

# Contents

<b>1</b>	<b>Algorithmen</b>	<b>2</b>
1.1	REG . . . . .	2
1.1.1	DFA $\rightarrow$ Grammar . . . . .	2
1.1.2	NFA $\rightarrow$ DFA (Rabin und Scott) . . . . .	2
1.1.3	Grammatik $\rightarrow$ NFA . . . . .	2
1.1.4	NFA $\rightarrow$ L(M) Gleichungssysteme . . . . .	2
1.1.5	Regex $\rightarrow$ NFA . . . . .	3
1.1.6	Mihill Nerode Minimalautomaten . . . . .	3
1.2	CF . . . . .	3
1.2.1	CYK Algorithmus . . . . .	3
1.2.2	Kontextfrei $\rightarrow$ PDA . . . . .	3
1.2.3	PDA $\rightarrow$ Kontextfrei . . . . .	4
1.3	Normalformen . . . . .	4
1.3.1	kfG $\rightarrow$ $\lambda$ -frei . . . . .	4
1.3.2	Einfache Regeln entfernen . . . . .	4
1.3.3	$\lambda$ -frei ohne einfache Regeln $\rightarrow$ Chomsky-Normalform . . . . .	5
1.3.4	Bemerkung CNF . . . . .	5
1.3.5	Greibach-Normalform . . . . .	5
1.3.6	Bemerkung GNF . . . . .	6
1.4	CS . . . . .	6
1.4.1	CS $\rightarrow$ LBA . . . . .	6
1.5	TM . . . . .	6
1.5.1	LBA $\rightarrow$ CS, TM $\rightarrow \mathcal{L}_0$ . . . . .	6
1.5.2	$\mathcal{L}_r \rightarrow$ TM . . . . .	7
<b>2</b>	<b>Definitionen</b>	<b>7</b>
2.1	Begriffe . . . . .	7
2.2	Operationen . . . . .	7
2.3	Symbole . . . . .	8
2.4	Grammatik . . . . .	8
2.5	Sonderregelung für $\lambda$ . . . . .	8
2.6	Abschlusseigenschaften . . . . .	9
2.7	DFA . . . . .	9
2.8	NFA . . . . .	9
2.9	Regex . . . . .	9
2.10	Kellerautomaten (PDA) . . . . .	10
2.11	Deterministische Kontextfreie Sprache, DCF . . . . .	10
2.12	$\mathcal{L}_0$ Sprachen (Turingmaschinen) . . . . .	11
2.12.1	Linear beschränkte Automaten (LBA) . . . . .	11
<b>3</b>	<b>Sätze</b>	<b>11</b>
3.1	Satz von Parikh . . . . .	11
3.1.1	Parikh Abbildung . . . . .	11
3.1.2	Parikh . . . . .	12
3.1.3	Beispiel . . . . .	12
3.2	Pumping Lemma REG . . . . .	12
3.3	Pumping-Lemma CF . . . . .	12

# 1 Algorithmen

## 1.1 REG

### 1.1.1 DFA $\rightarrow$ Grammar

- $N = Z$ ,
- $S = z_0$ ,
- $P$ :
  - Gilt  $\delta(z, a) = z'$ , so ist  $z \rightarrow az'$  in  $P$ .
  - Ist  $z' \in F$ , so ist zusätzlich  $z \rightarrow a$  in  $P$ .
  - ist  $\lambda \in A$  (d.h.,  $z_0 \in F$ ), so ist auch  $z_0 \rightarrow \lambda$  in  $P$ , und die bisher konstruierte Grammatik wird gemäß der Sonderregel für  $\lambda$  modifiziert.

### 1.1.2 NFA $\rightarrow$ DFA (Rabin und Scott)

NFA  $M = (\Sigma, Z, \delta, S, E)$  und DFA  $M' = (\Sigma, \mathcal{P}(Z), \delta', z'_0, F)$

- Zustandsmenge von  $M'$ :  $\mathcal{P}(Z)$ ,
- $\delta'(Z', a) = \cup_{z \in Z'} \delta(z, a) = \hat{\delta}(Z', a)$  für, all  $Z' \subseteq Z$  und  $a \in \Sigma$ ,
- $z'_0 = S$ ,
- $F = \{Z' \subseteq Z \mid Z' \cap E \neq \emptyset\}$

### 1.1.3 Grammatik $\rightarrow$ NFA

- $Z = N \cup \{X\}$ , wobei  $X \notin N \cup \Sigma$  ein neues Symbol ist,
- $F = \begin{cases} \{S, X\} & \text{falls } S \rightarrow \lambda \text{ in } P \\ \{X\} & \text{falls } S \rightarrow \lambda \text{ nicht in } P, \end{cases}$
- $S' = \{S\}$  und
- für alle  $A \in N$  und  $a \in \Sigma$  sei

$$\delta(A, a) = \left( \bigcup_{A \rightarrow aB \in P} \{B\} \right) \cup \bigcup_{A \rightarrow a \in P} \{X\}.$$

