann gilt:

1. $vx \neq \varepsilon$, Annahme: $vx = \varepsilon$, dann ist $A \vdash_G^* A$ eine einfache Regel!
2. $|vwx| \leq n$: Da die Höhe $t \upharpoonright x_1^0$ minimal ist, haben alle Pfade in $t \upharpoonright x_1^0$ eine Länge von höchstens $k+1$. Damit ist $|vwx| = |\text{Yield}(t_1)| \leq l^{k+1} = n$.
3. $uv^iwx^iy \in L(G)$ ist klar wegen

$$S \vdash^* uAy \vdash^* uv^iAx^iy \vdash^* uv^iwx^iy$$

Beispiel:

1. Die Sprache $L_1 = \{a^mb^mc^m \mid m \geq 1\}$ ist nicht kontextfrei. *Annahme:* L_1 sei kontextfrei. Dann existiert eine kontextfreie Grammatik mit k Variablen und rechten Regelseiten der Länge höchstens l (mit $l \geq 2$). Setze nun $n = l^{k+1}$. Jedes Wort z mit $|z| \geq n$ läßt sich zerlegen in $uvwxy$ mit den obigen Eigenschaften. Betrachte $z = a^nb^nc^n$ mit $|z| = 3n > n$. Wegen $|vwx| \leq n$ kann vx nicht aus a, b und c bestehen. Fallunterscheidung:
   (a) $vwx = a^ib^j$ mit $i + j \leq n$
   (b) $vwx = b^ic^j$ mit $i + j \leq n$

   Wegen $vx \neq \varepsilon$ und $uv^0wx^0y = uwy \in L(G)$ erhalten wir in den beiden Fällen:

   (a) $vwx = a^ib^j$ - wegen $x \neq \varepsilon$ gilt $|uwy|_a < n$ oder $|uwy|_b < n$, andererseits ist $|uwy|_c = n$, also $uwy \notin L(G)$.
   (b) $vwx = b^ic^j$ - analog bleibt $|\cdot|_a = n$, aber $|\cdot|_b < n$ oder $|\cdot|_c < n$.

SATZ:

1. Die kontextfreien Sprachen sind abgeschlossen unter $\cup, \cdot$ und *.
2. Die kontextfreien Sprachen sind nicht abgeschlossen unter $\cap$ und Komplementbildung.

Beweis:

1. Seien $G_1 = (V_1, \Sigma, P_1, S_1)$ und $G_2 = (V_2, \Sigma, P_2, S_2)$ kontextfreie Grammatiken mit $V_1 \cap V_2 = \emptyset$.

   „$\cup$“ Dann ist $G = (V_1 \cup V_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \to S_1 | S_2\}, S)$ mit $S \notin V_1 \cup V_2$ kontextfreie Grammatik für $L(G_1) \cup L(G_2)$.

   „$\cdot$“ Hier bilde zu G_1, G_2 mit $V_1 \cap V_2 = \emptyset$ die Grammatik: $G = (V_1 \cup V_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \to S_1 S_2\}, S)$.

   „*“ hier bilde $G = (V_1 \cup \{S\}, \Sigma, P_1 \cup \{S \to \varepsilon | S_1, S_1 \to S_1 S_1\} \setminus \{S_1 \to \varepsilon\}, S)$

(b) „$\cap$“ $L_1 = \{a^i b^j c^j | i, j > 0\}$ und $L_2 = \{a^i b^i c^j | i, j > 0\}$ sind beide kontextfrei. L_1 durch Grammatik:

$$\begin{aligned} S &\to AB \\ A &\to a | aA \\ B &\to bc | bBc \end{aligned}$$

$L_1 \cap L_2 = \{a^i b^i c^i | i > 0\}$ ist aber nicht kontextfrei.

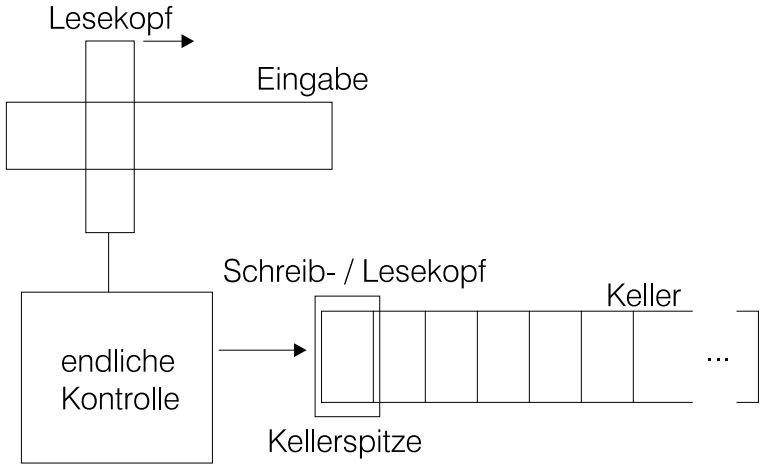
- Komplement: Kontextfreie Sprachen sind auch nicht unter dem Komplement abgeschlossen (ansonsten wäre auch der Durchschnitt abgeschlossen) wegen $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$

2.5 Kellerautomaten

DEFINITION: Ein Kellerautomat (Pushdown-Automat, PDA) hat die Form $\mathfrak{A} = (Q, \Sigma, \Gamma, q_0, z_0, \Delta, F)$

- endliche Zustandsmenge Q
- endliches Eingabealphabet Σ
- endliches Kelleralphabet Γ
- Anfangszustand $q_0 \in Q$
- Keller-Start-Symbol $z_0 \in \Gamma$
- endliche Transitionsmenge $\Delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \times \Gamma^* \times Q$
- endliche Endzustandsmenge $F \subseteq Q$

Dabei erlaubt $(q, a, z, \gamma, q') \in \Delta$ den Übergang von Zustand q nach q' beim Lesen von a von der Eingabe und Austausch von z durch γ auf der Kellerspitze. Für $\gamma = \varepsilon$ ist dieses ein *Pop-Schritt*, für $\gamma \neq \varepsilon$ ein *Push-Schritt*.



DEFINITION: Eine *Konfiguration* von $\mathfrak{A}$ hat die Form $K = (q, w, \gamma) \in Q \times \Sigma^* \times \Gamma^*$ mit momentanem Zustand q, Restwort zu Lesen w und Kellerinhalt γ.

Schreib- und Sprechweisen:

- $(q, aw, Z\gamma) \vdash_{\mathfrak{A}} (q', w, \beta\gamma)$ falls $(q, a, Z, \beta, q') \in \Delta$
- $(q, w, Z\gamma) \vdash_{\mathfrak{A}} (q', w, \beta\gamma)$ falls $(q, \varepsilon, Z, \beta, q') \in \Delta$

- $K \vdash^*_{\mathfrak{A}} K' \Leftrightarrow \exists n \geq 0 \ : \ K_0, \ldots, K_n$ mit $K = K_0 \vdash_{\mathfrak{A}} K_1 \vdash_{\mathfrak{A}} \ldots \vdash_{\mathfrak{A}} K_n = K'$
- $\mathfrak{A}$ *akzeptiert* $w \in \Sigma^*$, falls $(q_0, w, Z_0) \vdash^*_{\mathfrak{A}} (q, \varepsilon, \gamma)$ für ein $q \in F$ und beliebige $\gamma \in \Gamma^*$
- $\mathfrak{A}$ *akzeptiert w mit leerem Keller*, falls $(q_0, w, Z_0) \vdash^*_{\mathfrak{A}} (q, \varepsilon, \varepsilon)$ für beliebige $q \in Q$
- $L(\mathfrak{A}) = \{ w \in \Sigma^* \mid \mathfrak{A} \text{ akzeptiert } w \}$.
- $N(\mathfrak{A}) = \{ w \in \Sigma^* \mid \mathfrak{A} \text{ akzeptiert } w \text{ mit leerem Keller} \}$

DEFINITION: Ein Keller-Automat $\mathfrak{A} = (Q, \Sigma, \ldots)$ heißt *deterministisch* (DPDA), falls

1. $\forall q \in Q, z \in \Gamma, a \in \Sigma : (q, \varepsilon, z, \ldots) \in \Delta \Rightarrow (q, a, z, \ldots) \notin \Delta$
2. $\forall q \in Q, z \in \Gamma, a \in \Sigma \cup \{\varepsilon\}$ existiert höchstens eine Regel $(q, a, z, \ldots) \in \Delta$

Beispiele:

- Ein PDA für $L_1 = \{ a^i b^i \mid i > 0 \}$ ist gegeben durch: $Q = \{q_0, q_1, q_F\}$, $F = \{q_F\}$, $\Sigma = \{a, b\}$, $\Gamma = \{Z_0, Z\}$ und

$$\begin{aligned} \Delta \ = \ & \{(q_0, a, Z_0, ZZ_0, q_0), (q_0, a, Z, ZZ, q_0), (q_0, b, Z, \varepsilon, q_1), \\ & (q_1, b, Z, \varepsilon, q_1), (q_1, \varepsilon, Z_0, Z_0, q_F)\} \end{aligned}$$

  Mögliche Konfigurationsfolge:

$$\begin{aligned} (q_0, aabb, Z_0) \ &\vdash \ (q_0, abb, ZZ_0) \vdash (a_0, bb, ZZZ_0) \vdash (q_1, b, ZZ_0) \\ &\vdash \ (q_1, \varepsilon, Z_0) \vdash (q_F, \varepsilon, Z_0) \end{aligned}$$

  Ergänzt man den obigen PDA durch $\Delta' := \Delta \cup \{(q_F, \varepsilon, Z_0, \varepsilon, q_F)\}$, so gilt für den entstehenden PDA $\mathfrak{A}'$: $L(\mathfrak{A}') = N(\mathfrak{A}') = \{ a^i b^i \mid i > 0 \}$.

- Sei $\Sigma = \{a, b, \$\}$. Ein PDA für $L_2 = \left\{ w\$w^T \;\middle|\; w \in \{a, b\}^* \right\}$ (also Palindrome mit Trennsymbol) ist $\mathfrak{A} = (\{q_0, q_1\}, \{a, b, \$\}, \{\#, A, B\}, q_0, \#, \Delta, \emptyset)$ mit

$$\begin{aligned} \Delta \ = \ & \{(q_0, a, \#, A\#, q_0), (q_0, a, A, AA, q_0), (q_0, a, B, AB, q_0), (q_0, b, \#, B\#, q_0), \\ & (q_0, b, A, BA, q_0), (q_0, b, B, BB, q_0), (q_0, \$, \#, \#, q_1), (q_0, \$, A, A, q_1), \\ & (q_0, \$, B, B, q_1), (q_1, a, A, \varepsilon, q_1), (q_1, b, B, \varepsilon, q_1), (q_1, \varepsilon, \#, \varepsilon, q_1)\} \end{aligned}$$

  Es sind beispielsweise folgende Wörter in $N(\mathfrak{A})$:

– $\$ \in N(\mathfrak{A})$ wegen

$$(q_0, \$, \#) \vdash (q_1, \varepsilon, \#) \vdash (q_1, \varepsilon, \varepsilon)$$

– $ba\$ab \in N(\mathfrak{A})$ wegen

$$\begin{aligned}(q_0, ba\$ab, \#) &\vdash (q_0, a\$ab, B\#) \vdash (q_0, \$ab, AB\#) \vdash (q_1, ab, AB\#)\\ &\vdash (q_1, b, B\#) \vdash (q_1, \varepsilon, \#) \vdash (q_1, \varepsilon, \varepsilon)\end{aligned}$$

Beachte: der obige PDA ist deterministisch! *Frage* ist aber: Wie erkennt man $L_3 = \left\{ ww^T \;\middle|\; w \in \{a, b\}^* \right\}$ (also ohne Trennsymbol)? Ersetze dazu $(q_0, \$, \#, \#, q_1), (q_0, \$, A, A, q_1), (q_0, \$, B, B, q_1)$ durch

$$(q_0, a, A, \varepsilon, q_1), (q_0, b, B, \varepsilon, q_1), (q_0, \varepsilon, \#, \varepsilon, q_1)$$

Damit hat $\mathfrak{A}'$ hat dann mehrere Übergänge und ist somit nicht deterministisch! Nun läßt sich $w = abba$ ableiten durch:

$$(q_0, abba, \#) \vdash \begin{cases} (q_1, abba, \varepsilon) \\ (q_0, bba, A\#) \vdash (q_0, ba, BA\#) \vdash \begin{cases} (q_1, a, A\#) \vdash (q_1, \varepsilon, \#) \vdash (q_1, \varepsilon, \varepsilon) \\ (q_0, a, BBA\#) \vdash (q_0, \varepsilon, ABBA\#) \end{cases} \end{cases}$$

LEMMA:

1. Zu jedem PDA $\mathfrak{A}$ kann man einen PDA $\mathfrak{B}$ angeben mit $L(\mathfrak{A}) = N(\mathfrak{B})$
2. Zu jedem PDA $\mathfrak{A}$ kann man einen PDA $\mathfrak{B}$ angeben mit $N(\mathfrak{A}) = L(\mathfrak{B})$

Beweis:

1. $\mathfrak{B}$ soll $\mathfrak{A}$ simulieren und den Keller lesen, wenn $\mathfrak{A}$ einen Endzustand erreicht. Leert aber $\mathfrak{A}$ den Keller ohne zu akzeptieren, dann darf L aber nicht akzeptieren (Idee: verwende neues Kellersymbol X_0).

   formal: Sei $\mathfrak{A} = (Q, \Sigma, \Gamma, q_0, Z_0, \Delta, F)$. Baue L so:

   - q_0' neuer Anfangszustand
   - q_l neuer Zustand (zum Leeren des Kellers)
   - X_0 neues Kellerstartsymbol

   füge zu Δ hinzu: $(q_0', \varepsilon, X_0, Z_0X_0, q_0), (q, \varepsilon, Z, \varepsilon, q_l)$ für $q \in F, Z \in \Gamma \cup \{X_0\}$ ergibt $\Delta'(q_l, \varepsilon, Z, \varepsilon, q_l)$

   $L = (Q \cup \{q_0', q_l\}, \Sigma, \Gamma \cup \{X_0\}, q_0', X_0, \Delta', \emptyset) \Rightarrow L(\mathfrak{A}) = N(L)$

2. analog

2.5.1 Übergang von der kontextfreien Grammatik zum PDA

DEFINITION: Gegeben sei eine kontextfreie Grammatik G. Eine Ableitung $\alpha_0 \vdash_G \alpha_1 \vdash_G \cdots \vdash_G \alpha_n$ heißt *Linksableitung*, falls beim Übergang von α_i zu α_{i+1} die am weitesten links stehende Variable ersetzt wird.

LEMMA: Gegeben sei eine kontextfreie Grammatik G. Gilt $S \vdash_G^* u$, dann existiert auch eine Linksableitung $S \vdash \cdots \vdash u$.

SATZ: Zu jeder kontextfreien Grammatik G kann man einen PDA $\mathfrak{A}$ angeben mit $L(G) = N(\mathfrak{A})$.

Beweis: Betrachte zunächst den Fall $\varepsilon \notin L(G)$. Es sei $G = (V, \Sigma, P, S)$ in Greibach-Normalform[4]. *Idee:* Für z.B. eine Linksableitung $S \vdash aXY \vdash abX_1X_2Y \vdash abcX_2Y \vdash abcaY \vdash abcaa$ im Kellerautomaten:

$$\begin{aligned}(q, abcaa, S) &\vdash_{\mathfrak{A}} (q, bcaa, XY)\\ &\vdash_{\mathfrak{A}} (q, caa, X_1X_2Y)\\ &\vdash_{\mathfrak{A}} (q, aa, X_2Y)\\ &\vdash_{\mathfrak{A}} (q, a, Y)\\ &\vdash_{\mathfrak{A}} (q, \varepsilon, \varepsilon)\end{aligned}$$

DEFINITION: Sei $\mathfrak{A} = (\{q\}, \Sigma, V, q, S, \Delta, \emptyset)$ mit Δ definiert durch

$$(q, a, Z, \gamma, q) \in \Delta :\Leftrightarrow Z \to a\gamma$$

Behauptung: $L(G) = N(\mathfrak{A})$. *Zeige:* $S \vdash_G^* u\alpha$ (mit $u \in \Sigma^*$ und $\alpha \in V^*$) per Linksableitung genau dann, wenn $(q, u, S) \vdash_{\mathfrak{A}}^* (q, \varepsilon, \alpha)$. Die Behauptung folgt mit $\alpha = \varepsilon$.

„$\Leftarrow$“ Induktion über die Anzahl l der Schritte der PDA-Berechnung:

- Für $l = 0$ ist $u = \varepsilon$, $S = \alpha$, also $S \vdash_G^* S$.
- Für $l + 1$ sei folgende $\mathfrak{A}$-Berechnung der Länge $l + 1$:

$$(q, ua, S) \underbrace{\vdash^l}_{(a)} (q, a, \beta) \underbrace{\vdash}_{(b)} (q, \varepsilon, \alpha)$$

  (a) Es folgt $(q, u, S) \vdash^l (q, \varepsilon, \beta)$, mit Induktionsvoraussetzung folgt $S \vdash_G^* u\beta$.

[4] nur Regeln $A \to aB_1 \dots B_n$ mit $n \geq 0$

(*b*) Es folgt $(q, a, A, X_1 \cdots X_m, q) \in \Delta$, wobei $\beta = A\gamma$ und $\alpha = X_1 \cdots X_m\gamma$. Dann folgt $A \to aX_1 \cdots X_m \in P$, es folgt weiter $S \vdash_G^* u\beta = uA\gamma \vdash uaX_1 \cdots X_m\gamma = ua\alpha$

„⇒“ Induktion über Ableitungslänge l:

- Für $l = 0$ ist $u = \varepsilon$, $S = \alpha$.
- Es gelte $S \vdash_G^l uA\gamma \vdash uaX_1 \cdots X_m\gamma$ (mit $ua \in \Sigma^*$).
- Für $l+1$: Zeige $(q, u_a, S) \vdash_{\mathfrak{A}}^* (q, \varepsilon, X_1 \cdots X_m\gamma)$. Es gilt nach Induktionsvoraussetzung

$$S \vdash_G^l uA\gamma \Rightarrow (q, u, S) \vdash_{\mathfrak{A}}^* (q, \varepsilon, A\gamma)$$

Zudem ist $A \to aX_1 \cdots X_m \in P \Rightarrow (q, a, AX_1 \cdots X_m, q) \in \Delta$. Damit folgt $(q, ua, S) \vdash_{\mathfrak{A}}^* (q, a, A\gamma) \vdash (q, \varepsilon, X_1 \cdots X_m\gamma)$.

Für $\varepsilon \in L(G)$ füge noch $(q, \varepsilon, S, \varepsilon, q)$ zu Δ hinzu.

Folgerung: Jede kontextfreie Sprache wird durch einen PDA (mit leerem Keller) mit nur einem Zustand erkannt.

2.5.2 Übergang von einem PDA zur kontextfreien Grammatik

Satz: (ε-Bedingung für kontextfreie Grammatiken) Zu jeder Grammatik G mit Regeln $A \to \alpha$ (mit $\alpha \in (V \cup \Sigma)^*$) kann man eine äquivalente Grammatik $\bar{G}$ finden, die die ε-Bedingung erfüllt.

Beweis: Zwei Schritte: ε aus $L(G)$ eliminieren und gegebenenfalls ε hinzunehmen. Bilde $G' = (V', \Sigma, P', S')$ gemäß folgendem ε-Lemma. Falls $\varepsilon \notin L(G)$, so sind wir fertig, andernfalls bilde

$$\bar{G} = (V' \cup \{\bar{S}\}, \Sigma \cup \varepsilon, P' \cup \{\bar{S} \to \varepsilon, \bar{S} \to S'\}, \bar{S})$$

Damit erfüllt $\bar{G}$ die ε-Bedingung und $L(\bar{G}) = L(G)$.

Lemma: (ε-Lemma) Ist $G = (V, \Sigma, P, S)$ eine Grammatik mit nur Regeln $A \to \alpha$, so kann man eine Grammatik G' ohne Regeln $\alpha \to \varepsilon$ finden mit $L(G') = L(G) \setminus \{\varepsilon\}$.

Beweis: Sei $G = (V, \Sigma, P, S)$ gegeben. Definiere $V_\varepsilon = \{A \in V \mid A \vdash_G^* \varepsilon\}$. Beachte: V_ε ist algorithmisch berechenbar mit $V_\varepsilon = \bigcup_{i \geq 1} \{A \mid A \vdash^{\leq i} \varepsilon\}$ bzw. V_ε ist Vereinigung folgender Mengen V_i:

$$\begin{aligned} V_1 &= \{A \mid A \to \varepsilon \in P\} \\ V_{i+1} &= V_i \cup \{A \mid A \to \alpha \in P \text{ mit } \alpha \in (V_i)^*\} \end{aligned}$$

Dann folgt $V_1 \subseteq V_2 \subseteq V_3 \subseteq \ldots \subseteq V_l$ und diese Kette wird stationär: $V_l = V_{l+1}$ (wegen V endlich).

Setze nun

$$P' = \{A \to \alpha' \mid \alpha' \neq \varepsilon,\ \exists\, A \to \alpha \in P \text{ und } (\star)\}$$

mit $(\star)$ als Bedingung, daß α' aus α durch Streichen beliebig vieler Variablen aus V_ε entsteht. Zum Beispiel ist bei $A \to aX_1X_2Z$ mit $X_1, Z \in V_\varepsilon$ die Menge P' gegeben als $A \to aX_2Z|aX_1X_2|aX_2|aX_1X_2Z$ und $G' = (V, \Sigma, P', S)$. *Zeige:* $L(G') = L(G) \setminus \{\varepsilon\}$. *Beweisidee:*

„$\subseteq$“ überführen einer G'-Ableitung in eine G-Ableitung

„$\supseteq$“ Induktion über Länge der G-Ableitung $S \vdash_G^* w \neq \varepsilon \Rightarrow S \vdash_{G'}^* w$.

SATZ: Zu jedem PDA $\mathfrak{A}$ kann man eine kontextfreie Grammatik G angeben mit $L(G) = N(\mathfrak{A})$.

zunächst **Beweisidee:** Sei $\mathfrak{A} = (Q, \Sigma, \Gamma, q_0, Z_0, \Delta, \emptyset)$ gegeben. Finde G, so daß die Linksableitungen von G, die $\mathfrak{A}$-Berechnungen kodieren, mit $N(\mathfrak{A}) = L(G)$. Wir benutzen als G-Variablen $[pZq]$ mit $p, q \in Q$ und $Z \in \Gamma$ mit

$$[pZq] \vdash^* u \iff \mathfrak{A} : p \to^u q \text{ wobei } Z \text{ vom Keller genommen wird}$$

formal: Für $G = (V, \Sigma \cup \{\varepsilon\}, P, S)$ mit $V = \{s\} \cup \{[pZq] \mid p, q \in Q, Z \in \Gamma\}$ definiere P als

$$\begin{aligned} S &\to [q_0 Z_0 q] \ \forall\, q \in Q \\ [pZq] &\to a[p_0X_1p_1][p_1X_2p_2]\cdots[p_{m-1}X_mp_m] \\ &\qquad \text{für } p_1, \ldots, p_{m-1}, q \in Q \\ &\qquad \text{falls } (p, a, Z, X_1 \cdots X_m, p_0) \in \Delta \end{aligned}$$

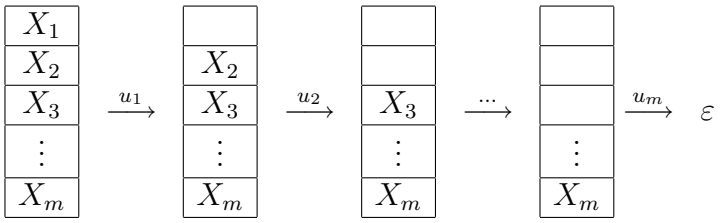
Zeige nun: $(p, u, Z) \vdash_{\mathfrak{A}}^+ (q, \varepsilon, \varepsilon) \Leftrightarrow [pZq] \vdash_G^* u$, daraus folgt:

$$\begin{aligned} u \in L(G) &\Leftrightarrow S \vdash_G [q_0 Z_0 q] \vdash_G^+ u \\ &\Leftrightarrow (q_0, u, Z_0) \vdash_{\mathfrak{A}}^+ (q, \varepsilon, \varepsilon) \\ &\Leftrightarrow u \in N(\mathfrak{A}) \end{aligned}$$

Wir zeigen also $(p, u, Z) \vdash_{\mathfrak{A}}^+ (q, \varepsilon, \varepsilon) \Leftrightarrow [pZq] \vdash_G^* u$:

„⇒“ Induktion über die Schrittzahl l von $\mathfrak{A}$:

- Für $l = 1$ ist $u = a \in \Sigma \cup \{\varepsilon\}$, $(p, a, Z, \varepsilon, q) \in \Delta$, also gilt nach Definition von G: $[pZq] \to a$.
- Es gelte $(p, au, Z) \vdash_{\mathfrak{A}} (p_0, u, X_1 \cdots X_m) \vdash^l_{\mathfrak{A}} (q, \varepsilon, \varepsilon)$.
- Zeige $[pZq] \vdash^+_G au$. Was passiert im Keller ausgehend von $(p_0, u, X_1 \cdots X_m)$?