### 1.1.4 NFA $\rightarrow$ L(M) Gleichungssysteme

Bilde ein Gleichungssystem mit  $n$  Variablen und  $n$  Gleichungen:

1. Jedes  $z_i \in Z, 1 \leq i \leq n$  ist Variable auf der linken Seite einer Gleichung
2. Gilt  $z_j \in \delta(z_i, a)$  für  $z_i, z_j \in Z$  und  $a \in \Sigma$ , so ist  $az_j$  Summand auf der rechten Seite der Gleichung „ $z_i = \dots$ “
3. Gilt  $z_i \in F$ , so ist  $\emptyset^*$  Summand auf der rechten Seite der Gleichung „ $z_i = \dots$ “.

Die  $z_i$  werden als reguläre Sprachen interpretiert und gemäß Lemma 2.24 und Satz 2.226 ausgerechnet. Es gilt dann:  $L(M) = \bigcup_{z_i \in S} z_i$  bzw.  $L(; ) = L(\alpha)$  für den regulären Ausdruck  $\alpha = \sum_{z_i \in S} z_i$ .

### 1.1.5 Regex $\rightarrow$ NFA

Fange mit Nuclei an und verbinde einfache Maschinen:

- $a \rightarrow$  Einfacher Übergang
- $a * b \rightarrow$  Einfacher Übergang oder Schleife
- $a? \rightarrow$  Einfacher Übergang und  $\lambda$  oder  $\lambda$

### 1.1.6 Mihill Nerode Minimalautomaten

$xR_L y$  zwischen  $x$  und  $y$  gilt genau dann, wenn  $(\forall z \in \Sigma^*)[xz \in L \leftrightarrow yz \in L]$ . Dies induziert eine Zerlegung von  $\Sigma^*$  in Äquivalenzklassen:

$$[x] = \{y \in \Sigma^* | xR_L y\}$$

Die Anzahl der Äquivalenzklassen ist  $\text{Index}(R_L) = \|\{[x] | x \in \Sigma^*\}\|$ .

$$L \in REG \leftrightarrow \text{Index}(R_L) < \infty$$

Von DFA  $M = (\Sigma, Z, \delta, z_0, F)$  zu  $M$  äquivalenter Minimalautomat

1. Entferne alle von  $z_0$  aus nicht erreichbaren Zustände aus  $Z$ .
2. Erstelle eine Tabelle aller (ungeordneten) Zustandspaare  $\{z, z'\}$  on  $M$  mit  $z \neq z'$ .
3. Markiere alle Paare  $\{z, z'\}$  mit  $z \in F \leftrightarrow z' \notin F$ .
4. Sei  $\{z, z'\}$  ein unmarkiertes paar. Prüfe für jedes  $a \in \Sigma$ , ob  $\{\delta(z, a), \delta(z', a)\}$  bereits markiert ist. Ist mindestens ein Test erfolgreich, so markiere auch  $\{z, z'\}$ .
5. Wiederhole Schritt 4, bis keine Änderung mehr eintritt.
6. Bilde maximale Mengen paarweise nicht disjunkter unmarkierter Zustandspaare und verschmelze jeweils alle Zustände einer Menge zu einem neuen Zustand.

## 1.2 CF

### 1.2.1 CYK Algorithmus

Überprüfe ob Wort in CF Sprache. Zeile i, Spalte j der Tabelle ist

```
1 for k in 0..<j:
2   t[i, j].add nonterminals_with_rule_to_pair( (i, k), (i+k+1, -k))
```

### 1.2.2 Kontextfrei $\rightarrow$ PDA

Von  $G = (\Sigma, N, S, P)$  zu  $M = (\Sigma, N \cup \Sigma, \{z\}, \delta, z, S)$ :

1. Ist  $A \rightarrow q$  eine Regel in  $P$  mit  $A \in N$  und  $q \in (N \cup \Sigma)^+$ , so sei  $(z, q) \in \delta(z, \lambda, A)$ .
2. Für jedes  $a \in \Sigma$  sei  $(z, \lambda) \in \delta(z, a, a)$ .