Bestimme $u_1, \ldots, u_m$ mit $u = u_1 \cdots u_m$ gemäß dem Abbau von $X_1, \ldots, X_m$ im Keller und daraus die Zustände $p_1, \ldots, p_m = q$:

$$(p_{i-1}, u_i, X_i) \vdash^{\leq l}_{\mathfrak{A}} (p_i, \varepsilon, \varepsilon)$$

Mit der Induktionsvoraussetzung gilt $[p_{i-1}X_ip_i] \vdash^+_G u_i$. Wegen des ersten Schrittes gilt

$$(p, a, Z, X_1 \cdots X_m, p_0) \in \Delta \Rightarrow [pZq] \vdash a[p_0X_1p_1][p_1X_2p_2] \cdots [p_{m-1}X_mp_m]$$

Nun gilt insgesamt:

$$\begin{array}{rcl} [pZq] & \vdash & a[p_0X_1p_1][p_1X_2p_2] \cdots [p_{m-1}X_mq] \\ & \vdash^* & au_1u_2 \cdots u_m = au \end{array}$$

„⇐“ Induktion über die Anzahl l der Ableitungsschritte:

- Für $l = 1$: Ein G-Schritt erfolgt mit Regel $[pZq] \to a \in \Sigma \cup \{\varepsilon\}$. Dann folgt $(p, a, Z, \varepsilon, q) \in \Delta$, hieraus folgt weiter $(p, a, Z) \vdash_{\mathfrak{A}} (q, \varepsilon, \varepsilon)$.
- Für $l + 1$ betrachte $[pZq] \vdash^{l+1}_G au$. Im ersten Schritt haben wir $[pZq] \vdash_G a[p_0X_1p_1] \cdots [p_{m-1}X_mq] \vdash^l_G au$. Dann gilt: $[p_{i-1}X_ip_i] \vdash^{\leq l}_G u_i$ mit $u = u_1 \cdots u_m$ und $p_m = q$. Die Induktionsvoraussetzung liefert: $(p_{i-1}, u_i, X_i) \vdash^*_{\mathfrak{A}} (p_i, \varepsilon, \varepsilon)$. Der erste Schritt ergibt: $(p, q, Z, X_1 \cdot X_m, p_0)$, d.h. es ist

$$\begin{array}{rcl} (p, au, Z) \vdash_{\mathfrak{A}} (p_0, u, X_1 \cdots X_m) & \vdash^*_{\mathfrak{A}} & (p_1, u_2 \cdots u_m, X_2 \cdots X_m) \\ & \vdash^*_{\mathfrak{A}} & (p_2, u_3 \cdots u_m, X_3 \cdots X_m) \\ & \vdash^*_{\mathfrak{A}} & (p_{m-1}, u_m, X_m) \vdash^*_{\mathfrak{A}} (q, \varepsilon, \varepsilon) \end{array}$$

Beispiel:

- Sei folgender PDA gegeben:

$$\mathfrak{A} = (\{q_0, q_1\}, \{0, 1\}, \{X, Z_0\}, q_0, Z_0, \Delta, \emptyset)$$

mit folgenden Transitionen (mit Nummern versehen)

$$\begin{aligned}\Delta &= \{(q_0, 0, Z_0, XZ_0, q_0)_{(1)}, (q_0, 0, X, XX, q_0)_{(2)}, (q_0, 1, X, \varepsilon, q_1)_{(3)} \\ &\quad (q_1, 1, X, \varepsilon, q_1)_{(4)}, (q_1, \varepsilon, Z_0, \varepsilon, q_1)_{(5)}\}\end{aligned}$$

Die Sprache, die $\mathfrak{A}$ erkennt, ist $\{0^n 1^n \mid \mathbb{N}_{>0}\}$. Die zugehörige Grammatik $G = (V, \Sigma, P, S)$ ergibt sich mit $\Sigma = \{0, 1\}$,

$$\begin{aligned}V &= \{S, [q_0Xq_0], [q_0Xq_1], [q_1Xq_0], [q_1Xq_1], \\ &= \quad [q_0Z_0q_0], [q_0Z_0q_1], [q_1Z_0q_0], [q_1Z_0q_1]\}\end{aligned}$$

und folgenden Regeln (in Klammern jeweils die Nummer der Transition, aus der sie entstehen):

$$\begin{array}{rcll}
S & \to & [q_0Z_0q_0]|[q_0Z_0q_1] & \\
[q_0Z_0q_0] & \to & 0[q_0Xq_0][q_0Z_0q_0]|0[q_0Xq_1][q_1Z_0q_0] & (1) \\
[q_0Z_0q_1] & \to & 0[q_0Xq_0][q_0Z_0q_1]|0[q_0Xq_1][q_1Z_0q_1] & (1) \\
[q_0Xq_0] & \to & 0[q_0Xq_0][q_0Xq_0]|0[q_0Xq_1][q_1Xq_0] & (2) \\
[q_0Xq_1] & \to & 0[q_0Xq_0][q_0Xq_1]|0[q_0Xq_1][q_1Xq_1]|1 & (2), (3) \\
[q_1Z_0q_0] & & \text{keine Regel} & \\
[q_1Z_0q_1] & \to & \varepsilon & (5) \\
[q_1Xq_0] & & \text{keine Regel} & \\
[q_1Xq_1] & \to & 1 & (4)
\end{array}$$

Wir können $[q_0Z_0q_0]$ weglassen, da die Regeln für die Variable nichts eliminieren bzw. zu Variablen ohne weitere Regeln führen. Ebenso $[q_0Xq_0]$ und Regeln, die zu diesen Variablen führen - es bleibt folgendes Regelwerk:

$$\begin{array}{rcl}
S & \to & [q_0Z_0q_1] \\
[q_0Z_0q_1] & \to & 0[q_0Xq_1][q_1Z_0q_1] \\
[q_0Xq_1] & \to & 0[q_0Xq_1][q_1Xq_1]|1 \\
[q_1Z_0q_1] & \to & \varepsilon \\
[q_1Xq_1] & \to & 1
\end{array}$$

Nun kann man beispielsweise ableiten:

$$S \vdash [q_0Z_0q_1] \vdash 0[q_0Xq_1][q_1Z_0q_1] \vdash \begin{cases} 0[q_0Xq_1] \vdash 01 \\ 00[q_0Xq_1][q_1Xq_1] \vdash \begin{cases} 0011 \\ \dots \end{cases} \end{cases}$$

2.6 Deterministische kontextfreie Sprachen

SATZ: Wird L durch einen DPDA erkannt und ist R regulär, so wird auch $L \cap R$ durch einen DPDA erkannt.

Beweisidee: Konstruktion über par. Automaten (siehe Übung).

Sprechweise: L heißt *deterministisch kontextfrei*.

Wir wissen: $L_1 = \{a^i b^i \mid i \geq 0\}$ und $L_2 = \left\{ w\$w^T \;\middle|\; w \in \Sigma^* \right\}$ sind deterministisch kontextfrei, $L_3 = \left\{ ww^T \;\middle|\; w \in \Sigma^* \right\}$ ist kontextfrei.

DEFINITION: $w \in \Sigma^*$ heißt *minimales L-Wort* genau gann, wenn $w \in L$ ist und kein echtes Präfix von w in L ist.

Operationen:

1. $\mathrm{MIN}(L) := \{w \in \Sigma^* \mid w \text{ mininales } L\text{-Wort}\}$
2. Komplement $\bar{L} = \Sigma^* \setminus L$

Beispiele:

- Falls $\varepsilon \in L$, so ist $\mathrm{MIN}(L) = \{\varepsilon\}$.
- Für $L = 1^*2$ ist $\mathrm{MIN}(L) = L$.

LEMMA: Ist L deterministisch kontextfrei, so auch $\mathrm{MIN}(L)$.

Beweis: in der Übung.

SATZ: Die Sprache $L = \left\{ ww^T \;\middle|\; w \in \{a,b\}^* \right\}$ ist nicht deterministisch kontextfrei.

Beweis: (ausführlich in der Übung) *Angenommen,* L wäre deterministisch kontextfrei. Die Sprache $R = (ab)^+(ba)^+(ab)^+(ba)^+$ ist regulär. Da L deterministisch kontextfrei ist, ist auch $\mathrm{MIN}(L)$ deterministisch kontextfrei und damit auch $\mathrm{MIN}(L) \cap R$. Zeige nun, daß

$$L_0 := \mathrm{MIN}(L) \cap R = \left\{ (ab)^i (ba)^j (ab)^j (ba)^i \;\middle|\; 0 < j < i \right\}$$

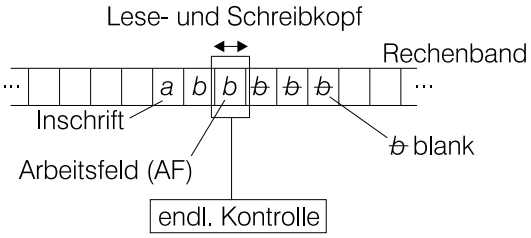
Wende das Pumping-Lemma für kontextfreie Sprachen auf L_0 an, damit ist $\mathrm{MIN}(L) \cap R$ nicht kontextfrei, Widerspruch!

SATZ:

1. Die deterministisch kontextfreien Sprachen sind abgeschlossen unter Komplement.
2. Die deterministisch kontextfreien Sprachen sind nicht abgeschlossen unter Durchschnitt und Vereinigung.

3 Berechenbarkeit, Entscheidbarkeit

Betrachte zunächst *Turingmaschinen*:



DEFINITION: Eine *Turingmaschine (DTM)* hat die Form $\mathfrak{A} = (Q, \Sigma, \Gamma, q_0, \delta, F)$ mit

- endlicher Zustandsmenge Q,
- Eingabealphabet Σ,
- Arbeitsalphabet $\Gamma \supseteq \Sigma$ (mit $\text{ƀ} \in \Gamma \setminus \Sigma$)
- Anfangszustand $q_0 \in Q$
- Endzustände $F \subseteq Q$
- (partielle) Übergangsfunktion $\delta\colon (Q \setminus F) \times \Gamma \to \Gamma \times \{l, r\} \times Q$, wobei $\delta(q, a) = (a', l/r, q')$ bedeutet, daß im Zustand q mit Lesen von a auf dem Arbeitsfeld ein a' auf das Arbeitsfeld gedruckt wird, der Kopf um eine Einheit nach l oder r bewegt wird und der neue Zustand q' ist.

DEFINITION: Eine *Konfiguration* einer Turingmaschine ist ein Wort aus $\Gamma^* Q \Gamma^*$, wobei uqv die Bandinschrift

$$\ldots \text{ƀ}\ \text{ƀ}\ \text{ƀ}\ u\ v\ \text{ƀ}\ \text{ƀ}\ \text{ƀ} \ldots$$

kodiert, q der aktuelle Zustand ist und das Arbeitsfeld auf dem ersten Buchstaben von v steht.

Für $n = n_0 b$ und $v = a v_0$ sei die *Folgekonfiguration* von nqv definiert durch

$$\begin{array}{ll} n_0 q' b a' v_0 & \text{falls} \quad \delta(q, a) = (a', l, q') \\ n_0 b a' q' v_0 & \text{falls} \quad \delta(q, a) = (a', r, q') \end{array}$$

Für $n = \varepsilon$ bzw. $v = \varepsilon$ gilt dies mit $n_0 = \varepsilon$, $b = \text{ƀ}$ bzw. $v_0 = \varepsilon$ und $a = \text{ƀ}$. K heißt *Stoppkonfiguration*, falls keine Folgekonfiguration von K existiert.

Notation:[5]

- $K \vdash_{\mathfrak{A}} K'$ genau dann, wenn K' Folgekonfiguration von K ist
- $K \vdash^*_{\mathfrak{A}} K'$ genau dann, wenn $n \in \mathbb{N}$, $K = K_0, K_1, \ldots, K_n = K'$ exisiteren mit $K_i \vdash_{\mathfrak{A}} K_{i+1}$

Beispiel: Sei $\mathfrak{A} = (\{q_0, q_1, q_2\}, \{|\}, \{|, \text{ƀ}\}, q_0, \delta, \{q_2\})$ mit folgendem δ:

$$\begin{aligned} \delta(q_0, |) &= (|, r, q_0) \\ \delta(q_0, \text{ƀ}) &= (|, l, q_1) \\ \delta(q_1, |) &= (|, l, q_1) \\ \delta(q_1, \text{ƀ}) &= (\text{ƀ}, r, q_2) \end{aligned}$$

Eine mögliche Konfigurationsfolge von $\mathfrak{A}$ ist z.B.

$$\begin{aligned} \text{ƀ}q_0|||\text{ƀƀ} &\vdash \text{ƀ}|q_0||\text{ƀƀ} \vdash^2 \text{ƀ}|||q_0\text{ƀ} \\ &\vdash \text{ƀ}||q_1||\text{ƀƀ} \vdash^3 q_1\text{ƀ}||||\text{ƀ} \\ &\vdash \text{ƀ}q_2||||\text{ƀ} \end{aligned}$$

Allgemein $q_o|^n \vdash^*_{\mathfrak{A}} q_2|^{n+1}$.

DEFINITION: Die von einer DTM $\mathfrak{A}$ akzeptierte Sprache ist

$$L(\mathfrak{A}) = \{ w \in \Sigma^* \mid q_0 w \vdash^*_{\mathfrak{A}} \alpha q \beta \text{ für } q \in F \text{ und } \alpha, \beta \in \Gamma^* \}$$

Beachte, daß $\alpha q \beta$ eine Stoppkonfiguration ist.

DEFINITION: Eine *nichtdeterministische Turingmaschine (NTM)* hat die Form $\mathfrak{A} = (Q, \Sigma, \Gamma, q_0, \delta, F)$, wobei alles wie bei DTM definiert ist, nur die Übergangsfunktion ist definiert als $\delta \colon (Q \setminus F) \times \Gamma \to \mathcal{P}(\Gamma \times \{l, r\} \times Q)$. Eine NTM akzektiert ihre Eingabe, wenn es eine Folge von Konfigurationen gibt, die in einen Endzustand läuft.

3.1 Varianten von Turingmaschinen

Möglich Varianten sind z.B.:

1. andere Definition der Übergangsfunktion und andere Akzeptanzbedingungen bei „Stoppkonfiguration“, „akzeptierte Stoppkonfiguration“ (mit erreichtem Zustand $q \in F$) oder akzeptierte Konfiguration

[5] „Klaus ist die nächsten zwei Wochen nicht da, das hat er wahrscheinlich nicht gesagt...“

2. ein nach links begrenztes Band

3. mehrere[6] Bänder, mehrere Schreib- und Leseköpfe

4. der Kopf kann auch stehenbleiben

3.1.1 Mehrband-Turingmaschine

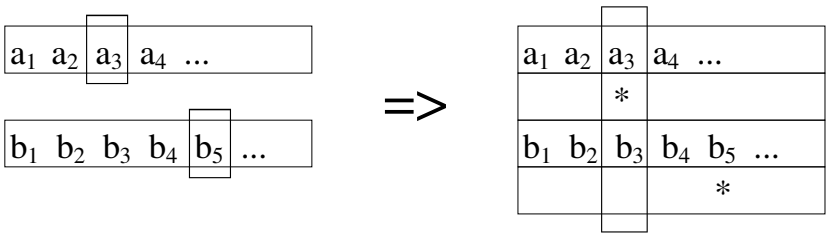
DEFINITION: Eine k-Band NTM (mit k Bändern und einem Kopf pro Band) hat die Form $\mathfrak{A} = (Q, \Sigma, \Gamma, q_0, \Delta, F)$ mit

$$\Delta \subseteq Q \times \Gamma^k \times \Gamma^k \times \{l, n, r\}^k \times Q$$

Eine Transition $q(a_1, \ldots, a_k)(a'_1, \ldots, a'_k)(d_1, \ldots, d_k)q' \in \Delta$ bedeutet Im Zustand q und a_i auf dem Arbeitsfeld vom Band i für alle $i = 1, \ldots, k$ drucke jeweils a'_i, bewege den Kopf nach d_i (wobei n „stehenbleiben“ bedeutet) und gehe in den Zustand q' über. Anfangskonfiguration ist $q_0 w$ mit w auf dem ersten Band und alle anderen enthalten nur Blanks.

SATZ: Jede k-Band NTM kann durch eine 1-Band NTM simuliert werden.

Beweisidee Sei $\mathfrak{A} = (Q, \Sigma, \Gamma, q_0, \Delta, F)$ eine k-Band NTM. Gesucht ist eine 1-Band NTM $\mathfrak{A}'$ und Kodierung von $\mathfrak{A}$-Konfigurationen so, daß zu jeder $\mathfrak{A}$-Transition eine Folge von $\mathfrak{A}'$-Transitionen existiert. $\mathfrak{A}'$ hat das Arbeitsalphabet $\Gamma' = (\Gamma \cup \{\star\})^{2k}$ (das Band von $\mathfrak{A}'$ hat also $2k$ Spuren). Die Spuren $2i - 1$ seien für die Inschriften auf den Bändern i zuständig, die Spuren $2i$ für die Markierung der Kopfstellung von $\mathfrak{A}$ auf den i-ten Bändern (dort steht dann ein $\star$):



Der Kopf von $\mathfrak{A}'$ steht auf dem am weitesten links befindlichen $\star$-Feld. Wie simuliert man nun die $\mathfrak{A}$-Transitionen?

1. gehe von links nach rechts bis zum letzten $\star$-Feld

   (a) zähle die Anzahl der gefundenen $\star$

[6] „Ich kann das nicht schreiben... das sieht scheiße aus so!“

(b) speicher die $a_1, \ldots, a_k$ auf den Spuren $1, 3, 5, \ldots, 2k-1$

2. laufe von rechts nach links bis zum ersten $\star$-Feld
   (a) verwende den Spurzählen
   (b) drucke a_i' auf die $(2i-1)$-te Spur
   (c) verschiebe $\star$ nach d_i auf der $2i$-ten Spur
3. gehe in den Zustand q' über

Da sich alles auch auf deterministisch realisieren läßt, gilt der folgende Satz:

SATZ: Jede k-Band DTM kann durch eine 1-Band DTM simuliert werden.

SATZ: Wenn L von einer NTM erkannt wird, dann auch von einer DTM.

Beweisidee: Sei $L = L(\mathfrak{A})$ für eine NTM $\mathfrak{A}$. Gesucht ist eine DTM $\mathfrak{B}$ mit $L = L(\mathfrak{B})$. $\mathfrak{B}$ hat alle möglichen Konfigurationsfolgen von $\mathfrak{A}$ zu erzeugen und bei Auffinden einer akzeptierten Konfiguration mit $q \in F$ zu akzeptieren. Sei $r = \max\{|\delta(q,a)| \mid (q,a) \in (Q \setminus F) \times \Gamma)\}$ (d.h. r beschreibt die maximale Anzahl von Transitionen $(q, a, \cdot, \cdot, \cdot)$). Für jedes Paar (q, a) beschreibt $i \in \{1, \ldots, r\}$ die Auswahl der Transition. Also sind alle Konfigurationsfolgen $K_0 \vdash_{\mathfrak{A}} \ldots \vdash_{\mathfrak{A}} K_n$ mit $K_0 = q_0 w$ eindeutig repräsentierbar durch Wörter $\{1, \ldots, r\}^*$ der Länge n.

Wir konstruieren nun eine DTM $\mathfrak{B}$ mit drei Bändern, wobei die Bänder zuständig seien für:

1. Band für die Eingabe
2. Band für das Erzeugen aller Wörter $\{1, \ldots, r\}^*$
3. Band für die Simulation einer $\mathfrak{A}$-Berechnung zu einem festen Wort auf dem zweiten Band

Wenn wir in einen Endzustand laufen, so haben wir eine akzeptierende Konfigurationsfolge (kodiert auf dem zweiten Band) gefunden.

3.1.2 Linear beschränkte Automaten

DEFINITION: Ein *linear beschränkter Automat (LBA)* ist von der Form $\mathfrak{A} = (Q, \Sigma, \Gamma, \delta, q_0, ¢, \$, F)$, wobei $Q, \Sigma, \Gamma, q_0, F$ wie für NTM definiert sind, $¢$ und \$ spezielle Symbole aus $\Gamma \setminus \Sigma$ sind (die linke und rechte Endmarkierung) und der Automat keine Bewegungen links von $¢$ und rechts von \$ macht noch schreibt er ein anderes Symbol über $¢$und \$.

Weiter ist folgendes die von $\mathfrak{A}$ akzeptierte Sprache:

$$L(\mathfrak{A}) = \{ w \in \Sigma^* \mid q_0 ¢ w \$ \vdash_{\mathfrak{A}}^* \alpha q \beta \text{ mit } q \in F, \alpha, \beta \in \Gamma^* \}$$

SATZ:

1. Die von allgemeinen Turingmaschinen akzeptierten Sprachen sind genau die Typ-0-Sprachen.
2. Die von LBAs akzeptierten Sprachen sind genau die kontextsensitiven Sprachen.

Beweis:

1. Wir zeigen: NTM-Sprachen = Typ-0-Sprachen

   „$\subseteq$“ Sei $\mathfrak{A}$ eine DTM mit $\mathfrak{A} = (Q, \Sigma, \Gamma, q_0, \delta, F)$. Sei $G = (V, \Sigma, P, S)$ mit

   $$V = \{S, A, B, E\} \cup Q \cup ((\Sigma \cup \{ƀ\}) \times \Gamma)$$

   Die Ableitung soll in drei Phasen geschehen (hier eine Beispiel-Ableitung: zunächst eine „Kopie“ des Wortes erstellen, dann für jede $\mathfrak{A}$-Konfiguration $b_1 \cdots b_{n+1} q b_{n+2} \cdots b_{n+l}$ eine entsprechende Ableitung erzeugen und zuletzt ein reines Wort ableiten, falls man eine akzeptierte Konfiguration mit $q \in F$ erreicht hat):

   $$\begin{aligned} S &\vdash^* [ƀ, ƀ]^k q_0 [a_1, a_1] \cdots [a_n, a_n] [ƀ, ƀ]^l && (1) \\ &\vdash^* [ƀ, ƀ]^k [a_1, b_1] \cdots [a_n, b_n] [ƀ, b_{n+1}] q [ƀ, b_{n+2}] \cdots [ƀ, b_{n+l}] && (2) \\ &\vdash^* a_1, \ldots, a_n && (3) \end{aligned}$$

   Regeln für die einzelnen Phasen:

   (1) Regeln für das Erzeugen eines Wortes und seiner Kopie:

   $$\begin{aligned} S &\to B q_0 A \\ A &\to [a, a] A | B \quad \forall\, a \in \Sigma \\ B &\to [ƀ, ƀ] B | \varepsilon \end{aligned}$$

(2) Für $a_1, a_2 \in \Sigma \cup \{\not b\}$ und $b_1, b_2 \in \Gamma$:

$$\begin{array}{rcll} p[a_1, b_1] & \to & [a_1, b_1']q & \text{für } \delta(p, b_1) = (b_1', r, q) \\ [a_1, b_1]\, p[a_2, b_2] & \to & q[a_1, b_1][a_2, b_2'] & \text{für } \delta(p, b_2) = (b_2', l, q) \end{array}$$

(3) Für $(q, b) \in F \times \Gamma$ verwende Regeln

$$\begin{array}{rcll} q[a, b] & \to & EaE & \forall\, a \in \Sigma \\ q[\not b, b] & \to & E & \end{array}$$

Für $a \in \Sigma$ und $b \in \Gamma$ erstelle Regeln

$$\begin{array}{rcl} E[a, b] & \to & aE \\ [a, b]\, E & \to & Ea \\ E[\not b, b] & \to & E \\ \left[\not b, b\right] E & \to & E \\ E & \to & \varepsilon \end{array}$$

„$\supseteq$“ Sei nun G eine Grammatik vom Typ 0, wir konstruieren eine 2-Band NTM $\mathfrak{A}$ so, daß $L(G) = L(\mathfrak{A})$ ist.

(a) Schreibe S (erste Satzform) auf das zweite Band. Simulation eines Ableitungsschritees (zweiter Kopf auf dem Anfang der Satzform γ)

(b) Wähle nichtdeterministisch eine Position i in γ ($i \leq |\gamma|$)

(c) Wähle nichtdeterministisch eine Regel $\alpha \to \beta$

(d) Falls α ein Teilwort von γ beginnend bei der Position i ist, ersetze α durch β (mit Verschiebung des Wortes rechts von i um $|\beta| - |\alpha|$ nach rechts (bzw. links falls negativ))

(e) Ist die neue Satzform $\gamma' = w$, dann gehe in eine akzeptierte Stoppkonfiguration. Falls nicht, gehe zum Schritt (b).[7]

[7] „Ich lass' Euch in Ruhe abschreiben, dabei wisch' ich schon mal!“

3.2 Berechenbarkeit

DEFINITION: Sei eine (partielle) Funktion $f\colon (\Sigma^*)^n \dashrightarrow \Sigma^*$ und eine DTM $\mathfrak{A}$ gegeben. Wir sagen, $\mathfrak{A}$ *berechnet* f, falls gilt:

- für alle $(w_1, \ldots, w_n) \in \mathrm{Def}(f)$ gilt:

$$q_0 w_1 ß w_2 ß \ldots ß w_n \vdash_{\mathfrak{A}}^* q f(w_1, \ldots, w_n)$$

  mit $q \in F$

- für alle $(w_1, \ldots, w_n) \notin \mathrm{Def}(f)$ wird von $q_0 w_1 ß w_2 ß \ldots ß w_n$ aus keine Stoppkonfiguration erreicht.

f heißt *Turing-berechenbar*, falls eine Turingmaschine existiert, die f berechnet.