### 1.2.3 PDA → Kontextfrei

Von  $M = (\Sigma, \Gamma, Z, \delta, z_0, \#)$  zu  $G = (\Sigma, \{S\} \cup Z \times \Gamma \times Z, S, P)$ :

O.B.d.A Für alle  $\delta$ -Regeln der Form  $zaA \rightarrow z'B_1B_2 \dots B_k$  gelte  $k \leq 2$ .  $P$  besteht dann aus den folgenden Regeln:

1.  $S \rightarrow (z_0, \#, z)$  für jedes  $z \in Z$ .
2.  $(z, A, z') \rightarrow a$ , falls  $(z', \lambda) \in \delta(z, a, A)$ .
3.  $(z, A, z') \rightarrow a(z_1, B, z')$  falls  $(z_1, B) \in \delta(z, a, A)$ .
4.  $(z, A, z') \rightarrow a(z_1, B, z_2)(z_2, C, z')$ , falls  $(z_1, BC) \in \delta(z, a, A)$ .

Wobei  $z, z', z_1, z_2 \in Z, A, B, C \in \Gamma$  und  $a \in \Sigma \cup \{\lambda\}$ .

## 1.3 Normalformen

Ausnahmeregel für das leere Wort muss nur für Typ-2 Grammatiken nicht genutzt werden. Eine  $kfgG = (\Sigma, N, S, P)$  heißt  $\lambda$ -frei, falls in  $P$  keine Regel  $A \rightarrow \lambda$  mit  $A \neq S$  auftritt. Diese Umwandlung ist immer möglich.

### 1.3.1 kfG → $\lambda$ -frei

Wenn  $\lambda \in L(G)$  wende Sonderregelung für  $\lambda$  an.

1. Bestimme die Menge  $N_\lambda = \{A \in N \mid A \vdash_G^* \lambda\}$  sukzessive wie folgt:
  - (a) Ist  $A \rightarrow \lambda$  eine Regel in  $P$ , so ist  $A \in N_\lambda$ .
  - (b) Ist  $A \rightarrow A_1A_2 \dots A_k$  eine Regel in  $P$  mit  $k \geq 1$  und  $A_i \in N_\lambda$  für alle  $i, 1 \leq i \leq k$ , so ist  $A \in N_\lambda$ .
2. Füge für jede Regel der Form

$$B \rightarrow uAv \text{ mit } B \in N, A \in N_\lambda \text{ und } uv \in (N \cup \Sigma)^+$$

zusätzlich die Regel  $B \rightarrow uv$  zu  $P$  hinzu.

3. entferne alle Regeln  $A \rightarrow \lambda$  aus  $P$ .

Schritt 2 muss auch für neu generierte Regeln iterativ angewendet werden. Dies ergibt die gesuchte  $\lambda$ -freie kfG  $G'$  mit  $L(G) = L(G')$ .

### 1.3.2 Einfache Regeln entfernen

Regeln  $A \rightarrow B$  heißen einfach falls  $A, B \in N$

1. Entferne alle Zyklen

$$B_1 \rightarrow B_2, B_2 \rightarrow B_3, \dots, B_{k-1} \rightarrow B_k, B_k \rightarrow B_1 \text{ mit } B_i \in N$$

und ersetze all  $B_i$  (in den verbleibenden Regeln) durch ein neues Nicht-terminal  $B$ .

2. Nummeriere die Nichtterminale als  $\{A_1, A_2, \dots, A_n\}$  so, dass aus  $A_i \rightarrow A_j$  folgt:  $i < j$ .

3. Für  $k = n-1, n-2, \dots, 1$  (RÜCKWÄRTS!) eliminiere die Regel  $A_k \rightarrow A_l$  mit  $k < l$  so: Sind die Regeln mit  $A_l$  als linker Seite gegeben durch

$$A_l \rightarrow u_1 | u_2 | \dots | u_m,$$

so entferne  $A_k \rightarrow A_l$  und füge die folgenden Regeln hinzu:

$$A_k \rightarrow u_1 | u_2 | \dots | u_m.$$

### 1.3.3 $\lambda$ -frei ohne einfache Regeln $\rightarrow$ Chomsky-Normalform

**Bedingung:** Alle Regeln in  $P$  haben die form

- $A \rightarrow BC$  mit  $A, B, C \in N$ ;
- $A \rightarrow a$  mit  $A \in N$  und  $a \in \Sigma$ .

**Überführung:**

1. Regeln  $A \rightarrow a$  mit  $A \in N$  und  $a \in \Sigma$  sind *CNF* und werden übernommen. Alle anderen Regeln sind von der Form:  $A \rightarrow x$  mit  $x \in (N \cup \Sigma)^*$  und  $|x| \geq 2$ .
2. Füge für jedes  $a \in \Sigma$  ein neues Nichtterminal  $B_a$  zu  $N$  hinzu, ersetze jedes Vorkommen von  $a \in \Sigma$  durch  $B_a$  und füge zu  $P$  die Regel  $B_a \rightarrow a$  hinzu.
3. Nicht in *CNF* Sind nun nur noch Regeln der Form

$$A \rightarrow B_1 B_2 \dots B_k, \text{ wobei } k \geq 3 \text{ und jedes } B_i \text{ ein Nichtterminal ist.}$$

Jede solche Regel wird ersetzt durch die Regeln:

$$\begin{aligned} A &\rightarrow B_1 C_2, \\ C_2 &\rightarrow B_2 C_3, \\ &\vdots \\ C_{k-2} &\rightarrow B_{k-2} C_{k-1}, \\ C_{k-1} &\rightarrow B_{k-1} B_k, \end{aligned}$$

wobei  $C_2, C_3, \dots, C_{k-1}$  neue Nichtterminale sind.

Dies liefert die gesuchte Grammatik  $G'$  in *CNF* mit  $L(G) = L(G')$

### 1.3.4 Bemerkung CNF

- Ableitung von  $w$  in  $G$  in genau  $2|w| - 1$  Schritten.
- Syntaxbaum ist ein Binärbaum.

### 1.3.5 Greibach-Normalform

Zu jeder kfG mit  $\lambda \notin L(G)$  gibt es eine kfG  $G'$  in *GNF*, so dass  $L(G) = L(G')$  und jede Regel in Folgender Form ist:

$$A \rightarrow a B_1 B_2 \dots B_k \text{ mit } k \geq 0 \text{ und } a \in \Sigma$$

### 1.3.6 Bemerkung GNF

Man unterscheidet bezüglich der Länge der rechten Seite der Produktion:

- Jede kfG kann in eine äquivalente kfG in Greibach-Normalform transformiert werden, so dass für alle Regeln  $A \rightarrow aB_1B_2 \dots B_k$  stets  $k \leq 2$  gilt.
- Für den Spezialfall  $k \in \{0, 1\}$  erhalten wir gerade die Definition rechtslinearer Grammatiken.
- Die Ableitung eines Wortes  $w \in L(G)$ ,  $w \neq \lambda$  ist genau  $|w|$  Schritte lang wenn  $G$  in GNF steht.

## 1.4 CS

### 1.4.1 CS $\rightarrow$ LBA

Von  $G = (\Sigma, N, S, P)$  zu  $M = (\Sigma, \Gamma, Z, \delta, z_0, \square, F)$

1. Eingabe  $x = a_1a_2 \dots a_n$ .
2. Wähle nichtdeterministisch eine Regel  $u \rightarrow v$  aus  $P$  und suche eine beliebiges Vorkommen von  $v$  in der aktuellen Bandinschrift von  $M$ .
3. Ersetzt  $v$  durch  $u$ . Ist dabei  $|u| < |v|$ , so verschiebe entsprechend alle Symbole rechts der Lücke um diese zu schließen.
4. Ist die aktuelle Bandinschrift nur noch das Startsymbol  $S$ , so halte im Endzustand und akzeptiere; andernfalls gehe zu (2) und wiederhole.

## 1.5 TM

### 1.5.1 LBA $\rightarrow$ CS, TM $\rightarrow \mathcal{L}_0$

Von  $M = (\Sigma, \Gamma, Z, \delta, z_0, \square, F)$  und  $G = (\Sigma, \{S, A\} \cup \{\Delta \times \Sigma\}, S, P)$  mit

$$\Delta = \Gamma \cup (Z \times \Gamma)$$

Variablen:

- Alphabet ist Eingabe-Alphabet  $\Sigma$
- Nichtterminale sind Bandsymbole ohne Eingabe-Alphabet  $\Gamma - \Sigma$
- Nehme  $\#$  als  $\square$ , ein Symbol  $q_i$  für jeden Zustand der TM und  $\$$  um Startposition zu merken
- $S$  als Startsymbol und  $A$  als initialization

Übergänge:

- Setup
  - $S \rightarrow \#S\#\$A$  als Startzustand
  - $A \rightarrow sA|As|q_f$  für alle  $s \in \Gamma$
- $\delta(q_i, a) = (q_j, b, R)$  wird zu  $bq_j \rightarrow q_ia$

- $\delta(q_i, a) = (q_j, b, L)$  wird zu  $q_jcb \rightarrow cq_ia$  für alle  $c \in \text{Gamma}$
- Cleanup
  - $\# \rightarrow \lambda$
  - $\$q_0 \rightarrow \lambda$

Von  $M = (\Sigma, \Gamma, Z, \delta, z_0, \square, F)$  und  $G = (\Sigma, \{S, A\} \cup \{\Delta \times \Sigma\}, S, P)$  mit

$$\Delta = \Gamma \cup (Z \times \Gamma)$$

### 1.5.2 $\mathcal{L}_I \rightarrow \text{TM}$

1. Eingabe  $x = a_1a_2 \dots a_n$ .
2. Wähle nichtdeterministisch eine Regel  $u \rightarrow v$  aus  $P$  und suche eine beliebige Vorkommen von  $v$  in der aktuellen Bandinschrift von  $M$ .
3. Ersetzt  $v$  durch  $u$ . Ist dabei  $|u| < |v|$ , so verschiebe entsprechend alle Symbole rechts der Lücke um diese zu schließen.
4. Ist die aktuelle Bandinschrift nur noch das Startsymbol  $S$ , so halte im Endzustand und akzeptiere; andernfalls gehe zu (2) und wiederhole.