Bemerkungen:

- Eine Turingmaschine $\mathfrak{A}$ mit m Bändern schreibt die Ausgabe auf das erste Band.
- Setze zur Vereinfachung voraus, daß die Turingmaschine auch stehen bleiben darf.

LEMMA: Jede sequentielle Funktion $f : \Sigma^* \to \Sigma^*$ ist Turing-berechenbar.

Beweis: Sei eine sequentielle Maschine $\mathfrak{A} = (Q, \Sigma, \Sigma, q_0, \delta, \lambda)$ gegeben mit $\delta\colon Q \times \Sigma \to Q$, $\lambda\colon Q \times \Sigma \to \Sigma$. Definiert waren

$$\begin{array}{lll} \delta^*\colon Q \times \Sigma^* \to Q & \text{mit} & \delta^*(q, \varepsilon) = q \text{ und} \delta^*(q, wa) = \delta(\delta^*(q, w), q) \\ \lambda^*\colon Q \times \Sigma^* \to \Sigma^* & \text{mit} & \lambda^*(q, \varepsilon) = \varepsilon \text{ und} \lambda^*(q, wa) = \lambda^*(q, w)\lambda(\delta^*(q, w), q) \end{array}$$

Sei also $f = f_{\mathfrak{A}}\colon \Sigma^* \to \Sigma^*$ mit $w \mapsto \gamma^*(q_0, w)$. Die Turingmaschine $\mathfrak{B}$ arbeitet wie $\mathfrak{A}$, überdruckt die Eingabebuchstaben durch Ausgabebuchstaben und geht an den Anfang des Ausgabewortes zurück.

$$\begin{array}{rcll} \delta_{\mathfrak{B}}(q, a) & = & (a', r, q') & \forall\, \delta(q, a) = q' \text{ und } \lambda(q, a) = a' \\ \delta_{\mathfrak{B}}(q, ß) & = & (ß, l, q_l) & \forall\, q \in Q \\ \delta_{\mathfrak{B}}(q_l, a) & = & (a, l, q_l) & \forall\, a \in \Sigma \\ \delta_{\mathfrak{B}}(q_l, ß) & = & (ß, r, q_s) & \end{array}$$

mit $Q_{\mathfrak{B}} = Q \cup \{q_l, q_s\}$ und $F = \{q_s\}$.

Beispiel: Die Funktion

$$f\colon \{|\}^* \to \{|\}^* \text{ mit } |^n \mapsto \begin{cases} | & \text{falls} \quad n \text{ ist Primzahl} \\ \varepsilon & \text{sonst} \end{cases}$$

Als Übung, verwende dazu eine 3-Band-Turingmaschine, das erste Band enthält $|^n$ (die Eingabe), das zweite Band enthält einen Zähler $|^i$, der von $|^2$ bis $|^{n-1}$ läuft, und das dritte Band testet, ob n durch i teilbar ist.

DEFINITION: Eine Funktion $f\colon (\Sigma^*)^n \to \Sigma^*$ heißt *im intuitiven Sinn berechenbar*, wenn es einen Algorithmus (entweder umgangssprachlich oder in Programmiersprache formuliert) gibt, der f berechnet, d.h. für alle $(w_1, \ldots, w_n) \in \mathrm{Def}(f)$ jeweils $f(w_1, \ldots, w_n)$ liefert und sonst nicht terminiert.

Eine bekannte Vermutung von 1936 ist die sogenannte **Churchsche These**: Eine Funktion f ist genau dann in intuitivem Sinne berechenbar, wenn sie Turing-berechenbar ist. Wenn wir $f\colon \mathbb{N}^k \to \mathbb{N}$ durch eine Funktion

$$f'\colon (\{|\}^*)^k \to \{|\}^* \text{ mit } f'(|^{i_1}, \ldots, |^{i_k}) = |^{f(i_1,\ldots,i_k)}$$

identifizierenm so können wir auch partielle Funktionen $f\colon \mathbb{N}^k \to \mathbb{N}$ auf Berechenbarkeit untersuchen.

4 Die Programmiersprache WHILE

4.1 WHILE, LOOP, und GOTO-Programme

4.1.1 WHILE und LOOP

Verwende im Folgenden

$$n \dot{-} m = \max\{n-m, 0\} = \begin{cases} n-m & \text{falls} \quad n-m \geq 0 \\ 0 & \text{sonst} \end{cases}$$

Sei $\Sigma = \{a, \ldots, z, X, 0, \ldots, 9, +, \dot{-}, :, =, ;, >\}$. Dann kann ein WHILE-*Programm* sein:

$$\begin{array}{rcl}
\langle \text{Programm} \rangle & ::== & \langle \text{Wertzuweisung} \rangle \mid \\
 & & \langle \text{Programm} \rangle \, ; \langle \text{Programm} \rangle \mid \\
 & & \texttt{loop} \ \langle \text{Variable} \rangle \ \texttt{begin} \ \langle \text{Programm} \rangle \ \texttt{end} \mid \\
(\star) & & \texttt{while} \ \langle \text{Variable} \rangle > 0 \ \texttt{do} \ \texttt{begin} \ \langle \text{Programm} \rangle \ \texttt{end} \\
\langle \text{Wertzuweisung} \rangle & ::== & \langle \text{Variable} \rangle := \langle \text{Variable} \rangle + 1 \mid \\
 & & \langle \text{Variable} \rangle := \langle \text{Variable} \rangle \ \dot{-} \ 1 \mid \\
 & & \langle \text{Variable} \rangle := 0 \mid \\
 & & \langle \text{Variable} \rangle := \langle \text{Variable} \rangle + \langle \text{Variable} \rangle \mid \\
 & & \langle \text{Variable} \rangle := \langle \text{Variable} \rangle \ \dot{-} \ \langle \text{Variable} \rangle \mid \\
 & & \langle \text{Variable} \rangle := \langle \text{Variable} \rangle \\
\langle \text{Variable} \rangle & ::== & X \, \langle \text{positiveZahl} \rangle \\
\langle \text{positiveZahl} \rangle & ::== & \langle \text{positiveZiffer} \rangle \mid \langle \text{positiveZahl} \rangle \, \langle \text{Ziffer} \rangle \\
\langle \text{positiveZiffer} \rangle & ::== & 1 \mid 2 \mid 3 \mid \ldots \mid 9 \\
\langle \text{Ziffer} \rangle & ::== & 0 \mid \langle \text{positiveZiffer} \rangle
\end{array}$$

Haben wir ein WHILE-Programm ohne Anwendung der Regel ($\star$), so nennen wir es auch LOOP-Programm.

4.1.2 Semantik

Ein Programm transformiert einen Vektor $(n_1, n_2, \ldots)$ von Werten der Variablen $X_1, X_2, \ldots$ in einen neuen Vektor oder liefert bei Nicht-Termination keinen Vektor zurück. Der Zustandsraum ist also die Menge $\mathbb{N}^{\mathbb{N}_+}$ der Folgen $(n_1, n_2, \ldots)$ in $\mathbb{N}$.

Vorbereitung: Für $(n_1, n_2, \ldots)$ sei $\mathrm{pr}_i(n_1, n_2, \ldots) = n_i$ die i-te Projektion. Für $f \colon A \dashrightarrow A$ sei $f^n \colon A \dashrightarrow A$ definiert durch $f^0 = \mathrm{id}_A$, $f^1 = f$,

und $f^{n+1}(a) = f(f^n(a))$. Man beachte: Ist $f^n(a) = \perp$ (undefiniert), so ist $f^{n+i}(a) = \perp$ für alle $i \in \mathbb{N}$.[8]

DEFINITION: Zu jedem $P \in$ `WHILE` wird die *Semantische Funktion* $[P] \colon \mathbb{N}^{\mathbb{N}_+} \dashrightarrow \mathbb{N}^{\mathbb{N}_+}$ (induktiv über den Aufbau von P) wie folgt definiert:

1. Ist P eine Wertzuweisung, so möge gelten:

P	$[P](n_1, n_2, \ldots)$
$Xi = Xj + 1$	$(n_1, \ldots, n_{i-1}, n_j + 1, n_{i+1}, \ldots)$
$Xi = Xj \dot{-} 1$	$(n_1, \ldots, n_{i-1}, n_j \dot{-} 1, n_{i+1}, \ldots)$
$Xi = 0$	$(n_1, \ldots, n_{i-1}, 0, n_{i+1}, \ldots)$

2. Ist $P = P_1; P_2$, so sei $[P] = [P_2] \circ [P_1]$.
3. Ist $P =$ `loop` Xi `begin` P_0 `end`, so sei $[P](n_1, n_2, \ldots) = [P_0]^{n_i}(n_1, n_2, \ldots)$.
4. Ist $P =$ `while` $Xi > 0$ `do begin` P_1 `end`, so sei

$$[P](n_1, n_2, \ldots) = \begin{cases} [P_1]^n & \text{für } n \text{ minimal mit } \mathrm{pr}_i([P_1]^n(n_1, n_2, \ldots)) = 0 \\ \perp & \text{falls kein solches } n \text{ existiert} \end{cases}$$

Das Programm arbeitet wie folgt:

- Zunächst legt man die Eingabe (ein n-Tupel[9]) in die Variablen $X_1, \ldots, X_n$,
- dann läuft das Programm (bzw. die semantische Funktion $[P]$),
- die Ausgabe ist der Wert von X_1.

DEFINITION: Für jedes `WHILE`-Programm P und $k \geq 0$ wird die durch P berechnete k-stellige Funktion $f_P^{(k)} \colon \mathbb{N}^k \dashrightarrow \mathbb{N}$ mit

$$(n_1, \ldots, n_k) \mapsto (\mathrm{pr}_1 \circ [P] \circ \mathrm{in}^{(k)})(n_1, \ldots, n_k)$$

mit $\mathrm{in}^{(k)} \colon \mathbb{N}^k \to \mathbb{N}^{\mathbb{N}_+}$ wobei $(n_1, \ldots, n_k) \mapsto (n_1, \ldots, n_k, 0, \ldots)$.

DEFINITION: Eine Funktion $f : \mathbb{N}^k \dashrightarrow \mathbb{N}$ ist `LOOP`- (bzw. `WHILE`-)berechenbar, falls ein `LOOP`- (bzw. `WHILE`-)Programm P existiert mit $f = f_P^{(k)}$.

[8] „Das ist so scheiße, das diese Tafel immer so schnell voll ist!“

[9] „Tubel“

Bemerkung: Man beachte, daß sich Konstruktionen wie `if ...then ...else` simulieren lassen: Das Programm `if` $X_i > 0$ `then begin` P_1 `end else begin` P_2 `end` läßt sich simulieren als:

$Y_1 := 1;$
$Y_2 := 1;$
$Y_2 := Y_2 \dot{-} X_i;$
$Y_1 := Y_1 \dot{-} Y_2;$
`loop` Y_1 `begin` P_1 `end`;
`loop` Y_2 `begin` P_2 `end`;

LEMMA: Jede LOOP-berechenbare Funktion ist total (d.h. nicht partiell).

Beweis: Folgt unmittelbar aus den Definitionen, da LOOP-Programme stets terminieren.

Folgerung: Es gibt WHILE-berechenbare Funktionen, die nicht LOOP-berechenbar sind.

Beweis: Wir müssen nur dank des obigen Lemmas eine partielle WHILE-berechenbare Funktion angeben, etwa: Sei P

`while` $X_1 > 0$ `do begin` $X_1 := X_1 + 1$ `end`

LEMMA: Ist P ein LOOP-Programm mit m Variablen $X_1, \ldots, X_m$, dann existiert ein WHILE-Programm P ohne `loop` mit $m' \geq m$ Variablen und

$$[P](n_1, \ldots, n_m, 0, \ldots) = [P'](n_1, \ldots, n_m, 0, \ldots)$$

Beweis: Per Induktion über den Aufbau der LOOP-Programme als Übungsaufgabe.

Folgerung: Jede LOOP-berechenbare Funktion ist durch ein WHILE-Programm ohne `loop` berechenbar.

4.1.3 GOTO

DEFINITION: GOTO-*Programme (URM-Programme)* haben die Form P : $1\ \alpha_1; 2\ \alpha_2; \ldots; m\ \alpha_m$ mit $\alpha_m = \texttt{stop}$ und

$$\begin{aligned} \alpha_i \quad ::== \quad & X_j := X_j + 1 \\ \vee \quad & X_j := X_j - 1 \\ \vee \quad & \texttt{if } X_j = 0 \texttt{ goto } l \text{ mit } l \in \{1, \ldots, m\} \end{aligned}$$

GOTO-Programme können semantisch durch Konfigurationen beschrieben werden. Eine GOTO-Konfiguration hat die Form $(l, n_1, n_2, \ldots)$ mit $1 \leq l \leq m$. Die Folgekonfiguration (für $l < m$) ist

$$\begin{aligned} (l+1, n_1, \ldots, n_{j-1}, n_j \pm 1, n_{j+1}, \ldots) \quad & \text{für} \quad \alpha_j == X_j := X_j \pm 1 \\ (l', n_1, \ldots) \quad & \text{für} \quad \texttt{if } X_j = 0 \texttt{ goto } l' \text{ mit } n_j = 0 \\ (l+1, n_1, \ldots) \quad & \text{für} \quad \texttt{if } X_j = 0 \texttt{ goto } l' \text{ mit } n_j \neq 0 \end{aligned}$$

DEFINITION: P *berechnet* $f\colon \mathbb{N}^k \dashrightarrow \mathbb{N}$, falls

- P von der Konfiguration $(1, n_1, \ldots, n_k, 0, \ldots)$ die Stoppkonfiguration $(m, x, \ldots)$ erreicht und $x = f(n_1, \ldots, n_k)$ ist, sofern $f(n_1, \ldots, n_k)$ definiert ist
- P nicht anhält, sofern $f(n_1, \ldots, n_k)$ nicht definiert ist.

Beispiel: Sei folgendes Programm gegeben:

$$\begin{aligned} 1 \quad & \texttt{if } X_2 = 0 \texttt{ goto } 5 \\ 2 \quad & X_1 := X_1 + 1 \\ 3 \quad & X_2 := X_2 - 1 \\ 4 \quad & \texttt{if } X_3 = 0 \texttt{ goto } 1 \\ 5 \quad & \texttt{stop} \end{aligned}$$

Rechenablauf:

Zeile	X_1	X_2	X_3
1	1	2	0
2	1	2	0
3	2	2	0
4	2	1	0
1	2	1	0
2	2	1	0
3	3	1	0
4	3	0	0
1	3	0	0
5	3	0	0

Somit liefert das Programm $X_1 := X_1 + X_2$.

4.2 Äquivalenz der Berechenbarkeit

Äquivalenzbeweis der Berechenbarkeit von WHILE-Programmen und Turingmaschinen durch Ringschluß

$$\texttt{WHILE} \rightarrow_{(1)} \texttt{WHILE}_0 \rightarrow_{(2)} \texttt{GOTO} \rightarrow_{(3)} \text{TM} \rightarrow_{(4)} \texttt{WHILE}$$

Wobei $\texttt{WHILE}_0$-Programme WHILE-Programme mit Wertzuweisungen $X_j := X_j \pm 1$ ohne LOOP sind.

SATZ: (1) Jede WHILE-berechenbare Funktion ist auch $\texttt{WHILE}_0$-berechenbar.

Beweis: in der Übung.

SATZ: (2) Jede $\texttt{WHILE}_0$-berechenbare Funktion ist auch GOTO-berechenbar.

Beweis: Per Induktion über den Aufabau der $\texttt{WHILE}_0$-Programme: Zu jedem $\texttt{WHILE}_0$-Programm P (mit m Veriablen) kann man ein GOTO-Programm P' konstruieren.

- Induktionsanfang: Für $P == X_i := X_i \pm 1$ setze

$$\begin{aligned} P' \quad ::== \quad & 1\ X_i := X_i \pm 1 \\ & 2\ \texttt{stop} \end{aligned}$$

- Induktionsschritt:
  - Für $P == P_1; P_2$ wähle nach Induktionsvoraussetzung P_1' und P_2'. Die Anzahl der Zeilen von P_1' sei l. Setze $P' ::== P_1'$ ohne stop-Zeile gefolgt von P_2' mit um $(l-1)$ erhöhten Zeilennummern.
  - Für $P == \texttt{while}\ X_i > 0\ \texttt{do}\ P_1$ wähle P_1' gemäß Induktionsvoraussetzung mit l Zeilen. Setze nun (mit neuer Variable X_{m+1}, die immer 0 ist):

$$\begin{array}{rl} 1 & \texttt{if}\ X_i = 0\ \texttt{goto}\ l+2 \\ \left.\begin{array}{r} 2 \\ \vdots \\ l \end{array}\right\} & P_1'\ \text{ohne}\ \texttt{stop}\ \text{, um eins erhöhte Zeilennummern} \\ l+1 & \texttt{if}\ X_{m+1} = 0\ \texttt{goto}\ 1 \\ l+2 & \texttt{stop} \end{array}$$

SATZ: (3) Jede `GOTO`-berechenbare Funktion ist auch Turing-berechenbar.

Idee: simuliere P-Konfiguration $(j, r_1, \ldots, r_m, 0, \ldots)$ durch Turingmaschinen-Konfiguration mit m Bändern (ein Band für jede Variable) und im Wesentlichen $l + 2$ Zustände $q_0, q_1, \ldots, q_l, q_s$. Zu P konstruiere Turingmaschine $\mathfrak{A}$ mit Blöcken $B_0, \ldots, B_l$ von Transitionen:

- B_0 überführt Konfiguration $q_0|^{r_1}\text{ƀ} \ldots |^{r^m}\text{ƀ}$ der Eingabe in den Zustand q_1 und die Zeichenfolgen $|^{r_i}\text{ƀ}$ auf dem i-ten Band.
- Der Block B_j für $1 \leq j < l$ leistet das Folgende: Es simuliert die Nachfolgekonfiguration

  $$(j, r_1, \ldots, r_m, 0, \ldots) \to (j', r'_1, \ldots, r'_m, 0, \ldots)$$

  abhängig von der j-ten Zeile:

  1. j-te Zeile $==$ $X_i := X_i + 1$: Füge einen Strich auf dem i-ten Band ein und gehe in Zustand q_{j+1}
  2. j-te Zeile $==$ $X_i := X_i - 1$: Lösche (falls möglich) einen Strich auf dem i-ten Band und gehe in Zustand q_{j+1}
  3. j-te Zeile $==$ $X_i :=$ `if` $X_i = 0$ `goto` k: Teste, ob das i-te Band leer ist, wenn ja, dann gehe in Zustand q_k über, andernfalls in den Zustand q_{l+1}.
- Der Block B_l überführt wieder die Konfiguration im Zustand q_l mit $|^{r_1}, \ldots, |^{r_m}$ auf den m Bändern in den Zustand q_s, wobei $|^{r_1}$ der Wert von f ist (also auf dem ersten Band).

SATZ: (4) Jede Turing-berechenbare Funktion ist `WHILE`-Berechenbar.

Beweis: Sei $\mathfrak{A}$ eine Turingmaschine mit $Q = \{q_0, \ldots, q_m\}, \Gamma = \{a_0, a_1, \ldots, a_{n-1}\}$, $a_0 = \text{ƀ}$, $a_1 = |$, $\Sigma = \{|\}$, q_0 Startzustand, q_m Endzustand. Für alle $i \in \{0, \ldots, m-1\}$ und für alle $j \in \{0, \ldots, n-1\}$ sei o.B.d.A. $\delta(q_i, a_j)$ definiert. Eine Konfiguration $K = a_{i_k} \cdots a_{i_0} q_r a_{j_0} \cdots a_{j_l}$ wird kodiert durch $Z_K = r$, $L_K = i_0 + i_q \cdot n + \ldots + i_k \cdot n^k$ und $R_K = j_0 + j_1 \cdot n + \ldots + j_l \cdot n^l$. wobei $i_0, \ldots, i_k, j_0, \ldots, j_l \in \{0, \ldots, n-1\}$.

Bemerkung: Die $i_0, \ldots, i_k$ sind eindeutig durch L_k gegeben (n-adische Zahlendarstellung). *Beispiel:* $\Gamma = \{\text{ƀ}, l\}, n = 2, K = |||\text{ƀ}|q_{17}|\text{ƀ}||\text{ƀ}$ ergibt $Z_K = 17$, $L_K = 1 + 4 + 8 + 16 = 29$, $R_K = 1 + 4 + 8 = 13$.

Idee: Verwende ein `WHILE`-Programm mit drei Variablen X_1, X_2, X_3. Das `WHILE`-Programm besteht aus drei Blöcken:

1. Gegeben sind Eingabewerte $n_1, \ldots, n_k$ in $X_1, \ldots, X_k$. Berechne zu einer gegebenen $\mathfrak{A}$-Konfiguration $K_0 = q_0|^{n_1}\text{ƀ}|^{n_2}\text{ƀ}\cdots\text{ƀ}|^{n_k}\text{ƀ}$ die Werte $X_1 := Z_{K_0}, X_2 := L_{K_0}$ und $X_3 := R_{K_0}$ durch das folgende Programm:

$X_{k+1} := 0;$
$X_{k+2} := 0;$
loop X_1 **begin** — *Schleife läuft*
  $X_{k+2} := X_{k+2} + n^{X_{k+1}}$ — n_1 *mal*
  $X_{k+1} := X_{k+1} + 1$
end
$X_{k+1} := X_{k+1} + 1$
loop X_2 **begin** — *Schleife läuft*
  $X_{k+2} := X_{k+2} + n^{X_{k+1}}$ — n_2 *mal*
  $X_{k+1} := X_{k+1} + 1$
end
$X_{k+1} := X_{k+1} + 1$
$\vdots$
loop X_k **begin** — *Schleife läuft*
  $X_{k+2} := X_{k+2} + n^{X_{k+1}}$ — n_k *mal*
  $X_{k+1} := X_{k+1} + 1$
end
$X_1 := 0;$
$X_2 := 0;$
$X_3 := X_{k+2};$
$X_4 := 0;$
$X_5 := 0;$
$\vdots$

2. Simulation der Turingmaschinen-Schritte, *Beispiel:* Bei $\Gamma = \{a_0, \ldots, a_9\} = \{0, \ldots, 9\}$ ist $K = 1085q_{17}7702$ (mit $\delta(q_{17}, 7) = (6, l, q_{18})$ angewandt: $K' = 108q_{18}56702$), es ergeben sich:

$$R_K = 7 + 7 \cdot 10 + 0 \cdot 10^2 + 2 \cdot 10^3 = 2077$$

Methode:

$$2077 \xrightarrow{-7} 2070 \xrightarrow{+6} 2076 \xrightarrow{\cdot 10} 20760 \xrightarrow{+5} 20765$$

Allgemein passiert bei einem Linksschritt $\delta(q_i, a_j) = (a_{j'}, l, q_{i'})$ folgendes als Block $B(i, j)_l$:

$$\begin{aligned} &X_3 := X_3 - j + j';\\ &X_3 := X_3 \cdot n;\\ &X_3 := X_3 + (X_2 \bmod n)\\ &X_2 := X_2 \operatorname{div} n; X_1 := i';\end{aligned}$$

Bei einem Rechtsschritt $\delta(q_i, a_j) = (a_{j'}, r, q_{i'})$ geschieht folgendes als Block $B(i, j)_r$:

$$\begin{aligned} &X_3 := X_3 \operatorname{div} n;\\ &X_2 := X_2 \cdot n; X_2 := X_2 + j';\\ &X_1 := i';\end{aligned}$$

Insgesamt simuliert man die Auswahl dieser Blöcke $B(i,j)_{r/l}$ duch eine große `WHILE`-Schleife:

```
while X1 < m do
  begin {Liste von Anw. für jede mögl. Transition}
    {für δ(q_i, a_j) = (a_j', l, q_i')}
    if X_1 = i and (X_3  mod n = j) then
      begin B(i,j)_{l/r} end
  end
end
```

3. Dekodierung der rechten Bandinschrift (Wort in $\Sigma^* = \{|\}^*$) durch zugehörige Zahl und Speicherung in X_1

```
X_1 := 0;
loop X_3 begin
  if X_3 mod n = 1 then
    X_1 := X_1 + 1;
  X_3 := X_3 div n;
end;
```

Beispiel: Konfiguration: $q_m|||\not b \rightarrow R_K = 1 + n + n^2 =: X_3$, dann ist

$$\begin{aligned} n^2 + n + 1 \bmod n &= 1 \Rightarrow X_1 = 1 \\ n^2 + n + 1 \operatorname{div} n &= n + 1 \\ n + 1 \bmod n &= 1 \Rightarrow X_1 = 2 \\ n + 1 \operatorname{div} n &= 1 \\ 1 \bmod n &= 1 \Rightarrow X_1 = 3 \\ 1 \operatorname{div} n &= 0 \Rightarrow X_3 = 0 \end{aligned}$$

Beachte:

- Die Werte sind durch $\sum_{i=0}^{l} j_i n^i$ kodiert (wobei $j_i \in \{0,1\}$) und werden durch $\bmod n$ extrahiert.
- n ist eine Konstante, die wir z.B. durch $(n+1)$ Befehle erzeugen können:

$$\left.\begin{array}{l} X_4 := 0; \\ X_4 := X_4 + 1; \\ \ldots; \\ X_4 := X_4 + 1; \end{array}\right\} n \text{ mal}$$

SATZ: (Normalformensatz) Jede `WHILE`- oder `GOTO`-berechenbare Funktion $f\colon \mathbb{N}^k \dashrightarrow \mathbb{N}$ ist durch ein `WHILE`-Programm mit nur

1. einer `while`-Anweisung,
2. höchstens den Variablen $X_1, \ldots, X_k$ und höchstens fünf Hilfsvariablen berechenbar.