## 2 Definitionen

### 2.1 Begriffe

- Ein Alphabet ist eine endliche, nichtleere Menge  $\Sigma$  von Buchstaben (oder Symbolen).
- Ein Wort über  $\Sigma$  ist eine endliche Folge von Elementen aus  $\Sigma$ .
- Die Länge eines Wortes  $w$  (bezeichnet mit  $|w|$ ) ist die Anzahl der Symbole in  $w$ .
- Das leere Wort ist das eindeutig bestimmte Wort der Länge 0 und wird mit dem griechischen Buchstaben  $\lambda$  bezeichnet.
- Die Menge aller Wörter über  $\Sigma$  bezeichnen wir mit  $\Sigma^*$ .
- Eine formale Sprache über  $\Sigma$  ist eine jede Teilmenge von  $\Sigma^*$ .
- Die leere Sprache ist die Sprache die keine Wörter enthält, und wird mit  $\emptyset$  bezeichnet.
- die Kardinalität einer Sprache  $L$  ist die Anzahl der Wörter von  $L$  und wird mit  $\|L\|$  bezeichnet.

### 2.2 Operationen

- Vereinigung  $\{1, 2\} \cup \{2, 3\} = \{1, 2, 3\}$ .
- Durchschnitt  $\{1, 2\} \cap \{2, 3\} = \{2\}$ .
- Differenz  $A - B = \{x \in A \text{ und } x \notin B\}$ .
- Komplement  $\overline{A} = \{x \in \Sigma^* | x \notin A\}$ .

- Konkatination von Wörtern
  - Ist  $u = v = \lambda$ , so ist  $uv = vu = \lambda$ .
  - Ist  $v = \Lambda$ , so ist  $uv = u$ .
  - Ist  $u = \lambda$  so ist  $uv = v$ .
  - Ist  $u = u_1u_2 \dots u_n$  und  $v = v_1v_2 \dots v_m$  mit  $u_i, v_i \in \Sigma$ , so ist

$$uv = u_1u_2 \dots u_nv_1v_2 \dots v_m.$$

- Konkatination von Sprachen:  $AB = \{ab | a \in A \text{ und } b \in B\}$ .
- Iteration einer Sprache:  $A^0 = \{\lambda\}$ ,  $A^n = AA^{n-1}$ ,  $A^* = \bigcup_{n \geq 0} A^n$ .
- Spiegelbildoperation von Wort  $sp(u) = u_n u_{n-1} \dots u_1$ .
- Spiegelbildoperation von Sprache  $sp(A) = \{sp(w) | w \in A\}$ .
- Teilwortrelation auf  $\Sigma^*$ :  $u \supseteq v \leftrightarrow (\exists v_1, v_2 \in \Sigma^*) [v_1uv_2 = v]$ .
- Anfangswortrelation auf  $\Sigma^*$ :  $u \supseteq_a v \leftrightarrow (\exists w \in \Sigma^*) [uw = v]$ .

## 2.3 Symbole

- $\Sigma$  ein Alphabet von Terminalsymbolen
- $N$  eine Endliche Menge von Nichtterminalen,  $\Sigma \cap N = \emptyset$
- $S$  Startsymbol,  $S \in N$
- $P$  Produktionsregeln,  $P \subseteq (N \cup \Sigma)^+ \times (N \cup \Sigma)^*$

## 2.4 Grammatik

$$G = (\Sigma, N, S, P)$$

- Typ-0: Ohne Einschränkungen.
- Typ-1:  $\forall p \rightarrow q \in P : |p| \leq |q|$
- Typ-2:  $\forall p \rightarrow q \in P : p \in N$
- Typ-3:  $\forall p \rightarrow q \in P : |p| \in N \text{ und } q \in \Sigma \cup \Sigma N$

$$REG \subseteq CF \subseteq CS \subseteq \mathcal{L}_0$$

## 2.5 Sonderregelung für $\lambda$

Typ-i Grammatiken mit  $i \in \{1, 2, 3\}$  sind nichtverkürzend, daher  $\lambda \notin L(G)$ .  
Daher folgende Sonderregelung:

1. Die Regel  $S \rightarrow \lambda$  ist als einzige verkürzende Regel für Grammatiken vom Typ 1, 2, 3 zugelassen.
2. Tritt die Regel  $S \rightarrow \lambda$  auf, so darf  $S$  auf keiner rechten Seite einer Regel vorkommen.