Beachte: Die `while`-Anweisung können wir nicht eliminieren.

Frage: Kann man die `while`-Anweisung bei der Berechnung von totalen Funktionen eliminieren? **Antwort:** Nein, siehe später die Ackermann-Funktion.

4.3 rekursive Funktionen

4.3.1 Grundfunktionen, Komposition

DEFINITION: *Grundfunktionen* sind

- die Nachfolgerfunktion $\mathrm{N}\colon \mathbb{N} \to \mathbb{N}$ mit $\mathrm{N}(x) = x + 1$
- die nullstellige Konstanten-Funktion $c_0^{(0)}$ mit Wert 0
- die einstellige Konstanten-Funktion $c_0^{(1)}\colon \mathbb{N} \to \mathbb{N}$ mit $c_0^{(1)}(x) = 0$
- die Projektionsfunktion $p_i^{(n)}\colon \mathbb{N}^n \to \mathbb{N}$ mit $p_i^{(n)}(x_1, \ldots, x_i, \ldots, x_n) = x_i$ für alle $i = 1, \ldots, n$

Bemerkung:

1. die Identität läßt sich darstellen als $\mathrm{id}_{\mathbb{N}} = p_1^{(1)}$
2. jede Grundfunktion ist total und `LOOP`-berechenbar

DEFINITION: Sei $g\colon \mathbb{N}^m \to \mathbb{N}$, $h_1\colon \mathbb{N}^n \dashrightarrow \mathbb{N}$, $\ldots$, $h_m\colon \mathbb{N}^n \dashrightarrow \mathbb{N}$. Die Funktion f entsteht aus $g, h_1, \ldots, h_m$ durch *Komposition*, falls

$$f(x) = g(h_1(x), \ldots, h_m(x)) \; \forall \; x \in \mathbb{N}^n$$

Beachte: Ist eine der Funktionen $h_i(x) = \perp$ oder $g(h_1(x), \ldots, h_m(x)) = \perp$, so ist auch $f(x) = \perp$.

Bemerkung:

$$\begin{aligned} c_1^{(0)} &= \mathrm{N}(c_0^{(0)}) \quad \text{Wert 1 ohne Argument} \\ c_1^{(1)} &= \mathrm{N}(c_0^{(1)}) \quad \text{Wert 1 mit einem Argument: } c_1^{(1)}(x) = 1 \end{aligned}$$

Schreibweise: $f = g \circ (h_1, \ldots, h_m)$ oder $f = \mathrm{Komp}(g, h_1, \ldots, h_m)$.

Beispiele:

$$\begin{array}{rcll} f(x) & = & x(x+1) & \Rightarrow \quad f = \mathrm{Komp}(\cdot, \mathrm{id}_{\mathbb{N}}, \mathrm{N}) \\ f(x) & = & x^3 & \Rightarrow \quad f = \mathrm{Komp}(\cdot, \mathrm{id}_{\mathbb{N}}, \mathrm{Komp}(\cdot, \mathrm{id}_N, \mathrm{id}_N)) \\ f(x,y) & = & x(x+y) & \Rightarrow \quad f = \mathrm{Komp}(\cdot, p_1^{(2)}, +) \end{array}$$

LEMMA:

1. sind $g, h_1, \ldots, h_m$ total, so auch $\mathrm{Komp}(g, h_1, \ldots, h_m)$
2. sind $g, h_1, \ldots, h_m$ `LOOP`- oder `WHILE`-berechenbar, so auch $\mathrm{Komp}(g, h_1, \ldots, h_m)$

Beweis:

2. Gegeben seien Programme $P_0, \ldots, P_m$ für $g, h_1, \ldots, h_m$. Sei k die maximale Anzahl von Variablen in $P_0, \ldots, P_m$ und $k^\star = \max\{m, n, k\}$. Das folgende Programm berechnet f:

$$\begin{array}{l} X_{k^\star+1} := X_1; \\ \ldots; \\ X_{k^\star+n} := X_n; \\ P_1; \\ X_{k^\star+n+1} := X_1; \\ X_1 := X_{k^\star+1}; \\ \ldots; \\ X_n := X_{k^\star+n}; \\ X_{n+1} := 0; \\ \ldots; \\ X_{k^\star} := 0; \\ P_2; \\ X_{k^\star+n+2} := X_1; \\ \vdots \\ P_m; \\ X_{k^\star+n+m} := X_1; \\ X_1 := X_{k^\star+n+1}; \\ \ldots; \\ X_m := X_{k^\star+n+m}; \\ X_{m+1} := 0; \\ \ldots; \\ X_{k^\star} := 0; \\ P_0; \end{array}$$

4.3.2 primitive Rekursion

DEFINITION: Sei $g\colon \mathbb{N}^n \dashrightarrow \mathbb{N}$, $h\colon \mathbb{N}^{n+2} \dashrightarrow \mathbb{N}$. Dann entsteht $f\colon \mathbb{N}^{n+1} \dashrightarrow \mathbb{N}$ aus g und h durch *primitive Rekursion*, falls

1. $f(x, 0) = g(x)$ für alle $x \in \mathbb{N}^n$
2. $f(x, y+1) = h(x, y, f(x, y))$ für alle $x \in \mathbb{N}^n, y \in \mathbb{N}$

Schreibweise: $f = \mathrm{PR}(g, h)$.

Bemerkung: Die Definition oben induziert ein induktives oder rekursives Berechnungsverfahren für $f(x, y)$: berechne nacheinander $f(x, 0)$, $f(x, 1)$, ..., $f(x, y)$.

Beispiele:

1. *Summe:* Es ist $f(x, 0) = x = g(x) = p_1^{(1)}(x)$, und $f(x, y + 1) = x + (y + 1) = (x + y) + 1 = h(x, y, x + y)$, also ist $h(x, y, t) = \mathrm{N}(t)$, also $h = \mathrm{N} \circ p_3^{(3)}$. Damit ergibt sich:

$$+ = \mathrm{PR}(p_1^{(1)}, \mathrm{N} \circ p_3^{(3)})$$

2. *Produkt:* Es ist $f(x, 0) = x \cdot 0 = 0 = c_0^{(1)}(x)$ und $f(x, y+1) = x(y+1) = xy + x = h(x, y, xy)$, also $h(x, y, t) = t + x$, d.h. $h = \mathrm{Komp}(+, p_1^{(3)}, p_3^{(3)})$. D.h. insgesamt:

$$\cdot = \mathrm{PR}(c_0^{(1)}, \mathrm{Komp}(+, p_1^{(3)}, p_3^{(3)}))$$

3. *Fakultät:* $f(0) = 0! = 1$, d.h. f ist einstellig und damit g nullstellig, also $c_1^{(0)}$. Weiter ist $f(y + 1) = y!(y + 1) = h(y, y!)$, also ist $h(y, t) = t(y + 1)$, dementsprechend $h = \mathrm{Komp}(\cdot, pr_2^{(2)}, \mathrm{N} \circ p_1^{(2)})$ also:

$$! = \mathrm{PR}(c_1^{(0)}, \mathrm{Komp}(\cdot, p_2^{(2)}, \mathrm{N} \circ p_1^{(2)}))$$

LEMMA:

1. sind g, h total, so auch $\mathrm{PR}(g, h)$
2. sind g, h `LOOP`- oder `WHILE`-berechenbar, so auch $\mathrm{PR}(g, h)$

Beweis:

2. Seien P_g und P_h `LOOP`- oder `WHILE`-Programme, die die n-stellige Funktion g und die $n + 2$-stellige Funktion h berechnen. Sei k die maximale Anzahl von Variablen in P_g, P_h und $k^\star := \max\{k, n + 2\}$. Folgendes

Programm berechnet f ($n+1$-stellig):

$X_{k^\star+1} := X_1;$
$\ldots;$
$X_{k^\star+n+1} := X_{n+1};$
$X_{n+1} := 0;$
$P_g;$
$X_{k^\star+n+2} := 0;$
`loop` $X_{k^\star+n+1}$ `begin`
  $X_{n+2} := X_1;$ //Wert von P_g, P_h
  $X_1 := X_{k^\star+1}$
  $\ldots;$
  $X_n := X_{k^\star+n};$
  $X_{n+1} := X_{k^\star+n+2};$ //Wert von y
  $X_{n+3} := 0;$
  $\ldots;$
  $X_{k^\star} := 0;$
  $P_h;$
  $X_{k^\star+n+2} := X_{k^\star+n+2} + 1$
`end`;

DEFINITION: Eine Funktion $f\colon \mathbb{N}^k \dashrightarrow \mathbb{N}$ heißt *primitiv rekursiv (p.r.)*, falls f entweder eine Grundfunktion ist oder aus diesen in endlich vielen Schritten durch Komposition und primitive Rekursion erzeugbar ist.

SATZ:

1. Jede primitiv rekursive Funktion ist total.
2. Jede primitiv rekursive Funktion ist `LOOP`-berechenbar.

Beweis: Folgt aus obigen Lemmata per Induktion über den Aufbau der Funktion.

Beispiel: Wir definieren die Funktion

$$c_k^{(n+1)}\colon \mathbb{N}^{n+1} \to \mathbb{N} \text{ mit } c_k^{(n+1)}(x_1, \ldots, x_{n+1}) = k$$

Diese wird gebildet durch

$$\begin{aligned}
c_k^{(1)}(x) &= N \circ \ldots (k \text{ mal}) \circ N \circ c_0^{(1)}(x) \\
c_k^{(n+1)}(x_1, \ldots, x_n, 0) &= c_k^n(x_1, \ldots, x_n) \\
c_k^{(n+1)}(x_1, \ldots, x_n, y+1) &= P_{n+2}^{(n+2)}(x_1, \ldots, x_n, y, c_k^{(n+1)}(x_1, \ldots, x_n, y)) \\
c_k^{(n+1)} &= \mathrm{PR}(c_k^{(n)}, P_{n+2}^{(n+2)})
\end{aligned}$$

Bemerkung: Die folgenden Funktionen sind primitiv rekursiv:

$$\begin{aligned} f(x,y) &= x^y \\ f(x,y) &= |x-y| \\ f(x,y) &= \underbrace{x^{x^{\cdots^x}}}_{y \text{ mal}} \\ f(x) &= \text{sign}(x) = \begin{cases} 0 & \text{falls} \quad x = 0 \\ 1 & \text{falls} \quad x > 0 \end{cases} \\ f(x) &= \overline{\text{sign}} = \begin{cases} 1 & \text{falls} \quad x = 0 \\ 0 & \text{falls} \quad x \neq 0 \end{cases} \end{aligned}$$

Beispiel: Die folgende Funktion ist primitiv rekursiv:

$$c\colon \mathbb{N}^2 \to \mathbb{N} \text{ mit } c(x,y) = \sum_{i=0}^{x+y} i + x = \binom{x+y+1}{2} + x$$

Beweis:

$$\begin{aligned} c(x,0) &= \binom{x+1}{2} + x = \frac{(x+1)x}{2} + x = g(x) \\ c(x,y+1) &= \binom{x+y+2}{2} + x = \binom{x+y+1}{2} + x + y + 1 + x \\ &= c(x,y) + x + y + 1 \stackrel{!}{=} h(x,y,c(x,y)) \end{aligned}$$

Damit sind $h(x,y,t) = x + y + t + 1$ und g wie oben primitiv rekursiv und es ist $c = \text{PR}(g,h)$. Folgende Werte ergeben sich:

	0	1	2	3	4
0	0	2	5	9	14
1	1	4	8	13	
2	3	7	12		
3	6	11			
4	10				

LEMMA:

1. Die Funktion $c\colon \mathbb{N}^2 \to \mathbb{N}$ ist bijektiv.
2. Die folgenden Umkehrfunktionen sind primitiv rekursiv:

$$\begin{aligned} e\colon \mathbb{N} \to \mathbb{N} \quad &\text{mit} \quad e(c(x,y)) = x \\ f\colon \mathbb{N} \to \mathbb{N} \quad &\text{mit} \quad f(c(x,y)) = y \end{aligned}$$

Beweisidee:

2. (a) Definiere $q(n) = \max\{x \leq n \mid p(x) = 0\}$. Falls das Maximum nicht existiert, definieren wir $q(n) = 0$. *Zeige:* Ist p primitiv rekursiv, so auch q.

   (b) Betrachte $b\colon \mathbb{N}^2 \to \mathbb{N}$ mit $b(x,n) = 0$ genau dann, wenn ein $y \leq n$ existiert mit $a(x,y) = n$; und $b(x,n) = 1$, falls kein solches y existiert. *Zeige:* Ist a primitiv rekursiv, so auch b.

   (c) Es ergibt sich:

$$e(n) = \max\{x \leq n \mid \ \exists\, y \leq n\colon c(x,y) = n\}$$

Bemerkung: Die Kodierung von zwei Zahlen x, y in eine Zahl n können wir verallgemeinern:

$$\langle n_1, \ldots, n_k\rangle = c(n_1, c(n_2, \ldots, c(n_k, 0)))$$

Für diese *Standard-k-Tupel-Funktion* gilt:

1. sie ist primitiv rekursiv und bijektiv
2. $\mathrm{pr}_j^{(k)} \circ \langle\cdot\rangle^{-1}$ ist primitiv rekursiv für $1 \leq j \leq k$

SATZ: Eine Funktion f ist primitiv rekursiv genau dann, wenn f LOOP-berechenbar ist.

Beweis:

„⇒“ bereits gezeigt

„⇐“ Sei $f\colon \mathbb{N}^k \to \mathbb{N}$ LOOP-berechenbar. Dann existiert ein LOOP-Programm P, das f mit $m \geq k$ Variablen berechnet.

Zeige: Es gibt eine primitiv rekursive Funktion $g_P\colon \mathbb{N} \to \mathbb{N}$ mit der folgenden Eigenschaft $(\star)$:

$$g_P(\langle a_1, \ldots, a_m\rangle) = \langle b_1, \ldots, b_m\rangle \ \text{ falls } \ [P](a_1, \ldots, a_m) = (b_1, \ldots, b_m)$$

Dann folgt:

$$f(a_1, \ldots, a_k) = \mathrm{pr}_1^{(m)} \circ \langle\cdot\rangle^{-1} \circ g_P(\langle a_1, \ldots, a_k, 0, \ldots, 0\rangle)$$

Zeige nun $(\star)$ per Induktion über den Aufbau der LOOP-Programme.

- Falls P die Gestalt $X_i := X_j \pm 1$ hat, so folgt:

$$g_P(a) = \langle p_1^{(m)} \circ \langle\cdot\rangle^{-1}(a), \ldots, \underbrace{p_j^{(m)} \circ \langle\cdot\rangle^{-1}(a) \pm 1}_{i\text{-te Stelle}}, \ldots, p_m^{(m)} \circ \langle\cdot\rangle^{-1}(a)\rangle$$

- Falls P die Gestalt $P_1; P_2$ hat, so gilt: $g_P(a) = g_{P_2}(g_{P_1}(a))$ Dies ist die Komposition von primitiv rekursiven Funktionen g_{P_1}, g_{P_2} nach Induktionsvoraussetzung.

- Ist $P ==$ `loop` X_i `do` Q `end`, so definieren wir durch primitive Rekursion eine zweistellige Funktion h:

$$\begin{aligned} h(a,0) &= a \\ h(a,n+1) &= g_Q(h(a,n)) \end{aligned}$$

  Dabei gibt $h(a,m)$ den Zustand der Programmvariablen $a = \langle a_1, \ldots, a_m\rangle$ nach n Aufrufen von Q wieder. Damit ergibt sich die folgende primitiv rekursive Funktion:

$$g_P(a) = h(a, p_i^{(m)} \circ \langle\cdot\rangle^{-1}(a))$$

4.3.3 μ-Rekursion

DEFINITION: Sei $g\colon \mathbb{N}^{n+1} \dashrightarrow \mathbb{N}$ gegeben. Die Funktion $f\colon \mathbb{N}^n \dashrightarrow \mathbb{N}$ entsteht durch *unbeschränkte Minimalisierung* (oder *Anwendung des μ-Operators*), falls:

> $f(x)$ ist das kleinste y mit $g(x,y) = 0$ und für alle $z < y$ ist $g(x,z)$ definiert und ungleich 0, falls ein solches y existiert, andernfalls $f(x) = \perp$.

Bemerkung: $f(x) = \perp$ genau dann, wenn kein y existiert mit $g(x,y) = 0$ oder vor dem ersten y mit $g(x,y) = 0$ ein $z < y$ existiert mit $g(x,z) = \perp$.

Schreibweise: $f(x) = \mu y\, g(x,y) = 0$ oder $f = \mu\text{-OP}(g)$. **Beispiele:**

1. Sei $g(x,y) = |x - y|$. Dann ergibt sich für $f(x) = \mu y\ |x - y| = 0$ die Funktion $f(x) = x$.

2. Sei $g(x,y) = x + y$. Dann ergibt sich für $f(x) = \mu y\, x + y = 0$ die Funktion

$$f(x) = \begin{cases} 0 & \text{falls} \quad x = 0 \\ \perp & \text{falls} \quad x > 0 \end{cases}$$

Bemerkung: Der μ-Operator führt aus dem Bereich der totalen Funktionen heraus.

LEMMA: Falls g `WHILE`-berechenbar ist, so auch μ-OP(g).

Beweis: Sei P_g ein `WHILE`-Programm, das $g\colon \mathbb{N}^{n+1} \dashrightarrow \mathbb{N}$ mit $m \geq n+1$ Variablen berechnet. *Idee:* Die Variablen nehmen nacheinander folgende Werte an:

X_1	...	X_n	X_{n+1}	...	X_m	X_{m+1}	...	X_{m+n}	X_{m+n+1}
a_1	...	a_n	0	...	0	a_1	...	a_n	0
$g(a,0)$	...	...	...	...	...	a_1	...	a_n	1
a_1	...	a_n	1	...	0	a_1	...	a_n	1
$g(a,1)$	...	...	...	...	...	a_1	...	a_n	2
a_1	...	a_n	2	...	0	a_1	...	a_n	2
$g(a,2)$	...	...	...	...	...	a_1	...	a_n	3

Programm für f:

$X_{m+1} := X_1;$
$\ldots;$
$X_{m+n} := X_n;$
$X_{m+n+1} := 0;$
$P_g;$
`while` $X_1 > 0$ `do begin`
  $X_{m+n-1} := X_{m+n-1} + 1;$
  $X_1 := X_{m+1};$
  $\ldots;$
  $X_n := X_{m+n};$
  $X_{n+1} := X_{m+n+1};$
  $X_{n+2} := 0$
  $\ldots;$
  $X_m := 0$
  $P_g;$
`end;`
$X_1 := X_{m+n+1}$

DEFINITION: Eine Funktion f heißt *μ-rekursiv*, falls f entweder eine Grundfunktion ist oder aus diesen in endlich vielen Schritten durch Komposition, primitive Rekursion und unbeschränkter Minimalisierung erzeugbar ist.

SATZ: Eine Funktion f ist μ-rekursiv genau dann, wenn f `WHILE`-berechenbar ist.

Beweis:

„⇒“ per Induktion über den Aufbau der μ-rekursiven Funktionen und mit Lemmata

„⇐“ Es fehlt noch: $P ==$ `while` $X_i > 0$ `do begin` Q `end`, definiere $h(a, n)$ wie vorher mit $a = \langle a_1, \ldots, a_m \rangle$. Dann setzen wir

$$g_P(a) = h(a, f(a)) \text{ mit } f(a) = \mu y \ \mathrm{pr}_i^{(m)} \circ \langle \cdot \rangle^{-1} h(a, y) = 0$$

Dabei gibt $f(a)$ gerade die minimale Wiederholungszahl des Programms Q an, so daß X_i den Wert 0 hat.

4.3.4 die Ackermann-Funktion

DEFINITION: Die Ackermann-Funktion $\alpha \colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ ist definiert durch

(1) $\alpha(0, y) = y + 1$

(2) $\alpha(x, 0) = \alpha(x - 1, 1)$ für $x > 0$

(3) $\alpha(x, y) = \alpha(x - 1, \alpha(x, y - 1))$ für $x, y > 0$

Beispiele:

- $\alpha(1, 0) = \alpha(0, 1) = 2$
- $\alpha(1, 1) = \alpha(0, \alpha(1, 0)) = \alpha(0, 2) = 3$
- $\alpha(1, 2) = \alpha(0, \alpha(1, 1)) = \alpha(0, 3) = 4$

LEMMA: Für alle x, y erhält man nach endlich vielen Anwendungen von (1), (2) und (3) eine Zahl für $\alpha(x, y)$.

Beweis: per Induktion nach x:

- Induktionsanfang 1: für $x = 0$ nach Regel (1)
- Induktionsvoraussetzung 1: für alle y sei $\alpha(n, y)$ definiert
- Induktionsschritt 1: für $x = n+1$: Zeige per Induktion nach y: $\alpha(n+1, y)$ ist definiert
  - Induktionsanfang 2: für $y = 0$ ist $\alpha(n + 1, 0) = \alpha(n, 1)$ und somit nach erster Induktionsvoraussetzung definiert
  - Induktionsvoraussetzung 2: für alle $y \leq m$ sei $\alpha(n + 1, y)$ definiert.

– Induktionsschritt 2: für $y = m+1$ ist $\alpha(n+1, m+1) = \alpha(n, \alpha(n+1, m))$, dabei ist $k := \alpha(n+1, m)$ definiert nach Induktionsvoraussetzung 2, weiter ist $\alpha(n, k)$ definiert nach Induktionsvoraussetzung 1.

Bemerkung: Es gilt:

- $\alpha(0, y) = y + 1 > y$
- $\alpha(1, y) = y + 2 > y + 1$
- $\alpha(2, y) = 2y + 3 > 2y$ wegen

$$\begin{aligned} \alpha(2, y) &= \alpha(1, \alpha(2, y-1)) = \alpha(2, y-1) + 2 = \alpha(2, y-2) + 4 \\ &= \ldots = \alpha(2, 0) + 2y = \alpha(1, 1) + 2y = 2y + 3 \end{aligned}$$

- $\alpha(3, y) > 2^y$ wegen

$$\alpha(3, y) = \alpha(2, \underbrace{\alpha(3, y-1)}_{>2^{y-1}}) > \alpha(2, 2^{y-1}) > 2 \cdot 2^{y-1} = 2^y$$

LEMMA: Es gilt folgendes:

1. $\alpha(x, y) > y$
2. $\alpha(x, y+1) > \alpha(x, y)$ (Monotonie im zweiten Argument)
3. $\alpha(x+1, y) \geq \alpha(x, y+1)$
4. $\alpha(x+1, y) > \alpha(x, y)$
5. $\alpha(x+2, y) > \alpha(x, 2y)$

Beweis:

1. per Induktion nach x:
   - Induktionsanfang 1: für $x = 0$ ist $\alpha(0, y) = y + 1 > y$
   - Induktionsvoraussetzung 1: für alle $x \leq n$ für alle y sei $\alpha(n, y) > y$
   - Induktionsschritt 1: für $x = n + 1$: per Induktion nach y:
     – Induktionsanfang 2: für $y = 0$ ist $\alpha(n+1, 0) = \alpha(n, 1) > 1 > 0$
     – Induktionsvoraussetzung 2: für alle $y \leq m$ sei $\alpha(n+1, y) > y$

– Induktionsschritt 2: für $y = m + 1$ ist $\alpha(n + 1, m + 1) = \alpha(n, \alpha(n+1, m)) > \alpha(n+1, m)$ nach Induktionsvoraussetzung 2 und weiter ist $\alpha(n+1, m) > m$ nach Induktionsvoraussetzung 1, damit ist $\alpha(n + 1, m + 1) > m + 1$ (da $\alpha\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$)

2. Fallunterscheidung:

   - für $x = 0$ ist $\alpha(0, y + 1) = y + 2 > y + 1 = \alpha(0, y)$
   - für $x > 0$ folgt aus Teil 1: Beachte $\alpha(x - 1, \bar{y}) > \bar{y}$ mit $\bar{y} = \alpha(x, y)$ für alle $x > 0$, also ist $\alpha(x, y + 1) = \alpha(x - 1, \alpha(x, y)) > \alpha(x, y)$

3.-5. siehe evtl. Übung

SATZ: Die Ackermann-Funktion α ist nicht LOOP-berechenbar.