Dies kann für alle Fälle mit folgender Umwandlung erreicht werden:

1. In allen Regeln der Form  $S \rightarrow u$  aus  $P$  mit  $u \in (N \cup \Sigma)^*$  wird jedes Vorkommen von  $S$  in  $u$  durch ein neues Nichtterminal  $S'$  ersetzt.
2. Zusätzlich enthält  $P'$  alle Regeln aus  $P$ , mit  $S$  ersetzt durch  $S'$ .
3. Die Regel  $S \rightarrow \lambda$  wird hinzugefügt.



## 2.6 Abschlusseigenschaften

1. Vereinigung, falls  $(\forall A, B \subseteq \Sigma^*)[(A \in \mathcal{C} \wedge B \in \mathcal{C}) \implies A \cup B \in \mathcal{C}]$ ;
2. Komplement, falls  $(\forall A \subseteq \Sigma^*)[A \in \mathcal{C} \implies \overline{A} \in \mathcal{C}]$ ;
3. Schnitt, falls  $(\forall A, B \subseteq \Sigma^*)[(A \in \mathcal{C} \wedge B \in \mathcal{C}) \implies A \cap B \in \mathcal{C}]$ ;
4. Differenz, falls  $(\forall A, B \subseteq \Sigma^*)[(A \in \mathcal{C} \wedge B \in \mathcal{C}) \implies A \setminus B \in \mathcal{C}]$ ;
5. Konkatenation, falls  $(\forall A, B \subseteq \Sigma^*)[(A \in \mathcal{C} \wedge B \in \mathcal{C}) \implies AB \in \mathcal{C}]$ ;
6. Iteration (Kleene-Hülle), falls  $(\forall A \subseteq \Sigma^*)[A \in \mathcal{C} \implies A^* \in \mathcal{C}]$ ;
7. Spiegelung, falls  $(\forall A \subseteq \Sigma^*)[A \in \mathcal{C} \implies sp(A) \in \mathcal{C}]$ ;

## 2.7 DFA

- $\Sigma$ : Alphabet
- $Z$ : endliche Menge von Zuständen mit  $\Sigma \cap Z = \emptyset$
- $\delta : Z \times \Sigma \rightarrow Z$  Überföhrungsfunktion
- $z_0 \in Z$  Startzustand
- $F \subseteq Z$  Endzustände

## 2.8 NFA

$M = (\Sigma, Z, \delta, S, F)$

- $\Sigma$ : Alphabet
- $Z$ : endliche Menge von Zuständen mit  $\Sigma \cap Z = \emptyset$
- $\delta : Z \times \Sigma \rightarrow \mathcal{P}(Z)$ : Überföhrungsfunktionen zur Potenzmenge von  $Z$
- $S \subseteq Z$ : Menge der Startzustände
- $F \subseteq Z$  Menge der Endzustände

## 2.9 Regex

- $\emptyset$  und  $\lambda$  sind reguläre Ausdröcke
- Jedes  $a \in \Sigma$  ist ein regulärer Ausdruck.
- Sind  $\alpha$  und  $\beta$  reguläre Ausdröcke, so sind auch
  - $\alpha\beta$
  - $(\alpha + \beta)$  und
  - $(\alpha)^*$reguläre Ausdröcke
- Nichts sonst ist ein regulärer Ausdruck

## 2.10 Kellerautomaten (PDA)

$M = (\Sigma, \Gamma, Z, \delta, z_0, \#)$

- $\Sigma$ : Eingabe-Alphabet
- $\Gamma$ : Kelleralphabet
- $Z$ : endliche Menge von Zuständen
- $\delta : Z \times (\Sigma \cup \{\lambda\}) \times \Gamma \rightarrow \mathcal{P}_e(Z \times \Gamma^+)$  die Überföhrungsfunktion,  $\mathcal{P}_e(Z \times \Gamma^+)$  ist die Menge aller endlichen Teilmengen von  $Z \times \Gamma^+$
- $z_0 \in Z$ : Startzustand
- $\# \in \Gamma$  das Bottom-Symbol im Keller

$\delta$ -Übergänge  $(z', B_1, B_2 \dots B_k) \in \delta(z, a, A)$  schreiben wir kurz auch  $zaA \rightarrow z'B_1B_2 \dots B_k$

- Akzeptiert wenn Keller leer
- Schritte ohne Lesen eines Eingabesymbols sind in der Form  $z\lambda A \rightarrow z'B_1B_2 \dots B_k$  möglich.
- Es ist nur ein Startzustand nötig

Sprache:

- $\|_m = Z \times \Sigma^* \times \Gamma^*$  ist die Menge aller Konfigurationen von M
- ist  $k = (z, \alpha, \gamma)$  eine Konfiguration aus  $\|_M$ , so ist im aktuellen Takt der Rechnung von M:
  - $z \in Z$  der aktuelle Zustand von M;
  - $\alpha \in \Sigma^*$  der noch zu lesende Teil des Eingabeworts;
  - $\gamma \in \Gamma^*$  der aktuelle Kellerinhalt.