Notation: Zu einem LOOP-Programm P mit m Variablen $a = (a_1, \ldots, a_m)$ gibt die semantische Funktion $[P](a_1, \ldots, a_m) = (b_1, \ldots, b_m) =: b$. Definiere Funktionen

$$\underline{\max}(b) := \max\{\, b_i \mid i = 1, \ldots, m\} \text{ und } \bar{f}_P(a) := \underline{\max}([P](a_1, \ldots, a_m))$$

LEMMA: Für jedes LOOP-Programm P mit m Variablen gibt es eine Konstante $k_p \in \mathbb{N}$ mit

$$\bar{f}_P(a) < \alpha(k_P, \underline{\max}(a)) \quad \forall\, a = (a_1, \ldots, a_m)$$

Beweis: per Induktion über den Aufbau der LOOP-Programme:

- Induktionsanfang: für Wertzuweisungen $X_j := X_i \pm 1$ gilt

  $$a_i \pm 1 \leq \underline{\max}(a) + 1 \leq 2\underline{\max}(a) + 1$$

  Damit folgt

  $$\begin{aligned} \bar{f}_P(a) &= \underline{\max}([P](a_1, \ldots, a_m)) \leq 2\underline{\max}(a) + 1 \\ \text{mit Lemma 1:} \quad &< \alpha(0, 2\underline{\max}(a) + 1) \\ \text{mit Lemma 3:} \quad &\leq \alpha(1, 2\underline{\max}(a)) \\ \text{mit Lemma 5:} \quad &\leq \alpha(3, \underline{\max}(a)) \end{aligned}$$

  Wähle also $k_P = 3$.

- Induktionsschritt:

– für $P == P_1; P_2$ wende die Induktionsvoraussetzung an und erhalte k_1 und k_2 mit $\bar{f}_{P_i} < \alpha(k_i, \underline{\max}(a))$. Setze $k := \max\{k_1, k_2\}$ und $k_P := k + 2$. Dann ist

$$\begin{aligned}
\bar{f}_P(a) &= \underline{\max}([P_2]([P_1](a))) \\
&= \bar{f}_{P_2}([P_1](a)) \\
\text{IV:} \quad &< \alpha(k_2, \underline{\max}([P_1](a))) \\
&< \alpha(k_2, \bar{f}_{P_1}(a)) \\
\text{IV und Lemma 2:} \quad &< \alpha(k_2, \alpha(k_1, \underline{\max}(a))) \\
\text{Lemma 4, 2:} \quad &< \alpha(k_2, \alpha(k+1, \underline{\max}(a))) \\
\text{Lemma 4:} \quad &< \alpha(k, \alpha(k+1, \underline{\max}(a))) \\
\text{Regel (3):} \quad &= \alpha(k+1, \underline{\max}(a) + 1) \\
\text{Lemma 3:} \quad &< \alpha(k+2, \underline{\max}(a))
\end{aligned}$$

– für $P ==$ `loop` X_i `do begin` P_1 `end`; wende die Induktionsvoraussetzung auf P_1 an und erhalte k_1. Setze $k_P := k_1 + 3$.

Zeige die Aussage $(\star)$: Für alle $y \geq 0$ ist $\underline{\max}[P_1]^y(a) < \alpha(k_1 + 1, \underline{\max}(a) + y)$.

Beweis per Induktion nach y:

* Induktionsanfang: für $y = 0$ ist

$$\underline{\max}[P_1]^0(a) = \underline{\max}(a) < \alpha(k_1 + 1, \underline{\max}(a))$$

* Induktionsschritt $y \rightsquigarrow y + 1$:

$$\begin{aligned}
\underline{\max}[P_1]^{y+1}(a) &= \underline{\max}[P_1]([P_1]^y(a)) \\
\text{IV für } P_1\text{:} \quad &< \alpha(k_1, \underline{\max}[P_1]^y(a)) \\
\text{IV für } y\text{:} \quad &< \alpha(k_1, \alpha(k_1 + 1, \underline{\max}(a) + y)) \\
\text{Regel (3):} \quad &= \alpha(k_1 + 1, \underline{\max}(a) + y + 1)
\end{aligned}$$

Aus $(\star)$ folgt:

$$\begin{aligned}
\bar{f}_P(a) &= \underline{\max}([P](a)) \\
&= \underline{\max}([P_1]^{a_i}(a)) \\
\text{mit } (\star)\text{ :} \quad &< \alpha(k_1 + 1, \underline{\max}(a) + a_i) \\
&\leq \alpha(k_1 + 1, 2\underline{\max}(a)) \\
\text{Lemma 5:} \quad &\leq \alpha(k_1 + 3, \underline{\max}(a))
\end{aligned}$$

SATZ: Die Ackermann-Funktion α ist nicht LOOP-berechenbar.

Beweis: *Angenommen*, α sei durch ein LOOP-Programm P berechnet mit m Variablen $X_1, \ldots, X_m$. Wähle k_p zu P gemäß obigem Lemma. Dann gilt jedoch:

$$\begin{aligned} \alpha(x,y) &= f_P^{(2)}(x,y,0,\ldots) \\ &\leq \bar{f}_P(x,y,0,\ldots) \\ &< \alpha(k_P, \underline{\max}((x,y,0,\ldots))) \\ &= \alpha(k_P, \max\{x,y\}) \end{aligned}$$

Wähle nun $x = y = k_P$ und erhalte folgenden Widerspruch

$$\alpha(k_P, k_P) < \alpha(k_P, \max\{k_P, k_P\}) = \alpha(k_P, k_P)$$

Bemerkung: Die Ackermann-Funktion ist WHILE-berechenbar. **Folgerung:** Es gibt totale, WHILE-berechenbare (d.h. μ-rekursive) Funktionen, die nicht LOOP-berechenbar (d.h. primitiv rekursiv) sind.

4.4 Halteproblem, Unentscheidbarkeit

DEFINITION: Es sei $\Sigma = \{a_1, \ldots, a_n\}$. Zu $L \subseteq \Sigma^*$ definiere $\chi_L \colon \Sigma^* \to \Sigma^*$ durch die charakteristische Funktion von L:

$$\chi_K(w) = \begin{cases} a_1 & \text{falls} \quad w \in L \\ \varepsilon & \text{falls} \quad w \notin L \end{cases}$$

Beispiel: Für $L = \{\,|^n \mid n \text{ Primzahl}\} \subseteq \{|\}^*$ ist $\chi_L = f_{\text{prim}}$.

DEFINITION: Eine Turingmaschine $\mathfrak{A}$ *entscheidet* L, falls $\mathfrak{A}$ die charakteristische Funktion χ_L berechnet (d.h. $\mathfrak{A}$ erreicht eine Stoppkonfiguration $q\chi_l(w)$ für alle $w \in \Sigma^*$). L ist *Turing-entscheidbar*, falls eine Turingmaschine $\mathfrak{A}$ existiert, die L entscheidet.

DEFINITION: Eine Turingmaschine $\mathfrak{A}$ *akzeptiert* L genau dann, wenn $\mathfrak{A}$ für genau die Eingabewörter $w \in L$ eine Stoppkonfiguration erreicht. L ist *Turing-akzeptierbar*, falls eine Turingmaschine $\mathfrak{A}$ existiert, die L akzeptiert.

Bemerkung: Wenn L Turing-enscheidbar ist, ist sie Turing-akzeptierbar.[10]

[10] „Was? Frage?“

Beweis: $\mathfrak{A}$ entscheide L. Konstruiere eine Turingmaschine $\mathfrak{B}$, die L akzeptiert, wie folgt: Sei $Q_{\mathfrak{B}} = Q_{\mathfrak{A}} \cup \{q_\infty\}$ und setze

$$\delta_{\mathfrak{B}}(q,b) = \begin{cases} \delta_{\mathfrak{A}}(q,b) & \text{falls} \quad \delta_{\mathfrak{A}}(q,b) \neq \perp \\ (b, r, q_\infty) & \text{falls} \quad \delta_{\mathfrak{A}}(q,b) = \perp \text{ und } b = \text{ƀ} \end{cases}$$

SATZ: Eine Sprache L ist Turing-entscheidbar genau dann, wenn L und $\Sigma^* \setminus L$ beide Turing-akzeptierbar sind.

Beweis:

„⇒“ Aus der Entscheidbarkeit von L folgt mit der Bemerkung oben eine Turingmaschine, die L akzeptiert. Nehme nun eine Maschine, die L entscheidet und vertausche hierin die Ausgaben für *ja* ($w \in L$) und *nein* ($w \in L$), mit der Bemerkung folgt aus dieser Entscheidbarkeit auch die Akzeptierbarkeit.

„⇐“ Seien $\mathfrak{A}_1, \mathfrak{A}_2$ die Turingmaschinen, die L bzw. $\Sigma^* \setminus L$ akzeptieren. Die gesuchte Turingmaschine $\mathfrak{B}$ kopiert zunächst w auf ein zweites Band und läßt dann $\mathfrak{A}_1$ und $\mathfrak{A}_2$ abwechselnd jeweils einen Schritt machen (auf Band 1 bzw. 2) bis $\mathfrak{A}_1$ oder $\mathfrak{A}_2$ anhält. Dies geschieht nach endlich vielen Schritten, da $w \in L$ oder $w \in \Sigma^* \setminus L$. Hält $\mathfrak{A}_1$ an, so erzeugt $\mathfrak{B}$ auf dem ersten Band a_1, hält $\mathfrak{A}_2$ an, so erzeugt $\mathfrak{B}$ auf dem ersten Band nur ε.

(formal in der Übung)

4.4.1 Universelle Turingmaschine

Ziel: Es gibt unentscheidbare Probleme/Sprachen.

Sei $\Sigma = \{a, b\}$. Zunächst kodieren wir jede deterministische Turingmaschine durch ein Wort über Σ: Zu $\mathfrak{A}$ sei $\text{code}(\mathfrak{A}) \in \Sigma^*$ wir folgt definiert: Sei $\Gamma = \{a_1, \ldots, a_k\}$ mit $a_1 = a$, $a_2 = b$, $Q = \{q_1, \ldots, q_n\}$ mit Anfangszustand q_1 und Endzustand q_n. Für jede δ-Regel $\delta(q_i, a_j) = (a_{j'}, l/r, q_{i'})$ sei

$$\text{code}(\delta(q_i, a_j)) = a^i b a^j b a^{j'} b \frac{a}{aa} b a^{i'} b b$$

Alle diese zu δ gehörenden Wörter $\text{code}(\delta(q_i, a_j))$ schreiben wir in beliebiger Reihenfolge hintereinaner und fügen hinten noch ein b an:

$$\text{code}(\mathfrak{A}) = \text{code}(\delta_1)\text{code}(\delta_2) \cdots \text{code}(\delta_x) b$$

Bemerkung:

1. In einem Wort $\text{code}(\mathfrak{A})w$ beginnt w nach dem ersten Block von drei b's.
2. $L = \{\text{code}(\mathfrak{A}) \mid \mathfrak{A} \text{ DTM über } \Sigma\}$ ist Turing-entscheidbar.

DEFINITION: Definiere die *universelle Sprache*

$$\text{Univ} := \{\text{code}(\mathfrak{A})w \mid \mathfrak{A} \text{ DTM über } \Sigma, \mathfrak{A} \text{ akzeptiert } w \in \Sigma^*\}$$

SATZ: Univ ist Turing-akzeptierbar.

Beweis: Für Eingabe $x \in \Sigma^*$

1. Teste, ob x die Form $\text{code}(\mathfrak{A})w$ hat. Falls nein, laufe in Endlosschleife.
2. Simuliere $\mathfrak{A}$ (in $\text{code}(\mathfrak{A})$ kodiert) auf Eingabe w:
   (a) Übersetze w gemäß Kodierung von $\text{code}(\mathfrak{A})$:
   $$\text{code}(\mathfrak{A})a_2a_2a_1 \to \text{code}(\mathfrak{A})aa\,b\,aa\,b\,a\,b$$
   (b) Markierung des Arbeitsfeldes und des Zustands q_i, z.B. durch Wort $\not\in a^i b$ an der entsprechenden Stelle: Anfangskonfiguration $\text{code}(\mathfrak{A})\not\in abaabaabab$ (hierbei $a^1 \mathrel{\hat{=}} q_1$).
   (c) Schrittsimulation: Bei Konfiguration $\ldots \not\in a^i b a^j b \ldots$ suche die δ-Regel $\delta(q_i, a_j)$ in $\text{code}(\mathfrak{A})$ kodiert und führe die Änderung der Konfiguration durch
   (d) Bei Endzustand akzeptiere x, sonst keine Termination.

SATZ: Univ ist nicht Turing-entscheidbar.

Beweis: [11] Annahme, Univ wäre Turing-entscheidbar und $\mathfrak{A}$ entscheidet Univ. Vertausche die Ausgaben ja/nein und erhalte Turingmaschine $\bar{\mathfrak{A}}_0$, die $\Sigma \setminus \text{Univ}$ entscheidet. Betrachte

$$D = \{\text{code}(\mathfrak{A}) \mid \text{code}(\mathfrak{A})\,\text{code}(\mathfrak{A}) \notin \text{Univ}\}$$

Die folgende Turingmaschine $\mathfrak{A}_D$ akzeptiert D:

1. Teste, ob $x = \text{code}(\mathfrak{A})$ für eine Turingmaschine $\mathfrak{A}$. Falls nein, führe Endlosschleife aus.
2. Kopiere $\text{code}(\mathfrak{A})$ zu $\text{code}(\mathfrak{A})\,\text{code}(\mathfrak{A})$

[11] Beweismethode: Diagonalschluß

3. Rufe $\bar{\mathfrak{A}}_0$ auf. Bei ja-Antwort halte an, ansonsten Endlosschleife.

Dann gilt für alle deterministischen Turingmaschinen $\mathfrak{A}$ über Σ:

$$\begin{aligned} & \mathfrak{A}_D \text{ akzeptiert } \mathrm{code}(\mathfrak{A}) \\ \Leftrightarrow\ & \mathrm{code}(\mathfrak{A})\,\mathrm{code}(\mathfrak{A}) \notin \mathrm{Univ} \\ \Leftrightarrow\ & \mathfrak{A} \text{ akzeptiert nicht } \mathrm{code}(\mathfrak{A}) \end{aligned}$$

Für $\mathfrak{A} = \mathfrak{A}_D$ erhält man einen Widerspruch! **Folgerungen:**

1. $\Sigma^* \setminus \mathrm{Univ}$ ist nicht Turing-akzeptierbar
2. $\Sigma^* \setminus \mathrm{Univ}$ ist nicht durch eine Typ-0-Grammatik erzeugbar.

Verwende die Church'sche These:

SATZ: Es gibt keinen Algorithmus, der zu einer Turingmaschine $\mathfrak{A}$ über Σ entscheidet, ob $\mathfrak{A}$ das Wort w akzeptiert. D.h. das Wortproblem für deterministische Turingmaschinen ist unentscheidbar.

Durch Reduktion/Transformation erhält man viele weitere Unentscheidbarkeitsergebnisse.

4.4.2 Wortproblem für DTMs

SATZ: Das Problem „$\mathfrak{A}$ akzeptiert ε“ für deterministische Turingmaschinen ist unentscheidbar.

Beweis: verwende eine berechenbare Transformation $f\colon (\mathfrak{A}, w) \mapsto \mathfrak{A}_w$ mit

$$\mathfrak{A} \text{ akzeptiert } w \iff \mathfrak{A}_w \text{ akzeptiert } \varepsilon \qquad (\star)$$

Dann liefert ein Algorithmus A_0 für „$\mathfrak{A}$ akzeptiert ε“ folgenden Algorithmus A_1 für „$\mathfrak{A}$ akzeptiert w“: Zu $\mathfrak{A}, w$ bestimme $\mathfrak{A}_w = f(\mathfrak{A}, w)$ und wende A_0 auf $\mathfrak{A}_w$ an. A_1 gibt dann ja aus, falls $\mathfrak{A}_w$ das leere Wort akzeptiert, sonst nein. Wegen der Bedingung $(\star)$ und der Berechenbarkeit von f ist A_1 korrekt.

Idee zu $\mathfrak{A}_w$: Die Turingmaschine $\mathfrak{A}_w$ schreibt w auf das leere Band und arbeitet dann wie $\mathfrak{A}$. Zu $\mathfrak{A} = (Q, \Sigma, \Gamma, q_0, \delta, F)$ und $w = a_1 \cdots a_n$ ist $\mathfrak{A}_w$ wie

folgt definiert: $\mathfrak{A}_w = (Q'_w, \Sigma, \Gamma, \bar{q}_0, \bar{\delta}, F)$ mit $Q'_w = Q \cup \{\bar{q}_0, \ldots, \bar{q}_{n+1}\}$

$$\begin{aligned}
\bar{\delta}(\bar{q}_0, ß) &= (a_1, r, \bar{q}_1) \\
\bar{\delta}(\bar{q}_i, ß) &= (a_{i+1}, r, \bar{q}_{i+1}) \; \forall \; i = 1, \ldots, n-2 \\
\bar{\delta}(\bar{q}_n, ß) &= (ß, l, \bar{q}_{n+1}) \\
\bar{\delta}(\bar{q}_{n+1}, a) &= (a, l, \bar{q}_{n+1}) \; \forall \; a \in \Sigma \\
\bar{\delta}(\bar{q}_{n+1}, ß) &= (ß, r, q_0) \\
\bar{\delta}(q, a) &= \delta(q, a) \; \forall \; q \in Q, a \in \Gamma
\end{aligned}$$

4.5 Entscheidungsprobleme

DEFINITION: Ein *Entscheidungsproblem* ist von der Form $\Pi = (E_P, P)$, wobei

- E_P die Menge der möglichen Eingaben ist und
- $P \subseteq E_P$ ist.

Π ist entscheidbar, falls ein Algorithmus existiert, der zu jeder Eingabe (d.h. zu jedem $x \in E_P$) terminiert und die Antwort liefert, ob $x \in P$ ist.

Beispiele:

1. Wortproblem (z.B. für DTM): $\Pi_1 = (E_{WP}, WP)$ mit E_{WP} als Menge aller Paare $(\mathfrak{A}, w)$, wobei $\mathfrak{A}$ eine DTM über Σ und $w \in \Sigma^*$ ist; sowie WP die Menge aller $(\mathfrak{A}, w)$ mit $\mathfrak{A}$ akzeptiert w.

2. Halteproblem für DTM: $\Pi_2 = (E_{HP}, HP)$ mit E_{HP} die Menge aller DTMs $\mathfrak{A}$ über Σ und HP als Menge aller DTMs $\mathfrak{A}$, die gestartet auf dem leeren Band anhalten.

   ! Beide Probleme sind unentscheidbar für beliebige DTM.

3. Äquivalenzproblem für DTM: $\Pi_3 = (E_{\ddot{A}P}, \ddot{A}P)$ mit $E_{\ddot{A}P}$ als Menge aller Paare $(\mathfrak{A}_1, \mathfrak{A}_2)$ von DTMs über Σ und

$$\ddot{A}P = \{\, (\mathfrak{A}_1, \mathfrak{A}_2) \mid \mathfrak{A}_1, \mathfrak{A}_2 \text{ DTM und } L(\mathfrak{A}_1) = L(\mathfrak{A}_2)\}$$

DEFINITION: Seien $\Pi_1 = (E_P, P)$ und $\Pi_2 = (E_Q, Q)$ Entscheidungsprobleme. Π_1 heißt *reduzierbar auf* Π_2 (in Zeichen $\Pi_1 \leq \Pi_2$) genau dann, wenn eine berechenbare totale Funktion $f\colon E_P \to E_Q$ existiert mit der Eigenschaft

$$\forall \; x \in E_P \colon x \in P \Leftrightarrow f(x) \in Q \qquad (\star)$$

D.h. eine ja- Eingabe von x wird abgebildet auf eine ja-Eingabe von $f(x)$.

LEMMA: (Resolutionslemma) Seien Π_1, Π_2 Entscheidungsprobleme. Ist Π_1 unentscheidbar und $\Pi_1 \leq \Pi_2$, so ist auch Π_2 unentscheidbar.

Beweis: Annahme: Ein Algorithmus A_0 entscheide Π_2. Folgender Algorithmus entscheidet dann Π_1: Zu $x \in E_P$ bestimme $f(x)$ mit dem Algorithmus A_f zur berechenbaren Funktion f und wende A_0 an auf $f(x)$. Dann möge der Algorithmus $x \in P$ ausgeben genau dann, wenn $f(x) \in Q$ ist.

Dieser Algorithmus entscheidet dann Π_1, das ist ein Widerspruch!

Bemerkung: Statt $(\star)$ verwendet man manchmal auch:

$$x \in P \Longleftrightarrow f(x) \notin Q$$

Ziel: Äquivalenzproblem für kontextfreie Grammatiken.

4.5.1 Das Postsche Korrespondenzproblem (PCP)

DEFINITION: Das *(modifizierte) Postsche Korrespondenzproblem* (M)PCP besteht aus E_{PCP} als Menge der Listenpaare (A, B) mit $A = (w_1, \ldots, w_k)$, $B = (x_1, \ldots, x_k)$ mit $w_i, x_i \in \Sigma^+$. Eine (spezielle) Lösung für A, B ist eine Indexfolge $i_1, \ldots, i_m$ (speziell: $i_1 = 1$) mit

$$w_{i_1} \cdot \ldots \cdot w_{i_m} = x_{i_1} \cdot \ldots \cdot x_{i_m}$$

PCP (bzw. MPCP) ist die Menge aller Paare (A, B), die eine (spezielle) Lösung haben.

Beispiele:

1. Sei $\Sigma = \{0, 1\}$ und sei gegeben:

i	Liste A	Liste B
1	$w_1 = 1$	$x_1 = 111$
2	$w_2 = 10111$	$x_2 = 10$
3	$w_3 = 10$	$x_3 = 0$

   Eine mögliche Lösung ist $2, 1, 1, 3$: $10111|1|1|10 = 10|111|111|0$.

2. Sei wieder $\Sigma = \{0, 1\}$ und

i	Liste A	Liste B
1	$w_1 = 10$	$x_1 = 101$
2	$w_2 = 011$	$x_2 = 11$
3	$w_3 = 101$	$x_3 = 011$

Eine Lösung muß mit Index 1 beginnen, die nächste Auswahl in A muß mit 1 beginnen, also 1 oder 3. Bei 1 ergibt sich in A 1010, in B aber 101101, was falsch ist. Bei $i_2 = 3$ folgt:

A	$10\|101$
B	$101\|011$

Die Argumentation läßt sich wiederholen, es gilt damit $i_3 = 3$, $i_4 = 3$ usw., dann ist aber das erzeugte Wort in B immer länger, d.h. diese Instanz hat keine Lösung.

SATZ: MPCP ist unentscheidbar.

Beweis: Es gilt: Wortproblem für DTMs $\leq$ MPCP! Gesucht ist also eine Abbildung $f\colon (\mathfrak{A}, w) \to (A, B)$ mit $\mathfrak{A}$ akzeptiert w genau dann, wenn A, B eine spezielle Lösung hat. **Idee:** Sei beispielsweise folgende Turingmaschine gegeben:

$$\begin{aligned} \mathfrak{A}\colon \quad \delta(q_0, 0) &= (1, r, q_1) \\ \delta(q_1, 1) &= (1, r, q_2) \\ \delta(q_2, \text{ƀ}) &= (1, l, q_3) \text{ mit } q_3 \in F \end{aligned}$$

Sei $w = 01$, Konfigurationen von $\mathfrak{A}$ sind dann:

$$q_0 01 \to 1 q_1 1 \to 11 q_2 \to 1 q_3 11$$

Diese werden kodiert durch

$$\#q_0 01 \# 1 q_1 1 \# 11 q_2 \# 1 q_3 11 \#$$

Wir simulieren eine akzeptierende Konfigurationsfolge durch Listen A, B mit $\Gamma = \{0, 1, \text{ƀ}\}$:

A	B	
$\#$	$\#q_0 01\#$	
a	a	$a \in \Gamma$
$\#$	$\#$	
$q_0 0$	$1 q_1$	
$q_1 1$	$1 q_2$	
$a q_2 \text{ƀ}$	$q_3 a 1$	$a \in \Gamma$
$a q_2 \#$	$q_3 a 1 \#$	$a \in \Gamma$
$q_3 a$	e	$a \in \Gamma$
$a e a'$	e	$a, a' \in \Gamma$
$e \# \#$	$\#$	

Die Indexfolge ergibt dann in A und B:

A	$\#\|q_0 0\|1\|\#\|1\|q_1 1\|\#\|1\|1q_2\#\|1\|q_3 1\|1\|\#\|1e1\|\#\|e\#\#$
B	$\#q_0 01\#\|1q_1\|1\|\#\|1\|1q_2\|\#\|1\|q_3 11\#\|1\|e\|1\|\#\|e\|\#\|\#$

Allgemein:

A	$\#K_0\#K_1\#\ldots$
B	$\#K_0\#K_1\#K_2\#\ldots$

Es wird zunächst die Anfangskonfiguration in B gewählt, dann $\mathfrak{A}$ simuliert, wobei B immer ein Zustand voraus eilt, danach wird der B-Wert eingeholt und ggf. eine Stoppkonfiguration erreicht.

Im allgemeinen Fall wird zu der DTM $\mathfrak{A} = (Q, \Sigma, \Gamma, \Delta, q_0, F)$ und $w \in \Sigma^*$ das Paar A, B wie folgt definiert:

A	B	
$\#$	$\#q_0 w\#$	
a	a	$a \in \Gamma$
$\#$	$\#$	
qa	bq'	$\delta(q,a) = (b,r,q')$
cqa	$q'cb$	$\delta(q,a) = (b,l,q'), c \in \Gamma$
$q\#$	$bq'\#$	$\delta(q,\text{ƀ}) = (b,r,q')$
$cq\#$	$q'cb\#$	$\delta(q,\text{ƀ}) = (b,l,q'), c \in \Gamma$
$\#qa$	$\#q'\text{ƀ}b$	$\delta(q,a) = (b,l,q'), c \in \Gamma$
qa	e	$q \in F, a \in \Gamma$(und $\delta(q,a) = \perp$ in diesem Fall)
$q\#$	$e\#$	$q \in F$(und $\delta(q,\text{ƀ}) = \perp$)
aeb	e	$a, b \in \Gamma$
$ae\#$	$e\#$	$a \in \Gamma$
$\#ea$	$\#e$	$a \in \Gamma$
$e\#\#$	$\#$	

Es bleibt noch zu zeigen: $\mathfrak{A}$ akzeptiert w genau dann, wenn A, B eine spezielle Lösung hat. Der Beweis folgt aus der folgenden Behauptung, die man per Induktion beweisen kann: Ist $K_0, \ldots, K_j$ eine Konfigurationsfolge von $\mathfrak{A}$ mit $K_0 = q_0 w$, dann beginnt jede spezielle Lösung von A und B mit der Indexfolge $(1, i_2, \ldots, i_w)$, die zu Wörtern $\#K_0\#K_1\#\ldots\#K_{j-1}\#$ (bezüglich A) und $\#K_0\#K_1\#\ldots\#K_{j-1}\#K_j\#$ (bezüglich B) führt.

SATZ: MPCP $\leq$ PCP

Beweis: Durch Angabe einer Transformation $(A, B) \mapsto (C, D)$ mit der Eigenschaft: (A, B) hat spezielle Lösung genau dann, wenn (C, D) eine Lösung

hat. Seien ¢ und \$ neue Buchstaben. Definiere Abbildungen h_r und h_l durch die folgenden Eigenschaften:

$$\begin{aligned} h_r(a_1, \ldots, a_n) &= a_1 ¢ a_2 ¢ \ldots a_n ¢ \\ h_l(a_1, \ldots, a_n) &= ¢ a_1 ¢ a_2 \ldots ¢ a_n \end{aligned}$$

Zu $A = (w_1, \ldots, w_k)$ und $B = (x_1, \ldots, x_k)$ definiere $C = (y_0, \ldots, y_{k+1})$ und $D = (z_0, \ldots, z_{k+1})$ mit

$$\begin{array}{rclcrcl} y_0 & = & ¢ h_r(w_1) & & z_0 & = & h_l(x_1) \\ y_1 & = & h_r(w_1) & & z_0 & = & h_l(x_1) \\ \vdots & & \vdots & & \vdots & & \vdots \\ y_k & = & h_r(w_k) & & z_0 & = & h_l(x_k) \\ y_{k+1} & = & \$ & & z_{k+1} & = & ¢\$ \end{array}$$

Beispiel:

i	Liste A	Liste B	Liste C	Liste D
0			¢1¢	¢1¢1¢1
1	1	111	1¢	¢1¢1¢1
2	10111	10	1¢0¢1¢1¢	¢1¢0
3	10	0	1¢0¢	¢0
4			\$	¢\$

Nun hat (A, B) eine spezielle Lösung genau dann, wenn (C, D) eine Lösung hat. *Beweis:*

„⇒“ (A, B) habe Lösung $(1, i_2, \ldots, i_m)$ mit $w_1 w_{i_2} \cdots w_{i_m} = x_1 x_{i_2} \cdots x_{i_m}$. Dann ist

$$¢ h_r(w_1) h_r(w_{i_2}) \cdots h_r(w_{i_m}) \$ = h_l(x_1) h_l(x_{i_2}) \cdots h_l(x_{i_m}) ¢ \$$$

Somit hat (C, D) die Lösung $0, i_2, \ldots, i_m, k+1$

„⇐“ Sei $i_1, \ldots, i_m$ Lösung für (C, D). Dann ist $i_1 = 0$ und $i_m = k+1$. Betrachte die kürzeste Teilfolge $i_1, \ldots, i_j$ mit $i_j = k+1$. Schon das liefert eine Lösung für (C, D):

$$y_{i_1} \cdots y_{i_j} = z_{i_1} \cdots z_{i_j} \text{ und } i_1 = 0;\ i_j = k+1$$

(und \$ nur am Ende in y_{i_j} und z_{i_j}). Dann ist

$$¢ h_r(w_1) h_r(w_{i_2}) \cdots h_r(w_{i_{j-1}}) \$ = h_l(x_1) h_l(x_{i_2}) \cdots h_l(x_{i_{j-1}}) ¢ \$$$

Nach Streichen der ¢ und \$ gilt:

$$w_1 w_{i_2} \cdots w_{i_{j-1}} = x_1 x_{i_2} \cdots x_{i_{j-1}}$$

Damit ist $1, i_2, \ldots, i_{j-1}$ eine spezielle Lösung für (A, B).

4.5.2 Entscheidbarkeit von Problemen für alle Sprachtypen

Nächstes Ziel: Vervollständigung der folgenden Tabelle:

Typ	Typ 3		Typ 2	Typ 1	Typ 0
Automaten	NEA/DEA	DPDA	PDA	LBA	TM
Bezeichnung	regulär	determin. kontextfrei	kontextfrei	kontext-sensitiv	rekursiv aufzählbar
Entscheidungen					
Wortproblem	✓	✓	✓	✓	✗
Leerheitsproblem	✓	✓	✓	**?**	?
Äquivalenzproblem	✓	✓	**?**	?	?

Satz: Das Leerheitsproblem für LBAs (kontextsensitive Grammatiken) ist unentscheidbar.

Beweis: Wir zeigen „PCP $\leq$ Leerheitsproblem für LBAs“ durch Angabe einer Transformation $(A, B) \mapsto \mathfrak{A}$ mit

$$(A, B) \text{ hat Lösung} \iff L(\mathfrak{A}) \neq \emptyset \qquad (\star)$$

Seien $A = (w_1, \ldots, w_k)$, $B = (x_1, \ldots, y_k)$ gegeben. $\mathfrak{A}$ testet zur Eingabe $w \in \Sigma^+$, ob w durch eine Lösung für (A, B) entsteht. Dies funktioniert wie folgt: Angesetzt auf $\not\in w\$$ generiert $\mathfrak{A}$ nicht-deterministisch eine Folge $i_1, \ldots, i_m$ mit $i_j \in \{1, \ldots, k\}$ und $m \leq |w|$ durch Annahme von ausgezeichneten Zuständen $q_{i_1}, \ldots, q_{i_m}$ nacheinander. Mit Hilfsbuchstaben aus Γ werden auf der Eingabe w die Zerlegungen $w_{i_1} w_{i_2} \cdots$ bzw. $x_{i_1} x_{i_2} \cdots$ markiert:

$$\not\in a_1 a_2 \underline{a_3} a_4 \overline{\underline{a_5} a_6} \cdots \overline{\underline{a_n}}\$$

(d.h. bezüglich A ist die Zerlegung $a_1 a_2 a_3 | a_4 a_5 |$, bezüglich B ist die Zerlegung $a_1 a_2 a_3 a_4 a_5 | a_6 |$) Ausgehend vom Zustand q wählen wir (nicht-deterministisch) einen Zustand q_{i_l} aus. Dann vergleichen wir ausgehend von der letzten Markierung $\underline{b}$, ob das Folgewort gleich w_{i_l} ist, und ausgehend von der letzten Markierung $\overline{b}$ testen wir, ob das Folgewort x_{i_l} ist. Für diese Vergleiche verwenden wir zusätzliche Hilfszustände. Ist einer der Vergleiche nicht erfolgreich, so laufe in Endlosschleife. Ansonsten markiere die Endpunkte mit $\underline{b'}$ und $\overline{b''}$. Falls das letzte Symbol a_n unten und oben markiert ist, akzeptieren wir das Wort (Übergang in Endzustand). Diese Berechnungen sind durch LBA realisierbar und $\mathfrak{A}$ akzeptiert w genau dann, wenn w entsteht als Lösung von (A, B).

SATZ: Das Entscheidungsproblem $L(G_1) \cap L(G_2) = \emptyset$ für zwei kontextfreie Grammatiken G_1, G_2 ist unentscheidbar.

Beweis: Wir zeigen: „PCP $\leq$ Durchschnittsproblem“. Sei eine Abbildung $(A, B) \mapsto (G_1, G_2)$ definiert mit

$$(A, B) \text{ hat Lösung} \iff L(G_1) \cap L(G_2) \neq \emptyset \qquad (\star)$$

Seien A, B Listen über Σ der Länge k. Verwende als Hilfsalphabet neue Buchstaben $K = \{c_1, \ldots, c_k\}$ und setze $\tilde{\Sigma} = \Sigma \cup K \cup \{\$\}$. Dann seien folgende Sprachen definiert:

$$\begin{aligned} L_1 &= \left\{ y\$y^T \;\middle|\; y \in \Sigma^+ K^+ \right\} \\ L_2 &= \left\{ y\$z^T \;\middle|\; y = w_{i_1} \cdots w_{i_m} c_{i_m} \cdots c_{i_1}, z = x_{j_1} \cdots x_{j_r} c_{j_r} \cdots c_{j_1} \right\} \end{aligned}$$

Beide Sprachen L_1 und L_2 sind deterministisch kontextfrei (als Übung). Seien G_1, G_2 Grammatiken mit $L_i = L(G_i)$.

- Wenn nun (A, B) die Lösung $i_1, \ldots, i_m$ hat, dann ist

$$\begin{aligned} w := w_{i_1} \cdots w_{i_m} &= x_{i_1} \cdots x_{i_m} =: x \\ \Rightarrow w c_{i_m} \cdots c_{i_1} &= x c_{i_m} \cdots c_{i_1} \\ \Rightarrow w c_{i_m} \cdots c_{i_1} \quad \$ & \quad c_{i_1} \cdots c_{i_m} w^T \in L_1 \\ \text{und } w c_{i_m} \cdots c_{i_1} \quad \$ & \quad c_{i_1} \cdots c_{i_m} x^T \in L_2 \end{aligned}$$

  Damit ist $L_1 \cap L_2 \neq \emptyset$.

- Sei umgekehrt nun $v \in L_1 \cap L_2 \neq \emptyset$. Dann ist wegen $v \in L_2$:

$$v = w_{i_1} \cdots w_{i_m} c_{i_m} \cdots c_{i_1} \$ (x_{j_1} \cdots x_{j_r} c_{j_r} \cdots c_{j_1})^T$$

  Gleichzeitig ist wegen $v \in L_1$ und $K \cap \Sigma = \emptyset$: $r = m$ und $i_1 = j_1$ usw. $i_m = j_m$, also

$$v = w_{i_1} \cdots w_{i_m} c_{i_m} \cdots c_{i_1} \$ c_{i_1} \cdots c_{i_m} x_{i_m} \cdots x_{i_1}$$

  Damit ist (wegen $v \in L_1$)

$$w_{i_1} \cdots w_{i_m} = (x_{i_m} \cdots x_{i_1})^T$$

Folgerung: Der Satz gilt auch für deterministisch kontextfreie Sprachen.

Bemerkung: Da L_1, L_2 deterministisch kontextfrei sind, ist auch $(\tilde{\Sigma}^* \setminus L_1) \cup$

$(\tilde{\Sigma}^* \setminus L_2)$ kontextfrei.

Beweis: Die deterministisch kontextfreien Sprachen sind bezüglich Komplement abgeschlossen, d.h. $(\tilde{\Sigma}^* \setminus L_i)$ sind deterministisch kontextfrei. Kontextfreie Sprachen sind bezüglich Vereinigung abgeschlossen, daraus folgt die Behauptung.

SATZ: Das Entscheidungsproblem $L(G) = \Sigma_G^*$ ist für kontextfreie Grammatiken unentscheidbar.

Beweis: Es gilt: PCP $\leq (L(G) = \Sigma_G^*)$. Zu Listen (A, B) über Σ konstruiere Grammatik $\tilde{G}$ mit

$$(A, B) \text{ hat Lösung} \iff L(\tilde{G}) \neq \Sigma_{\tilde{G}}^* \qquad (\star)$$

Wie vorher: definiere zu (A, B) Grammatiken G_1, G_2 und bilde kontextfreie Grammatiken $\tilde{G}$ bezüglich

$$(\tilde{\Sigma}^* \setminus L_1) \cup (\tilde{\Sigma}^* \setminus L_2)$$

Dann gilt: (A, B) hat Lösung genau dann, wenn $L(G_1) \cap L(G_2) \neq \emptyset$, d.h. genau dann, wenn

$$L(\tilde{G}) = \Sigma^* \setminus (L(G_1) \cap L(G_2)) \neq \Sigma^*$$

Folgerung: Äquivalenzproblem für kontextfreie Grammatiken ist unentscheidbar.

Beweis: $L(G) = \Sigma^*$ ist ein Spezialfall davon, da wir für Σ^* eine zweite Grammatik G_2 mit $L(G_2) = \Sigma^*$ angeben können.

LEMMA: Für L_1 und L_2 wie oben definiert gilt:

$$L_1 \cap L_2 \text{kontextfrei} \iff L_1 \cap L_2 = \emptyset$$

„$\Leftarrow$“ klar

„$\Rightarrow$“ Sei $L_1 \cap L_2 \neq \emptyset$. Zeige: $L_1 \cap L_2$ ist nicht kontextfrei. *Annahme:* $L_1 \cap L_2$ ist kontextfrei. Wähle ein festes $z \in L_1 \cap L_2$. Somit ist $z = uv\$v^T u^T$ mit $u \in \Sigma^+$ und $v \in K^+$. Definiere

$$D = \underbrace{\underbrace{(L_1 \cap L_2)}_{\text{kontextfrei}} \cap \underbrace{(u^+ v^+ \$ (v^T)^+ (u^T)^+)}_{\text{regulär}}}_{kontextfrei}$$

Somit ist D kontextfrei. D hat (wegen Schnitt mit L_1) nur Wörter $u^m v^n \$ (v^T)^n (u^T)^m$, wegen Schnitt mit L_2 ist die Indexfolge bei den w's in u^m gleich der Indexfolge bei bei den c's in v^n, damit ist $m = n$. Daher ist

$$D = \left\{ u^n v^n \$ (v^T)^n (u^T)^n \;\middle|\; n \geq 1 \right\}$$

Mit dem Pumping-Lemma sieht man, daß D nicht kontextfrei ist, Widerspruch zu oben!

Bemerkung: Es gilt auch:

$$L_1 \cap L_2 \text{regulär} \Longleftrightarrow L_1 \cap L_2 = \emptyset$$

SATZ: Für kontextfreie Grammatiken sind folgende Probleme unentscheidbar:

1. Ist $L(G_1) \cap L(G_2)$ kontextfrei?
2. Ist $\Sigma^* \setminus L(G)$ kontextfrei?
3. Ist $L(G)$ regulär?

Beweis:

1. Transformation von PCP mit $(A, B) \mapsto (G_1, G_2)$ mit

   $$(A, B) \text{ hat Lösung} \Longleftrightarrow L(G_1) \cap L(G_2) \neq \emptyset$$

   Mit dem Lemma ist dies äquivalent zu $L(G_1) \cap L(G_2)$ nicht kontextfrei.

2. Transformation von PCP mit $(A, B) \mapsto \tilde{G}$ mit der Eigenschaft (wobei $L(\tilde{G}) = \Sigma^* \setminus (L_1 \cap L_2)$)

   $$(A, B) \text{ hat Lösung} \Longleftrightarrow L(\tilde{G}) \neq \tilde{\Sigma}^*$$

   Dies ist äquivalent dazu, daß $L_1 \cap L_2 = \tilde{\Sigma}^* \setminus L(\tilde{G})$ nicht kontextfrei ist, d.h. daß das Komplement von $L(\tilde{G})$ nicht kontextfrei ist.

3. Haben wir aus $L_1 \cap L_2 \neq \emptyset \Leftrightarrow \tilde{\Sigma}^* \setminus L(\tilde{G})$ nicht regulär, d.h. $L(\tilde{G})$ ist nicht regulär.

5 Komplexität

5.1 P und NP

DEFINITION: Sei $G = (V, E)$ ein ungerichteter Graph. $V' \subseteq V$ heißt *Clique*, falls für alle $u, v \in V'$ mit $u \neq v$ jeweils $(u, v) \in E$.

Cliquenproblem: Entscheide zu einem gegebenen Graphen $G = (V, E)$ und $k \in \mathbb{N}$, ob G eine Clique mit k Knoten besitzt.

Algorithmus: Teste für alle $V' \subseteq V$ mit $|V'| = k$, ob V' eine Clique bildet.

Zeitaufwand: Für $|V| = n$, $k = \frac{n}{2}$, n gerade: Test für jedes V' erfordert $\geq \Omega(1)$ Schritte. Es gibt $\binom{n}{k}$ viele k-elementige Teilnmengen, es gilt:

$$\binom{n}{\frac{n}{2}} = \frac{n(n-1)\cdots(\frac{n}{2}+1)}{\frac{n}{2}(\frac{n}{2}-1)\cdots 1} \geq 2^{\frac{n}{2}}$$

Also mindestens exponentielle Laufzeit in n.

Frage: Gibt es schnellere (polynomielle) Algorithmen?

Kodierung des Cliquenproblems durch Sprache:

$$\text{CLIQUE} = \left\{ u\#v \;\middle|\; \begin{array}{l} u \in \{0,1\}^* \text{ stellt Adjazenzmatrix von } G \text{ dar} \\ v \in \{0,1\}^* \text{ stellt } k \text{ dar} \\ G \text{ hat Clique mit } k \text{ Knoten} \end{array} \right\}$$

Dabei ist u ein Wort der Länge $\mathcal{O}(|V|^2)$ und v ist ein Wort der Länge $\mathcal{O}(\log |V|)$.

DEFINITION: Sei $L \subseteq \Sigma^*$ und $T\colon \mathbb{N} \to \mathbb{N}$.

- Eine NTM $\mathfrak{A}$ akzeptiert L [in nicht-deterministischer Zeit T] genau dann, wenn
  - für alle $w \in \Sigma^*$ gilt: $w \in L$ genau dann, wenn eine akzeptierende Berechnung von $\mathfrak{A}$ zu w existiert [mit Länge $\leq T(|w|)$]
- Eine DTM $\mathfrak{A}$ entscheidet L [in deterministischer Zeit T] genau dann, wenn
  - für alle $w \in \Sigma^*$ endet die Berechnung von w mit einer Stoppkonfiguration [und hat Länge $\leq T(|w|)$] und
  - der Zustand der Stoppkonfiguration ist in F genau dann, wenn $w \in L$ ist.

DEFINITION: Sei $L \subseteq \Sigma^*$ und $\Gamma \supseteq \Sigma$.

- Es sei L *in NP* genau dann, wenn eine NTM $\mathfrak{A}$ über Alphabet Γ und ein Polynom T existieren, so daß $\mathfrak{A}$ L in nicht-deterministischer Zeit T akzeptiert.
- Es sei L *in P* genau dann, wenn eine DTM $\mathfrak{A}$ über Alphabet Γ und ein Polynom T existieren, so daß $\mathfrak{A}$ L in deterministischer Zeit T akzeptiert.

DEFINITION: Es sei $\mathfrak{A}$ eine DTM über $\Gamma \supseteq \Sigma_1 \cup \Sigma_2$ und $T\colon \mathbb{N} \to \mathbb{N}$ ein Polynom. Die Funktion $f\colon \Sigma_1^* \to \Sigma_2^*$ wird *berechnet durch* $\mathfrak{A}$ *in Polynomzeit* T, falls $\mathfrak{A}$ für $u \in \Sigma_1^*$ zur Startkonfiguration $q_0 u$ eine Stoppkonfiguration qw liefert mit $w = f(u)$ und die Länge der Konfigurationsfolge (d.h. Berechnungslänge) $\leq T(|u|)$ ist. Die Funktion f heißt dann *polynomzeitberechenbar*.

Bemerkung: Wird f in Zeit T berechnet, so gilt

$$|f(u)| \leq |u| + T(|u|)$$

DEFINITION: Es seien $L_1 \subseteq \Sigma_1^*$ und $L_2 \subseteq \Sigma_2^*$. Dann sei $L_1 \leq L_2$ (d.h. L_1 *polynomiell reduzierbar auf* L_2) genau dann, wenn eine „polynomielle Transformation“ von L_1 auf L_2 existiert, d.h. es existiert eine Abbildung $f\colon \Sigma_1^* \to \Sigma_2^*$, die polynomzeitberechenbar ist mit

$$u \in L_1 \Leftrightarrow f(u) \in L_2 \;\; \forall\, u \in \Sigma_1^*$$

DEFINITION: Eine Sprache L_0 heißt *NP-vollständig* genau dann, wenn $L \in \mathsf{NP}$ und für alle $L \in \mathsf{NP}$ gilt: $L \leq L_0$

SATZ: Es sei L_0 NP-vollständig. Dann gilt

$$\mathsf{P} = \mathsf{NP} \Longleftrightarrow L_0 \in \mathsf{P}$$

Beweis:

„$\Rightarrow$“ klar

„$\Leftarrow$“ Wenn $L_0 \in P$, so entscheidet $\mathfrak{M}_0$ die Sprache L_0 in Polynomzeit T_0. Es sei $L \in \mathsf{NP}$, zu zeigen: $L \in \mathsf{P}$. Da $L \in \mathsf{NP}$ ist, ist $L \leq L_0$. Sei also $\mathfrak{M}_f$ eine Turingmaschine, die die polynomielle Transformation f von

L auf L_0 in Polynomialzeit T_f berechnet mit $u \in L \Leftrightarrow f(u) \in L_0$. Wir konstruieren $\mathfrak{M}$ aus $\mathfrak{M}_0$ und $\mathfrak{M}_f$ durch „Hintereinanderschalten“:

$$\mathfrak{M}\colon\ \text{b}u \overset{\mathfrak{M}_f}{\underset{T_f(|u|)}{\mapsto}} \text{b}f(u) \overset{\mathfrak{M}_0}{\underset{T_0(|f(u)|)}{\mapsto}} \begin{cases} \in F \\ \notin F \end{cases}$$

Damit ist

$$\begin{aligned} \mathfrak{M}\colon u \to F &\Leftrightarrow \mathfrak{M}_0\colon f(u) \to F \\ &\Leftrightarrow f(u) \in L_0 \\ &\Leftrightarrow u \in L \end{aligned}$$

Damit entscheidet $\mathfrak{M}$ L mit Gesamtlaufzeit kleinergleich $T_f(|m|) + T_0(|f(u)|) \leq T(|u|)$ für ein Polynom T (wegen $|f(u)| \leq |u| + T_f(|u|)$).

Bemerkung: Wenn $L_1 \leq L_2$ und $L_2 \leq L_3$, so ist $L_1 \leq L_3$.

Folgerung: Ist L_0 NP-vollständig und $L_0 \leq L_1$ für $L_1 \in \mathsf{NP}$, so ist auch L_1 NP-vollständig.

Beweis: Es sei $L \in \mathsf{NP}$, damit ist $L \leq L_0$, nach Voraussetzung $L_0 \leq L_1$, dann gilt $L \leq L_1$ und somit L_1 NP-vollständig

5.2 NP-vollständige Probleme

5.2.1 Erfüllbarkeitsproblem (SAT)

Ziel: Existenz eines NP-vollständigen Problems.

Definition: *Aussagenlogische Ausdrücke* bestehen aus Variablen (kodiert durch X_0, X_1, X_{10}, X_{11}, X_{100}, ...) und Operatoren $\wedge$, $\vee$, $\neg$. *Literale* sind X_i oder $\neg X_i$; aus Literalen $Y_1, \ldots, Y_k$ zusammengesetzte Formeln $(Y_1 \vee \ldots \vee Y_k)$ heißen *Klauseln*. Schließlich ist ein Ausdruck $\alpha = c_1 \wedge \ldots \wedge c_m$ mit Klauseln $c_1, \ldots, c_m$ in *konjunktiver Normalform (KNF)*.

Eine Formel α ist erfüllbar genau dann, wenn es eine Belegung der Variablen in α mit `true` oder `false`, so daß sich für α der Wert `true` ergibt.