Für jedes Eingabewort  $w \in \Sigma^*$  ist  $(z_0, w, \#)$  die entsprechende Startkonfiguration von M.

- Auf  $\|_m$  definieren wir eine binäre Relation  $\vdash_m \subseteq \|_M \times \|_M$  wie folgt:
  - Für  $k, k' \in \|_M$  gilt  $k \vdash_m k'$  genau dann, wenn  $k'$  aus  $k$  durch eine Anwendung von  $\delta$  hervorgeht.
- $\vdash_M^*$  ist die reflexive und transitive Hölle von  $\vdash_M$
- Die vom PDA  $M$  akzeptierte Sprache ist definiert durch

$$L(M) = \{w \in \Sigma^* \mid (z_0, w, \#) \vdash_M^* (z, \lambda, \lambda) \text{ für ein } z \in Z\}$$

$(z, \lambda, \lambda)$  ist dann eine Endkonfiguration für PDA  $M$ .

## 2.11 Deterministische Kontextfreie Sprache, DCF

$M = (\Sigma, \Gamma, Z, \delta, z_0, \#, F)$

1.  $M' = (\Sigma, \Gamma, Z, \delta, z_0, \#)$  ist PDA
2.  $(\forall a \in \Sigma)(\forall A \in \Gamma)(\forall z \in Z)[\|\delta(z, a, A)\| + \|\delta(z, \lambda, A)\| \leq 1]$ ;
3.  $F \subseteq Z$  ist eine ausgezeichnete Teilmenge von Endzuständen (M akzeptiert per Endzustand, nicht per leerem Keller)

## 2.12 $\mathcal{L}_0$ Sprachen (Turingmaschinen)

$M = (\Sigma, \Gamma, Z, \delta, z_0, \square, F)$

Mit  $k$  Bändern:

- $\Sigma$ : Eingabe-Alphabet,
- $\Gamma$ : Arbeitsalphabet mit  $\Sigma \subseteq \Gamma$ ,
- $Z$ : endliche Menge von Zuständen mit  $Z \cap \Gamma = \emptyset$ ,
- $\delta : Z \times \Gamma^k \rightarrow \mathcal{P}(Z \times \Gamma^k \times \{L, R, N\}^k)$ : Überföhrungsfunktion,
- $z_0 \in Z$ : Startzustand,
- $\square \in \Gamma - \Sigma$ : „Blank“-Symbol
- $F \subseteq Z$ : Menge der Endzustände

Spezialfall der deterministischen Turingmaschine mit  $k$  Bändern mit  $k$  Bändern ergibt sich, wenn  $\delta$  von  $Z \times \Gamma^k$  nach  $Z \times \Gamma^k \times \{L, R, N\}^k$  abbildet. Für  $k = 1$  ergibt sich die 1-Band-Turingmaschine die mit TM abgekürzt wird. Jede Turingmaschine kann durch eine TM simuliert werden. Statt  $(z', b, x) \in \delta(z, a)$  mit  $z, z' \in Z, x \in \{L, R, N\}, a, b \in \Gamma$  schreiben wir kurz:  $(z, a) \rightarrow (z', b, x)$ , oder  $za \rightarrow z'bx$

### 2.12.1 Linear beschränkte Automaten (LBA)

1. Verdoppele das Eingabe-Alphabet  $\Sigma$  mit  $\hat{\Sigma} = \Sigma \cup \{\hat{a} | a \in \Sigma\}$
  2. Repräsentiere die Eingabe  $a_1a_2 \dots a_n \in \Sigma^+$  durch das Wort  $a_1a_2 \dots a_{n-1}\hat{a}_n$  über  $\hat{\Sigma}$
- Eine nichtdeterministische TM  $M$  heißt linear beschränkter Automat (kurz LBA), falls für all Konfigurationen  $\alpha z \beta$  und

– für alle Wörter  $x = a_1a_2 \dots a_{n-1}a_n \in \Sigma^+$  mit

$$z_0a_1a_2 \dots a_{n-1}\hat{a}_n \vdash_m^* \alpha z \beta$$

gilt:  $|\alpha\beta| = n$ , und

– für  $x = \lambda$  mit  $z_0\square \vdash_M^* \alpha z \beta$  gilt:  $\alpha\beta = \square$