Beispiel: $\alpha = (X_1 \vee X_{10} \vee \neg X_{11}) \wedge (X_1 \vee \neg X_{10})$ ist erfüllbar mit $X_1 = X_{10} =$ `true`.

DEFINITION: Definiere die Sprache

$$\text{SAT} = \{\alpha \text{ KNF} \mid \alpha \text{ erfüllbar}\} \subseteq \{x, 1, 0, \wedge, \vee, \neg, (,)\}^*$$

SATZ: (Satz von COOK, 1971) SAT ist NP-vollständig

Beweis: Zu zeigen:

1. SAT $\in$ NP: Rate nicht-deterministisch eine Belegung der Variablen und werte danach die Belegung aus, falls die Formel insgesamt wahr ist, gehe in Endzustand, ansonsten nicht.

2. $L \in \text{NP} \Rightarrow L \leq \text{SAT}$: Sei $\mathfrak{M}$ eine Turingmaschine, die L in nicht-deterministischer Zeit T (polynomial) akzeptiert.

   Gesucht: eine Transformation $u \mapsto \alpha_u$, die polynomzeit-berechenbar ist *und* $u \in L$ genau dann, wenn $\alpha_u \in \text{SAT}$. Dabei wird α_u so beschrieben, daß $\mathfrak{M}$ für u in *genau* $T(|u|)$ Schritten eine akzeptierende Stoppkonfiguration erreicht.

Sei nun

$$\text{SAT}(3) := \{\alpha \in \text{SAT} \mid \alpha \text{ in KNF mit } \leq 3 \text{ Literalen}\}$$

SATZ: SAT(3) ist NP-vollständig.

Beweis:

1. SAT(3) $\in$ NP: analog zu oben

2. SAT $\leq$ SAT(3): Suche eine polynomzeitberechenbare Transformation $\alpha \to \bar{\alpha}$ mit $\alpha \in \text{SAT}$ genau dann, wenn $\alpha \in \text{SAT}(3)$. *Idee:*

   $(Y_1 \vee Y_2 \vee Y_3 \vee Y_4)$ erfüllbar $\Longleftrightarrow (Y_1 \vee Y_2 \vee X) \wedge (\neg X \vee Y_3 \vee Y_4)$ erfüllbar

   Allgemein: $(Y_1 \vee Y_2 \vee \ldots \vee Y_n)$ ist erfüllbar genau dann, wenn folgende Formel erfüllbar ist:

   $$(Y_1 \vee Y_2 \vee X_1) \wedge (\neg X_1 \vee Y_3 \vee X_2) \wedge \ldots \wedge (\neg X_{i-2} \vee Y_i \vee X_{i-1}) \wedge \ldots \wedge (\neg X_{n-3} \vee Y_{n-1} \vee Y_n)$$

   Beweis:

   „$\Rightarrow$“ Es sei $(Y_1 \vee Y_2 \vee \ldots \vee Y_n)$ wahr. Sind $Y_1 \vee Y_2$ bzw. $Y_{n-1} \vee Y_n$ wahr, setze dann X_i für $i = 1, \ldots, n-3$ auf falsch bzw. auf wahr, dann ist der große Ausdruck auch wahr. Andernfalls sei z.B. Y_k wahr, dann setze X_i auf wahr für $i = 1, \ldots, k-2$ und X_i auf falsch für $i = k-1, \ldots, n-3$.

„⇐“ Es sei eine Belegung gegeben, die die rechte Seite erfüllt. Falls alle Y_i falsch sind, können nicht alle Klauseln wahr sein. Es existiert also ein wahres Y_i.

Reduziere nun SAT auf SAT(3), indem die obige Transformation α klauselweise auf $\bar{\alpha}$ abbildet, dann gilt: $|\bar{\alpha}| \leq c \cdot |\alpha|$ und die Transformation ist polynomzeit-berechenbar. Nach Konsdtruktion ist α erfüllbar genau dann, wenn $\bar{\alpha}$ erfüllbar ist. Damit ist SAT $\leq$ SAT(3).

5.2.2 Cliquenproblem

Seien ein ungerichteter Graph G und eine Zahl m gegeben, gewünschte Ausgabe: *ja*, falls eine m-Clique in G existiert (vollständiger Teilgraph mit m Knoten) **Beweis:**

1. CLIQUE $\in$ NP: wähle nicht-deterministisch (rate) m Knoten von G und teste, ob jeder Knoten mit jedem anderen durch eine Kante verbunden ist

2. SAT $\leq$ CLIQUE: Sei F ein Boolescher Ausdruck in KNF, wobei $F = F_1 \wedge \ldots \wedge F_m$ (wobei F_i Klausel mit k_i Literalen, d.h. $F_i = (Y_{i1} \vee \ldots \vee Y_{ik_i})$) Konstruktion von $G = (V, E)$:

$$\begin{aligned} V &= \{\, [i,j] \mid 1 \leq i \leq m, 1 \leq j \leq k_i \} \\ E &= \big\{\, ([i,j],[k,l]) \mid (i \neq k) \wedge (Y_{ij} \neq \bar{Y}_{kl}) \big\} \end{aligned}$$

Zeige: F ist erfüllbar genau dann, wenn G eine m-Clique enthält.

„⇒“ F ist erfüllbar, dann existiert eine Belegung φ mit $\varphi(F) = \texttt{true}$, dann existiert eine Belegung φ mit $\varphi(F_i) = \texttt{true}$ für alle $i = 1, \ldots, m$, damit existiert φ, so daß für alle $1 \leq i \leq m$ ein $r_i \in \{1, .., k_i\}$ existiert mit $\varphi(Y_{ir_i}) = \texttt{true}$. Definiere nun:

$$C = \{\, [i, r_i] \mid 1 \leq i \leq m \}$$

Zeige: C ist eine m-Clique. *Angenommen,* $([i,r_i],[j,r_j]) \notin E$ für ein Paar $i \neq j$. Dann ist $Y_{ir_i} = \bar{Y}_{jr_j}$. Dagegen gilt aber $\varphi(Y_{ir_i}) = \varphi(Y_{jr_j}) = \texttt{true}$, Widerspruch!

„⇐“ Sei C eine m-Clique in G, dann ist für gewisse $r_1, \ldots, r_m$:

$$C = \{[1, r_1], [2, r_2], \ldots, [m, r_m]\}$$

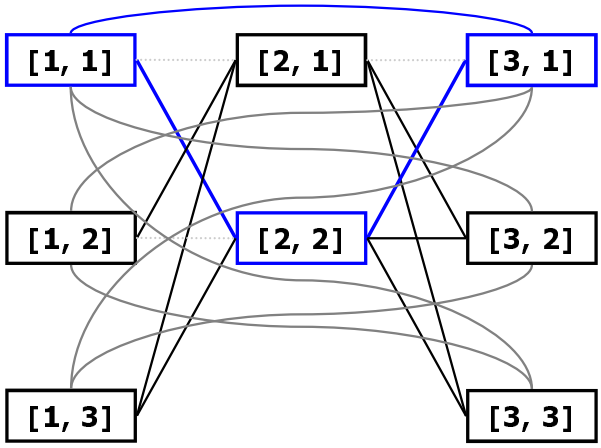
Definiere eine Belegung φ mit $\varphi(Y_{1r_1}) = \varphi(Y_{2r_2}) = \ldots = \varphi(Y_{mr_m}) = \texttt{true}$. Das geht widerspruchsfrei, da $Y_{ir_i} \neq \bar{Y}_{jr_j}$ für alle $1 \leq i, j \leq m$.

Dann ist $\varphi(F_i) = \texttt{true}$ für alle $1 \leq i \leq m$. Erweitere φ eventuell noch auf fehlende Variablen (beliebige Belegung). Das ergibt eine korrekte Belegung φ mit $\varphi(F) = \texttt{true}$.

Beispiel: Zu

$$(X_1 \vee X_2 \vee X_3) \wedge (\bar{X}_1 \vee \bar{X}_2) \wedge (X_1 \vee \bar{X}_2 \vee \bar{X}_3)$$

erfüllbar durch $\varphi(X_1) = \texttt{true}$, $\varphi(X_2) = \texttt{false}$ und $\varphi(X_3) = \texttt{true}$. Dann ist $G = (V, E)$:



5.2.3 Vertex-Cover

DEFINITION: $V' \subseteq V$ ist ein Vertex-Cover in einem Graphen $G = (V, E)$, falls jede Kante aus E mit mindestens einem Knoten aus V' inzident ist.

Ziel: Vertex-Cover mit minimaler Kardinalität finden, als Entscheidungsproblem formuliert:

- Eingabe: ein ungerichteter Graph $G = (V, E)$ und eine Zahl $m \in \{1, \ldots, |V|\}$
- Ausgabe: *ja*, falls G ein Vertex-Cover der Größe m hat und *nein* sonst

SATZ: VertexCover ist NP-vollständig.

Beweis:

1. VertexCover $\in$ NP ist klar (Knoten raten und Eigenschaft überprüfen)

2. Wir zeigen CLIQUE $\leq$ VertexCover: Zu $G = (V, E)$ konstruiere $\bar{G} = (V, \bar{E})$ mit

$$\bar{E} = \{(v, w) \mid v, w \in V; v \neq w; (v, w) \notin E\}$$

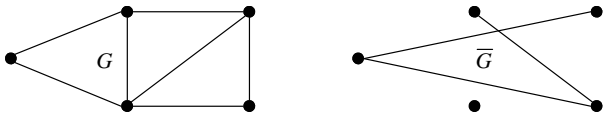
Zeige: C ist Clique in G genau dann, wenn $V \setminus C$ Vertex-Cover in $\bar{G}$ ist.

„⇒“ C ist Clique in G, dann gibt es keine Kanten zwischen den Knoten von C in $\bar{G}$, d.h. jede Kante in $\bar{E}$ ist inzident mit mindestens einem Knoten in $V \setminus C$, somit ist $V \setminus C$ Vertex-Cover in $\bar{G}$.

„⇐“ Sei $V \setminus C$ Vertex-Cover in $\bar{G}$, dann ist nach Definition jede Kante in $\bar{E}$ mit mindestens einem Knoten aus $V \setminus C$ inzident; d.h. keine Kante in $\bar{G}$ verbindet zwei Kanten aus C, damit ist C Clique in G.

Reduktion: Es wird CLIQUE mit $G = (V, E)$ und k in polynomieller Zeit transformiert auf VertexCover mit $\bar{G} = (V, \bar{E})$ und $|V| - k$.

Beispiel:



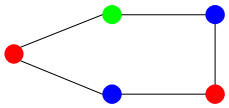
5.2.4 Färbbarkeit

Betrachte das Entscheidungsproblem $k -$ COLOR:

- Eingabe: ein ungerichteter Graph $G = (V, E)$ und eine Zahl $k \in \{1, \ldots, |V|\}$
- Ausgabe *ja*, falls G mit k Farben gefärbt werden kann (d.h. je zwei adjazente Knoten $v \neq v'$ mit $(v, v') \in E$ haben unterschiedliche Farben), *nein* sonst

Man bezeichnet $\chi(G)$ als *chromatische Zahl* (d.h. die minimale Anzahl von Farben, um G zu färben).

Beispiel: Der folgende Graph hat $\chi(G_5) = 3$:



SATZ: $k -$ COLOR ist NP-vollständig.

Beweis:

1. $k-\text{COLOR} \in \mathsf{NP}$ ist klar

2. $3-\text{SAT} \leq k-\text{COLOR}$: Sei $F = F_1 \wedge \ldots \wedge F_m$ ein Boolescher Ausdruck in KNF mit Variablen $X_1, \ldots, X_n$ und jedes F_i habe Länge ≤ 3. Konstruiere nun $G = (V, E)$ mit $|V| = 3n + m + 1$ so, daß

$$\begin{aligned} V &= \left\{ X_i, \bar{X}_i, V_i \;\middle|\; i = 1, \ldots, n \right\} \cup \{ F_j \mid j = 1, \ldots, m \} \cup \{Z\} \\ E &= \left\{ (V_i, V_j), (V_i, X_j), (V_i, \bar{X}_j) \;\middle|\; i \neq j \right\} \cup \left\{ (X_i, \bar{X}_i) \;\middle|\; i = 1, \ldots, n \right\} \\ &\quad \cup \{ (X_i, F_j) \mid X_i \notin F_j \} \cup \left\{ (\bar{X}_i, F_j) \;\middle|\; \bar{X}_i \notin F_j \right\} \\ &\quad \cup \{ (V_i, Z) \mid i = 1, \ldots, n \} \cup \{ (F_j, Z) \mid j = 1, \ldots, m \} \end{aligned}$$

   Behauptung: F ist erfüllbar genau dann, wenn G mit $n + 1$ Farben gefärbt werden kann

   „$\Rightarrow$“ Es existiert eine Belegung der Variablen φ, so daß für alle j gilt: F_j enthält ein Literal Y mit $\varphi(Y) = \texttt{true}$. Färbe nun $V_1, \ldots, V_n, Z$ mit den $1, \ldots, n, n+1$ und färbe weiter

$$X_i \text{ mit } \begin{cases} i & \text{falls} \quad \varphi(X_i) = \texttt{true} \\ n+1 & \text{sonst} \end{cases}$$

   und $\bar{X}_i$ entsprechend anders in $i, n{+}1$. Dann sind $\left\{ X_i, \bar{X}_i, V_i \;\middle|\; i = 1, \ldots, n \right\}$ und z korrekt mit $n + 1$ Farben gefärbt. Weiter ist F_j verbunden mit X_i ($\bar{X}_i$), falls X_i ($\bar{X}_i$) kein Literal in F_j ist. Für ein Literal $Y \in \{X_i, \bar{X}_i\}$ in F_j gilt: $\varphi(Y) = \texttt{true}$ und $(Y, F_j) \notin E$. Färbe F_j mit Farbe i.

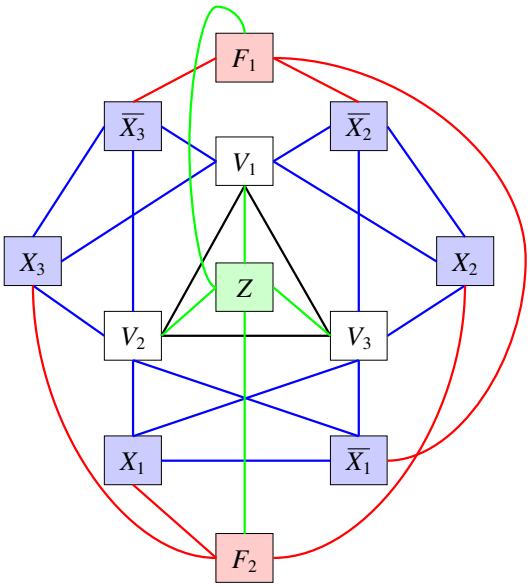
   „$\Leftarrow$“ Sei F mit Klauseln F_j gegeben, konstruiere den Graphen G, dieser sei mit $n + 1$ Farben gefärbt. O.B.d.A seien $V_1, \ldots, V_n, Z$ mit $1, \ldots, n + 1$ gefärbt (sie bilden nach Konstruktion eine Clique). Weiter sind damit X_i und $\bar{X}_i$ mit Farben aus i und $n + 1$ gefärbt. Betrachte

$$\varphi(X_i) = \begin{cases} \texttt{true} & \text{falls} \quad X_i \text{ hat Farbe } i \\ \texttt{false} & \text{sonst} \end{cases}$$

   Setze entsprechend $\varphi(\bar{X}_i) = \neg\varphi(X_i)$. F_j hat höchstens drei Literale, F_j ist also mit mindestens $2n{-}3$ Knoten aus $\left\{ X_i, \bar{X}_i \;\middle|\; i = 1, \ldots, n \right\}$ verbunden.

   Annahme: alle drei Literale in F_j sind mit $n + 1$ gefärbt (d.h. $\varphi(F_j) = \texttt{false}$), dann müßte F_j mit $n + 1$ gefärbt werden, aber Z ist schon mit $n + 1$ gefärbt, Widerspruch zur richtigen Färbung! Also ist in jeder Klausel mindestens ein Literal nicht mit $n + 1$ gefärbt und damit $\texttt{true}$. Somit ist F erfüllbar!

Beispiel: Sei $F = (X_1 \vee X_2 \vee X_3) \wedge (\bar{X}_1 \vee \bar{X}_2 \vee \bar{X}_3)$, dann ist $n = 3$ und $m = 2$. Der entsprechende Graph ist[12]:



5.2.5 Hamilton-Kreis

Betrachte das folgende Entscheidungsproblem *Hamilton-Kreis* (HC):

- Eingabe: ungerichteter Graph $G = (V, E)$
- Ausgabe: *ja*, falls G einen *Hamilton-Kreis* enthält, d.h. ein geschlossener Kantenzug, der jeden Knoten genau einmal besucht.

SATZ: HC ist NP-vollständig.

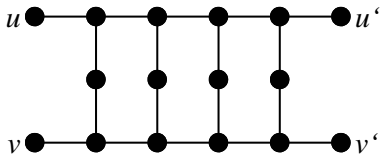
Beweis: HC $\in$ NP ist klar: rate Permutation und prüfe die Eigenschaft nach. Wir zeigen nun: $3 - \text{SAT}' \leq \text{HC}$ mit $3 - \text{SAT}'$ die Menge der Booleschen Ausdrücke mit genau drei Literalen pro Klausel.

Gegeben sei ein Boolescher Ausdruck $F = F_1 \wedge \ldots \wedge F_m$ mit m Klauseln in n Variablen $X_1, \ldots, X_n$ in KNF und mit genau drei Literalen pro Klausel. Konstruiere zu F einen Graphen G_F mit F erfüllbar genau dann, wenn G_F einen Hamilton-Kreis enthält.

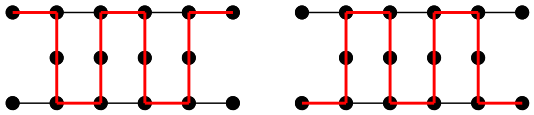
[12] die Farben der Knoten und Kanten entsprechen nicht der gesuchten Färbung!

Es werden folgende *Spezialkomponenten (Gadgets)* verwendet:

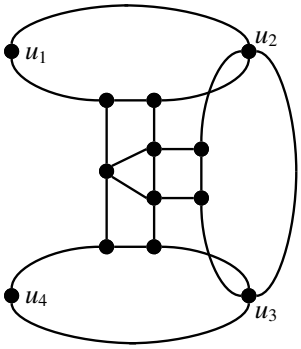
- Komponente A: (andere Kanten sind höchstens mit u, v, u' oder v' verbunden)



  Es gibt nur zwei Wege, diese Struktur von links nach rechts zu durchlaufen und dabei alle inneren Knoten zu besuchen:

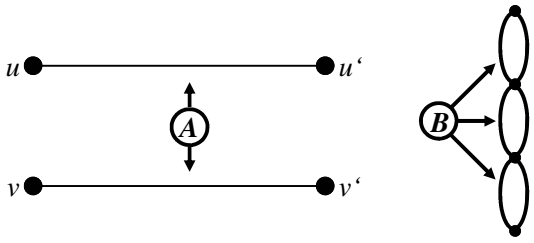


- Komponente B (andere Kanten können höchstens zu $u_1, \ldots, u_4$ inzident sein):



  Falls G einen Hamilton-Kreis enthält, dann kann der Kreis nicht alle drei Kanten (u_1, u_2), (u_2, u_3) und (u_3, u_4) durchlaufen. Jede echte Teilmenge dieser drei Kanten kann aber vorkommen.

- Kurzschreibweise für beiden Komponenten:



Konstruktion von G_F aus F:

- Den m Klauseln $F_1, \ldots, F_m$ entsprechen m Kopien von B „in Serie“ geschaltet. Jeder Variablen X_i entsprechen zwei Knoten v_i, w_i und zwei Kopien der Kante (v_i, w_i) (linke, rechte Kante).
- Weitere Kanten: (w_i, v_{i+1}) für $i = 1, \ldots, n-1$ und (u_{11}, v_1) sowie (u_{m4}, w_n), wobei u_{ij} die i-te Kopie von u_j im i-ten B-Graphen ist.
- Die Struktur der Klauseln wird durch die A-Graphen simuliert: Verbinde die Kante (u_{ij}, u_{ij+1}) mit linken (rechten) Kopie von (v_k, w_k) durch einen A-Graphen, falls X_k ($\bar{X}_k$) das j-te Literal in F_i ist.

Zeige: G_F hat einen Hamilton-Kreis genau dann, wenn F erfüllbar ist.

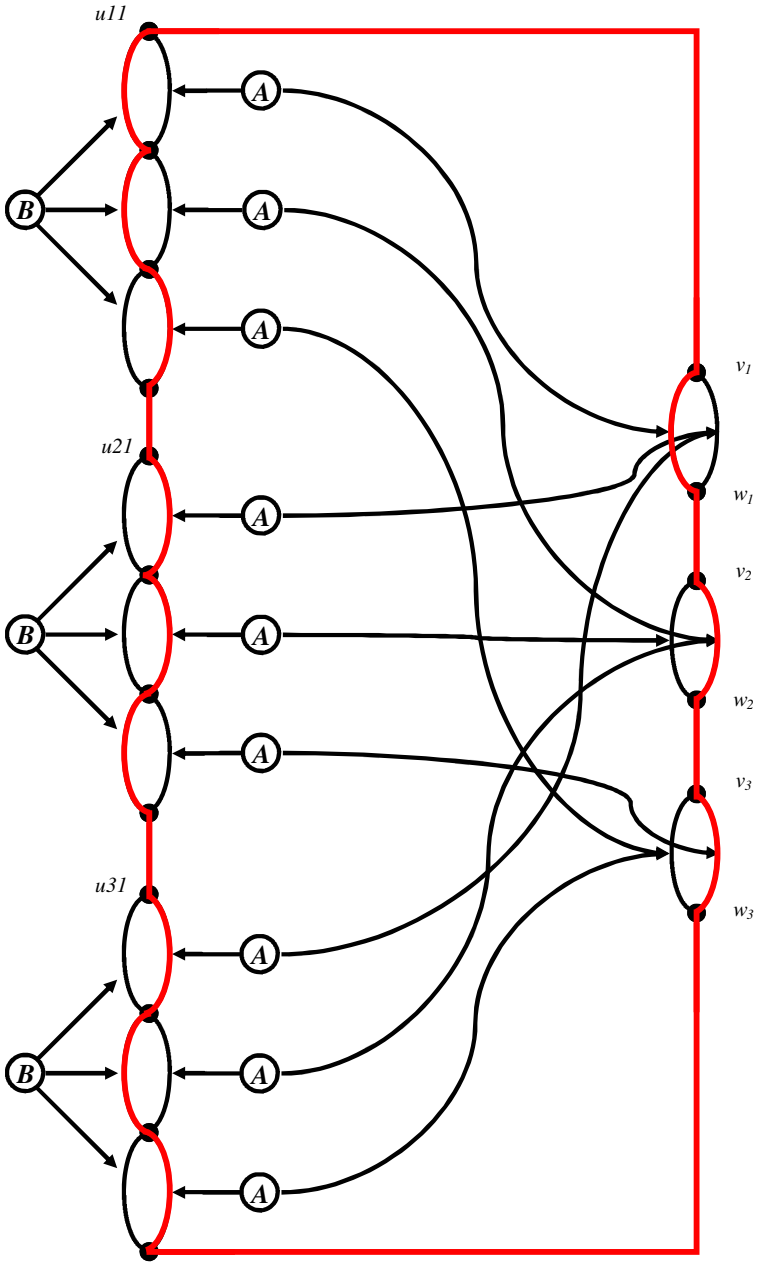
„$\Rightarrow$“ Sei K ein Hamilton-Kreis von G_F. Dann gilt wegen der A-Graphen: K muß (u_{11}, v_1) durchlaufen, dann alle v- und w-Knoten von oben nach unten, wobei eine der Kopien von (v_i, w_i) gewählt wird und dann (w_u, u_{m4}) und alle Kopien von B von unten nach oben. Die Wahl der Kopie von (v_i, w_i) entspricht dem Setzen von X_i auf `true` bzw. `false`. Wegen der A-Graphen werden die Kanten (u_{ij}, u_{ij+1}) für eine Klausel F_i genau dann durchlaufen, wenn die zugehörige Kopie von (v_k, w_k) nicht durchlaufen wird (d.h. das zugehörige Literal falsch ist). Wegen der Konstruktion der B-Graphen wird eine der Kanten (u_{ij}, u_{ij+1}) für $j = 1, 2, 3$ nicht durchlaufen (d.h. das zugehörige Literal erfüllt F_i). Damit ist F erfüllbar.

„$\Leftarrow$“ Sei F erfüllbar durch Belegung φ. Durchlaufe dann die linke (rechte) Kopie von (v_i, w_i), falls $\varphi(X_i) =$ `true` (= `false`) und durchlaufe (u_{ij}, u_{ij+1}) genau dann, wenn das j-te Literal von F_i bezüglich φ falsch ist (Bedingung von A!). Da $\varphi(F_i) =$ `true`, ist mindestens ein Literal in F_i wahr (d.h. nicht alle drei Kanten (u_{ij}, u_{ij+1}) von B werden durchlaufen) Dies ergibt dann den Hamilton-Kreis in G_F.

Beispiel: Betrachte folgende Formel:

$$F = (X_1 \vee \bar{X}_2 \vee X_3) \wedge (\bar{X}_1 \vee X_2 \vee \bar{X}_3) \wedge (\bar{X}_1 \vee \bar{X}_2 \vee X_3)$$

Diese ist erfüllbar z.B. für $\varphi(X_1) = \texttt{true}$, $\varphi(X_2) = \texttt{false}$, $\varphi(X_3) = \texttt{false}$. Dann wären Graph und Hamilton-Kreis folgendermaßen:



6 Korrektheit

6.1 Das Hoare-Kalkül

6.1.1 Partielle Korrektheit

Beispiel: Betrachte folgendes Programm für den ggT:

P_{ggT} $\{X_1 > 0 \ \wedge \ X_2 > 0\}$
$Y := X_1; Z := X_2;$
$\{Y > 0 \ \wedge \ Z > 0 \ \wedge \ \text{ggT}(X_1, X_2) = \text{ggT}(Y, Z)\}$
`while` $Y \neq Z$ `do begin`
  `if` $Y < Z$ `then`
    $Z := Z - Y;$
  `else`
    $Y := Y - Z;$
  $\{Y > 0 \ \wedge \ Z > 0 \ \wedge \ \text{ggT}(X_1, X_2) = \text{ggT}(Y, Z)\}$
`end`
$\{Y = Z \ \wedge \ \text{ggT}(X_1, X_2) = \text{ggT}(Y, Z)\}$
$\{Y = \text{ggT}(X_1, X_2)\}$

Lemma: über den ggT:

1. Für $y > 0$ gilt $\text{ggT}(y, Y) = y$.
2. Für $y > z > 0$ ist $\text{ggT}(y, z) = \text{ggT}(y - z, z)$.
3. Für $z > y > 0$ gilt $\text{ggT}(y, z) = \text{ggT}(y, z - y)$.

Die Zusicherungen im ggT-Programm gelten also. Getrennt weisen wir die Termination nach: Die Laufzeitschranke $t\colon \mathbb{N}^2 \to \mathbb{N}$ sei definiert durch $t(y, z) = \max(y, z)$. Es gilt:

1. Es gilt: $t(y, Z) > 0$ für $y, z \in \mathbb{N}$.
2. Falls $y \neq z$ ist, wird $t(y, z)$ im `while`-Schleifenkörper echt vermindert.

Damit terminiert das ggT-Programm.

Zur **Programmkorrektheit** spielen drei Sprachen eine Rolle: die Programmiersprache, die Zusicherungssprache und die Korrektheitssprache:

1. *Programmiersprache* — hier betrachten wir die `WHILE`-Programme; $\mathcal{P}_n$ sei die Menge der `WHILE`-Programme mit höchstens den Variablen $X_1, \ldots, X_n$.

2. *Zusicherungssprache* — hier die Sprache der Prädikatenlogik erster Stufe mit folgender Signatur:

   - Gleichheit $=$
   - Konstanten 0 1 2 ...
   - Funktionssymbole $+ \; - \; * \;$ ggT kgV !
   - Relationssymbole $\neq < > \leq \geq$ even odd

   Alle Symbole seien jeweils mit der Interpretation über $\mathbb{N}$ gegeben in der Struktur $\mathcal{N} = (\mathbb{N}, 0^{\mathbb{N}}, 1^{\mathbb{N}}, \ldots, +^{\mathbb{N}}, -^{\mathbb{N}}, \ldots)$. Setze diese nun zusammen:

   - *Terme* seien aus Konstanten, Variablen wie X_1 oder X_2 und Funktionssymbolen aufgebaut.
   - *Formeln* entstehen aus atomaren Formeln (Gleichungen $t_1 = t_2$ oder Relationsausdrücken $t_1 < t_2$) verknüpft durch $\neq, \wedge, \vee, \rightarrow, \leftrightarrow, \forall, \exists$.

     Formeln geben wir durch φ, ψ, χ wieder; $\varphi(X_1, \ldots, X_n)$ bedeutet, daß höchstens $X_1, \ldots, X_n$ in φ frei vorkommen. In üblicher Weise wird die Gültigkeitsbeziehung $(\mathcal{N}, k_1, \ldots, k_n) \models \varphi(X_1, \ldots, X_n)$ benutzt.

3. *Korrektheitssprache* — diese besteht aus den HOARE-Formeln der Form $\{\varphi\}\, P\, \{\psi\}$, wobei φ die *Vorbedingung* ist, P das `WHILE`-Programm und ψ die *Nachbedingung*.

   Weiter deutet die Formel $\{\varphi(X_1, \ldots, X_n)\}\, P\, \{\psi(X_1, \ldots, X_n)\}$ an, daß die Variablen $X_1, \ldots, X_n$ höchstens frei in φ, ψ sind und $P \in \mathcal{P}_n$.

   Semantik der HOARE-Formeln: Eine Formel

   $$\{\varphi(X_1, \ldots, X_n)\}\, P\, \{\psi(X_1, \ldots, X_n)\}$$

   trifft zu, falls folgendes zutrifft:

   Für $\bar{k} = (k_1, \ldots, k_n)$ und $\bar{l} = (l_1, \ldots, l_n)$ gilt:

   $$\begin{array}{rl} & (\mathcal{N}, k_1, \ldots, k_n) \models \varphi(X_1, \ldots, X_n) \\ \wedge & [P](k_1, \ldots, k_n) = (l_1, \ldots, l_n) \\ \Longrightarrow & (\mathcal{N}, l_1, \ldots, l_n) \models \psi(X_1, \ldots, X_n) \end{array}$$

   Kurz: Gilt φ vor der Ausführung von P und terminiert P, so gilt ψ nach der Ausführung von P.

Beispiele:

- Im obigen Beispiel ggT gilt:

$$\{X_1 > 0 \ \wedge \ X_2 > 0\}\ P_{\text{ggT}}\ \{Y = \text{ggT}(X_1, X_2)\}$$

- $\{X_1 = 7\}\ X_1 := X_1 + 1\ \{X_1 = 8\}$ gilt
- $\{X_1 \leq 7\}\ X_1 := X_1 + 1\ \{X_1 = 8\}$ gilt nicht
- $\{X_1 = 7\}\ X_1 := X_1 + 1\ \{X_1 \leq 8\}$ gilt
- $\{X_1 < X_1\}\ X_1 := X_1 + 1\ \{X_1 = 8\}$ gilt
- $\{X_1 = 1\}$ `while` $X_1 > 0$ `do` $X_1 := X_1 + 1\ \{X_1 = 0\}$ gilt (!)

HOARE-Formeln behaupten die *partielle Korrektheit*, d.h. unter der Annahme der Termination. *Totale Korrektheit* behauptet zusätzlich Termination zur partiellen Korrektheit.

Der Nachweis der Korrektheit eines Programms bedeutet: Beweis einer HOARE-Formel $\{\varphi\}\ P\ \{\psi\}$ gegebenenfalls mit gesondertem Terminationsbeweis (Aufgaben der *Programmverifikation*). Die Beweis-Methode folgt dem Programmaufbau: Führe den Nachweis passender Hoare-Formeln für Teilprogramme Q von P; ausgehend von den Wertzuweisungen bis hin zum Gesamtprogramm P.

DEFINITION: Ein `WHILE`-Programm P heißt *korrekt kommentiert*, wenn es durch Vorbedingung, Nachbedingung und weitere Zusicherungen so ergänzt ist, daß

1. jede darin auftretende HOARE-Formel $\{\varphi\}\ Q\ \{\psi\}$ gilt für ein Teilprogramm Q und Zusicherungen φ, ψ direkt vor und nach Q
2. Treten $\varphi(X_1, \ldots, X_n)$ und $\psi(X_1, \ldots, X_n)$ direkt hintereinander auf, so gilt:

$$\mathcal{N} \vdash \forall X_1, \ldots, X_n\ (\varphi(X_1, \ldots, X_n) \Rightarrow \psi(X_1, \ldots, X_n))$$

Beispiel: Das ggT-Programm ist korrekt kommentiert.

Wir können die Verifikationsaufgabe auch aus rein syntaktische Regeln stützen.

Ein solcher formaler Aufbau ist möglich mit den sogenannten *Hoare-Regeln*: Eine Hoare-Regel hat die Form

$$\frac{H_1, \ldots, H_k}{H}$$

mit Hoare-Formel H und Hoare-Formeln bzw. Zusicherungen $H_1, \ldots, H_k$. Die Regel heiß korrekt, falls sie von gültigen (Hoare-)Formeln immer zu einer gültigen Formel[13] führt. **Beispiel:** Kompositionsregel:

$$\frac{\{\varphi\}\ P_1\ \{\psi\}, \{\psi\}\ P_2\ \{\chi\}}{\{\varphi\}\ P_1; P_2\ \{\chi\}}$$

Beweis: Korrektheit der Regel: Betrachte $\varphi(X_1, \ldots, X_n)$ und $\psi(X_1, \ldots, X_n)$ mit $P_1, P_2 \in \mathcal{P}_n$. Angenommen, es gilt:

1. $\forall \bar{k}, \bar{l}\ ((\mathcal{N}, \bar{k}) \models \varphi(X_1, \ldots, X_n) \wedge [P_1](\bar{k}) = \bar{l} \Rightarrow (\mathcal{N}, \bar{l}) \models \psi(X_1, \ldots, X_n))$
2. $\forall \bar{k}, \bar{l}\ ((\mathcal{N}, \bar{k}) \models \psi(X_1, \ldots, X_n) \wedge [P_2](\bar{k}) = \bar{l} \Rightarrow (\mathcal{N}, \bar{l}) \models \chi(X_1, \ldots, X_n))$

Dann gilt insgesamt: $[P_1](\bar{k}) = \bar{k}'$ für ein geeignetes $\bar{k}'$ und $[P_2](\bar{k}') = \bar{l}$. Nun gilt $(\mathcal{N}, \bar{k}') \models \varphi(X_1, \ldots, X_n)$, nach der zweiten Voraussetzung gilt (mit $\bar{k}'$ als $\bar{k}$): $(\mathcal{N}, \bar{l}) \models \chi(X_1, \ldots, X_n)$.

Wertzuweisung	$\dfrac{\varphi \to \psi\left(\frac{X}{t}\right)}{\{\varphi\}\ X := t\ \{\psi\}}$
Kompositionsregel	$\dfrac{\{\varphi\}\ P_1\ \{\psi\}, \{\psi\}\ P_2\ \{\chi\}}{\{\varphi\}\ P_1; P_2\ \{\chi\}}$
`if-then-else`-Regel	$\dfrac{\{\varphi \wedge \beta\}\ P_1\ \{\psi\}, \{\psi \wedge \neg\beta\}\ P_2\ \{\psi\}}{\{\varphi\}\ \texttt{if}\ \beta\ \texttt{then}\ P_1\ \texttt{else}\ P_2\ \{\psi\}}$
`if-then`-Regel	$\dfrac{\{\varphi \wedge \beta\}\ P_1\ \{\psi\}, \psi \wedge \neg\beta \to \psi}{\{\varphi\}\ \texttt{if}\ \beta\ \texttt{then}\ P_1\ \{\psi\}}$
`while`-Regel	$\dfrac{\{\varphi \wedge \beta\}\ P\ \{\varphi\}}{\{\varphi\}\ \texttt{while}\ \beta\ \texttt{do}\ P\{\varphi \wedge \neg\beta\}}$
Konsequenzregeln	$\dfrac{\{\varphi\}P\{\psi\}, \psi \to \chi}{\{\varphi\}\ P\{\chi\}}$ $\dfrac{\varphi \to \psi\{\psi\}P\{\chi\}}{\{\varphi\}\ P\{\chi\}}$

[13] eine Formel $\varphi(X_1, \ldots, X_n)$ heiße gültig, falls $\mathcal{N} \models \forall X_1, \ldots, X_n\ \varphi(X_1, \ldots, X_n)$

Der Korrektheitsbeweis ist in allen Fällen relativ einfach.

6.1.2 Termination

Die Standardmethode zum **Terminationsbeweis** verwendet den Begriff der Wohlordnung:

DEFINITION: Eine lineare Ordnung $(A, <)$ heißt *Wohlordnung*, wenn es keine absteigende Kette unendlicher Länge in A gibt.

Beispiele:

1. $(\mathbb{N}, <)$ ist eine Wohlordnung (bricht bei 0 ab)
2. $(\mathbb{N}^2, <)$ mit $(i, j) < (i', j') :\Leftrightarrow (i < i') \vee (i = i' \wedge j < j')$ ist eine Wohlordnung
3. $(\mathbb{Z}, <)$ ist keine Wohlordnung

LEMMA: (Terminationslemme für `WHILE`-Schleifen)

(1) Ein Programm $P \in \mathcal{P}_n$ terminiere für alle Eingaben $\bar{k} \in \mathbb{N}^n$ mit $(\mathcal{N}, \bar{k}) \models (\varphi \wedge \beta)(\bar{X})$

(2) Es gebe eine Wohlordnung $(A, <)$ und eine Funktion $t\colon \mathbb{N}^n \to A$ (Laufzeitschranke) mit

$$\{\varphi \wedge \beta \wedge t(\bar{k}) = a\}\ P\ \{\varphi \wedge t([P](\bar{k})) < a\}$$

Dann gilt: Für alle $\bar{k} \in \mathbb{N}^n$ mit $(\mathcal{N}, \bar{k}) \models \varphi(\bar{X})$ terminiert

$$\texttt{while}\ \beta\ \texttt{do}\ P$$

Beweis: Es gelte $(\mathcal{N}, \bar{k}) \models \varphi(\bar{X})$ für ein $\bar{k} \in \mathbb{N}^n$. Falls $(\mathcal{N}, \bar{k}) \models \neg\beta(\bar{X})$, dann terminiert die `WHILE`-Schleife sofort. Sei also $(\mathcal{N}, \bar{k}) \models \beta(\bar{X})$. *Angenommen,* die `WHILE`-Schleife terminiert nicht. Dann ist wegen (1) für alle $i \geq 0$ $[P]^i(\bar{k})$ definiert, d.h. $[P]^i(\bar{k}) = \bar{k}_i$ für geeignetes $\bar{k}_i$ und $(\mathcal{N}, \bar{k}_i) \models (\varphi \wedge \beta)(\bar{X})$. Wegen (2) gibt es eine unendlich absteigende Kette $t(\bar{k}) = t(\bar{k}_1) > t(\bar{k}_2) > \ldots$, das ist ein Widerspruch zur Wohlordnung. Also terminiert die `WHILE`-Schleife.

6.1.3 Beispiel

Basis zur Verifikation ist folgendes Schema der korrekten Kommentierung:

Vorbedingung	$\{\varphi_0\}$	
	Initialisierung	
	$\{\varphi_1\}$	
Invariante	$\{\varphi\}$	Laufzeitschranke $t\colon \mathbb{N}^k \to \mathbb{N}$
	`while` β `do begin`	
	$\quad\{\varphi \ \wedge \ \beta\}$	
	$\quad P$	
	$\quad\{\varphi\}$	
	`end`	
	$\{\varphi \ \wedge \ \neg\beta\}$	
Nachbedingung	$\{\psi\}$	

Checkliste für die `WHILE`-Schleife:

1. Zeige, daß die Invariante φ vor Erreichen der `WHILE`-Schleife erfüllt ist.
2. Zeige, daß $\{\varphi \ \wedge \ \beta\}\ P_0\ \{\varphi\}$ gilt.
3. Zeige, daß $\varphi \ \wedge \ \neg\beta \to \psi$ gilt.
4. Zeige, daß im Falle $\varphi \wedge \beta$ der t-Wert durch P_0 echt verkleinert wird.

Beispiel:

Vorbedingung	$\{X_1 \geq 0 \ \wedge \ X_2 \geq 0\}$	
Initialisierung	$u := X_1;\ v := X_2;\ w := 0;$	
Invariante	$\{V \geq 0 \ \wedge \ w + u \cdot v = X_1 \cdot X_2\}$	Laufzeitschranke $t = V$
	`while` V $>$ 0 `do begin`	
	$\quad\{\varphi \geq 0 \ \wedge \ w + u \cdot v = X_1 \cdot X_2 \ \wedge \ V > 0\}$	
	$\quad$ `if` odd(V) `then`	
	$\quad\quad$ `begin` $v := v - 1;\ w := w + u$ `end`	
	$\quad$ `else`	
	$\quad\quad$ `begin` $v :=$ v `div` 2; $u := u * 2$ `end`;	
	`end`;	
	$\{V = 0 \ \wedge \ V \geq 0 \ \wedge \ w + u \cdot v = X_1 \cdot X_2\}$	
Nachbedingung	$\{w = X_1 \cdot X_2\}$	

Sei beispielsweise $X_1 = 3$ und $X_2 = 5$, dann ist

- $u = 3,\ v = 5,\ w = 0$
- $u = 3,\ v = 4,\ w = 3$ — es gilt $3 + 3 \cdot 4 = 15$ ✓
- $u = 6,\ v = 2,\ w = 3$ — es gilt $3 + 2 \cdot 6 = 15$ ✓

- $u = 12$, $v = 1$, $w = 3$ — es gilt $3 + 12 \cdot 1 = 15$ ✓
- $u = 12$, $v = 0$, $w = 15$ — es gilt $15 + 0 \cdot 12 = 15$ ✓

Die einzeln Schritte der Checkliste sind nun leicht nachzuweisen.

6.1.4 Tragweite des HOARE-Kalküls

DEFINITION: Eine Hoare-Formel $\{\varphi\}\ P\ \{\psi\}$ heißt *herleitbar* relativ zur Arithmetik, wenn man sie ausgehend von geltenden Formeln der Zusicherungssprache mit den HOARE-Regeln herleiten kann.

SATZ: Adäquatheitssatz für den HOARE-Kalkül: Für die Zusicherungssprache der Arithmetik erster Stufe und für `WHILE`-Programme gilt: Eine HOARE-Formel $\{\varphi\}\ P\ \{\psi\}$ gilt genau dann, wenn sie relativ zur Arithmetik im HOARE-Kalkül herleitbar ist.

„$\Leftarrow$“ Korrektheit des HOARE-Kalüls

„$\Rightarrow$“ Vollständigkeit des HOARE-Kalküls

Frage: Kann man genau die geltenden Aussagen der Arithmetik durch einen Kalkül herleiten?

SATZ: Die Menge der geltenden HOARE-Formeln ist nicht aufzählbar.

Beweisidee: Sei $P \in \mathcal{P}_m$: Die HOARE-Formel

$$\{X_1 = 0 \ \wedge\ \ldots\ \wedge\ X_m = 0\}\ P\ \{0 = 1\}$$

gilt genau dann, wenn P für die Eingabe 0 nicht stoppt. *Angenommen*, die Menge der geltenden HOARE-Formeln sei aufzählbar. Dann ist auch die Menge der HOARE-Formeln der obigen Form aufzählbar. Damit ist sowohl die Menge der `WHILE`-Programme, die bei Eingabe 0 stoppen, als auch die nicht stoppenden Programme aufzählbar. Damit ist entscheidbar, ob ein gegebenes Programm P auf Eingabe 0 stoppt, ein Widerspruch zur Unentscheidbarkeit des Halteproblems!

6.1.5 Gödelscher Unvollständigkeitssatz

SATZ: Es gibt keinen Beweiskalkül, der genau die geltenden Formeln der Arithmetik erster Stufe herzuleiten gestattet.

Beweisidee: *Annahme*, es gibt einen Beweiskalkül. Dieser zusammen mit dem HOARE-Kalkül ergäbe nach dem Adäquatheitssatz ein Gesamtkalkül zur Herleitung der geltenden Formeln HOARE-Formeln. Dies liefert einen Aufzählungsalgorithmus für die geltenden HOARE-Formeln, im Widerspruch zum vorangehenden Satz.

6.1.6 Historische Situation

- A. TURING: „Checking a large Routine“ (1950)
- R. FLOYD: Zusicherungen in `GOTO`-Programmen (1966)
- C.A.R. HOARE: weitere Grundlagen (1969)
- DIJKSTRA in den 70er-Jahren
- D. GRIES: „The Science of Programming“
- J. LOECKX, K. SIEBER: „The Foundation of Programm Verification“

6.2 Programmentwicklung

Mathematik	**Programmierung**
Problem	Spezifikation
Satz	Programm
Beweis	Verifikation

A Automaten- und Sprachtypen

A.1 Automatentypen, Bestandteile

Automat		**Übergangsfunktion**
NEA(1)	$(Q,\ \Sigma,\ q_0,\ \Delta,\ F)$	$\Delta \subseteq Q \times \Sigma \times Q$
DEA(8)	$(Q,\ \Sigma,\ q_0,\ \delta,\ F)$	$\delta\colon Q \times \Sigma \to Q$
PDA(44)	$(Q,\ \Sigma,\ \Gamma,\ q_0,\ z_0,\ \Delta,\ F)$	$\Delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \times \Gamma^* \times Q$
DPDA(45)	$(Q,\ \Sigma,\ \Gamma,\ q_0,\ z_0,\ \delta,\ F)$	$\delta\colon Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to \Gamma^* \times Q$
NTM(55)	$(Q,\ \Sigma,\ \Gamma,\ q_0,\ \Delta,\ F)$	$\Delta \subseteq (Q \setminus F) \times \Gamma \times \Gamma \times \{l, r\} \times Q$
DTM(56)	$(Q,\ \Sigma,\ \Gamma,\ q_0,\ \delta,\ F)$	$\delta\colon (Q \setminus F) \times \Gamma \to \Gamma \times \{l, r\} \times Q$
LBA(58)	$(Q,\ \Sigma,\ \Gamma,\ q_0,\ ¢, \$,\ \delta,\ F)$	$\delta\colon (Q \setminus F) \times \Gamma \to \Gamma \times \{l, r\} \times Q$

A.2 Abschlußeigenschaften, Entscheidungsprobleme

Typ (32)	Typ 3		Typ 2	Typ 1	Typ 0
Bezeichnung	regulär	determin. kontextfrei	kontextfrei	kontext-sensitiv	rekursiv aufzählbar
Automaten	NEA/DEA	DPDA	PDA	LBA	TM
Beispiel	$a^n b$	$a^n b^n c^m$	$a^n b^n$	$a^n b^n c^n$	
abgeschlossen					
$X \cap \mathrm{R}$[14]	✓ (23)	✓ (52)	✓	✓	✓
$X \cup Y$	✓ (23)	✗ (53)	✓ (43)	✓	✓
$X \cap Y$	✓ (23)	✗ (53)	✗ (94)	✓	✓
$\Sigma^* \setminus X$	✓ (23)	✓ (53)	✗ (94)	✓	✗ (83)
$X \cdot Y$	✓ (23)	✗	✓ (43)	✓	✓
X^*	✓ (23)	✗	✓ (43)	✓	✓
Entscheidungen					
Wortproblem	✓	✓	✓ (39)	✓	✗ (85)
Leerheitsproblem	✓ (23)	✓	✓ (40)	✗ (91)	✗
Äquivalenzproblem	✓ (23)	✓	✗ (93)	✗	✗
Endlichkeitsproblem	✓ (23)				
Halteproblem					✗ (86)

[14] mit einer regulären Sprache R

B Funktionstypen

Typ	berechenbar	Definitionsbereich	Entstehtung
primitiv rekursiv	`LOOP`	total	Grundfunktion, Komposition, primitive Rekursion

μ-rekursiv	`WHILE`
	$\texttt{WHILE}_0$
	`GOTO`
	Turing
		evtl. partiell	Grundfunktion, Komposition, primitive Rekursion, unbeschr. Minimalisierung

C Hoare-Kalkül

Wertzuweisung
$$\frac{\varphi \to \psi\left(\frac{X}{t}\right)}{\{\varphi\}\ X := t\ \{\psi\}}$$

Kompositionsregel
$$\frac{\{\varphi\}\ P_1\ \{\psi\}, \{\psi\}\ P_2\ \{\chi\}}{\{\varphi\}\ P_1; P_2\ \{\chi\}}$$

`if-then-else`-Regel
$$\frac{\{\varphi \wedge \beta\}\ P_1\ \{\psi\}, \{\psi \wedge \neg\beta\}\ P_2\ \{\psi\}}{\{\varphi\}\ \texttt{if}\ \beta\ \texttt{then}\ P_1\ \texttt{else}\ P_2\ \{\psi\}}$$

`if-then`-Regel
$$\frac{\{\varphi \wedge \beta\}\ P_1\ \{\psi\}, \psi \wedge \neg\beta \to \psi}{\{\varphi\}\ \texttt{if}\ \beta\ \texttt{then}\ P_1\ \{\psi\}}$$

`while`-Regel
$$\frac{\{\varphi \wedge \beta\}\ P\ \{\varphi\}}{\{\varphi\}\ \texttt{while}\ \beta\ \texttt{do}\ P\{\varphi \wedge \neg\beta\}}$$

Konsequenzregeln
$$\frac{\{\varphi\}P\{\psi\}, \psi \to \chi}{\{\varphi\}\ P\{\chi\}}$$
$$\frac{\varphi \to \psi\{\psi\}P\{\chi\}}{\{\varphi\}\ P\{\chi\}}$$

Index