- Die vom LBA  $M$  akzeptierte Sprache ist definiert durch

$$L(M) = \left\{ a_1a_2 \dots a_{n-1}a_n \in \Sigma^* \left| \begin{array}{l} z_0a_1a_2 \dots a_{n-1}\hat{a}_n \vdash_m^* \alpha z \beta \\ \text{mit } z \in F \text{ und } \alpha, \beta \in \Gamma^* \end{array} \right. \right\}$$

## 3 Sätze

### 3.1 Satz von Parikh

#### 3.1.1 Parikh Abbildung

$\Psi : LL \rightarrow \mathcal{N}^n$  ist definiert durch

$$\Psi(w) = (|w|_a1, |w|_a2, \dots, |w|_an),$$

wobei  $|w|_a$  die Anzahl der Vorkommen des Zeichens  $a$  in  $w$  angibt. Für Sprachen:  $\Psi : \mathcal{P}(\Sigma^*) \rightarrow \mathcal{P}(\mathcal{N}^n)$  ist definiert durch

$$\Psi(L) = \{\Psi(w) | w \in L\}$$

Eine Menge ist semilinear wenn sie aus linearen Mengen zusammengesetzt ist.

### 3.1.2 Parikh

Für jede kontextfreie Sprache  $L$  in  $\Psi(L)$  ist semilinear

### 3.1.3 Beispiel

$$L = \{x \in \{0, 1\}^* | x = 1^k \text{ mit } k \geq 0 \text{ oder } x = 0^j 1^{k^2} \text{ mit } j \geq 1 \text{ und } k \geq 1\}$$

über  $\Sigma = \{0, 1\}$ . Die Menge

$$\Psi(L) = \{(i, j) | (i = 0 \text{ und } j \geq 0) \text{ oder } (i \geq 1 \text{ und } j = k^2 \text{ und } k \geq 1)\}$$

ist nicht semilinear und  $L$  somit nicht kontextfrei.

## 3.2 Pumping Lemma REG

Sei  $L \in \text{REG}$ . Dann existiert eine (von  $L$  abhängige) Zahl  $n \geq 1$ , so dass sich alle Wörter  $x \in L$  mit  $|x| \geq n$  zerlegen lassen in  $x = uvw$  wobei gilt:

1.  $|uv| \leq n$ ,
2.  $|v| \geq 1$ ,
3.  $(\forall i \geq 0) [uv^i w \in L]$ .

## 3.3 Pumping-Lemma CF

Sei  $L$  eine kontextfreie Sprache. Dann existiert eine (von  $L$  abhängige) Zahl  $n \geq 1$ , so dass sich alle Wörter  $z \in L$  mit  $|z| \geq n$  zerlegen lassen in  $z = uvwxy$ , wobei gilt:

1.  $|vx| \geq 1$ .
2.  $|vwx| \leq n$ .
3.  $(\forall i \geq 0) [uv^i wx^i y \in L]$ .

Wortproblem:

$$\text{Wort}_i = \{(G, X) | \text{Gist Typ-i-Grammatik und } x \in L(G)\}$$

Typ 3	reguläre Grammatik deterministischer endlicher Automat (DFA) nichtdeterministischer endlicher Automat (NFA) regulärer Ausdruck (REG)
deterministisch	LR(1)-Grammatik deterministischer Kellerautomat (DPDA)
Typ 2	kontextfreie Grammatik Kellerautomat (PDA)
Typ 1	kontextsensitive Grammatik linear beschränkter Automat (LBA)
Typ 0	Typ-0-Grammatik Turingmaschine (NTM bzw. DTM)

Deterministischer Automat	Nichtdeterministischer Automat	äquivalent?
DFA	NFA	ja
DPDA	PDA	nein
DLBA	LBA	?
DTM	NTM	ja

	Typ 3	det.kf.	Typ2	Typ 1	Typ 0
Schnitt	ja	nein	nein	ja	ja
Vereinigung	ja	nein	ja	ja	ja
Komplement	ja	ja	nein	ja	nein
Konkatenation	ja	nein	ja	ja	ja
Iteration	ja	nein	ja	ja	ja
Spiegelung	ja	nein	ja	ja	ja

Typ 3 (DFA gegeben)	lineare Komplexität
det. kf.	lineare Komplexität
Typ 2 (CNF gegeben)	Komplexität $\mathcal{O}(n^3)$ (CYK-Algorithmus)
Typ 1	exponentielle Komplexität
Typ 0	unentscheidbar (d.h. algorithmisch nicht lösbar)