

# **Vorlesung**

## **Informatik III – Theoretische Informatik**

**Formale Sprachen, Berechenbarkeit, Komplexitätstheorie**

Matthias Heizmann

Basierend auf einem Mitschrieb von Ralph Lesch\*  
der von Prof. Dr. Peter Thiemann im WS 2015/16 gehaltenen Vorlesung

WS 2017/18

Zuletzt aktualisiert: 2017-12-28

\*ralph.lesch@neptun.uni-freiburg.de

## Inhaltsverzeichnis

<b>1</b>	<b>Vorspann: Sprachen</b>	<b>4</b>
<b>2</b>	<b>Reguläre Sprachen und endliche Automaten</b>	<b>7</b>
2.1	Endliche Automaten . . . . .	7
2.2	Minimierung endlicher Automaten . . . . .	12
2.2.1	Exkurs: Äquivalenzrelationen . . . . .	13
2.3	Pumping Lemma (PL) für reguläre Sprachen . . . . .	21
2.4	nichtdeterministischer endlicher Automat (NEA) . . . . .	22
2.4.1	$\varepsilon$ -Übergänge . . . . .	26
2.5	Abschlusseigenschaften . . . . .	31
2.6	Reguläre Ausdrücke . . . . .	32
2.6.1	Motivation . . . . .	32
<b>3</b>	<b>Grammatiken und kontextfreie Sprachen</b>	<b>40</b>
3.1	Kontextfreie Sprachen . . . . .	43
3.2	Die Chomsky-Normalform für kontextfreie Sprachen . . . . .	45
3.3	Wortproblem und Leerheitsproblem für kontextfreie Sprachen . . . . .	50
3.4	Einschub: Typ-3-Sprachen . . . . .	52
<b>4</b>	<b>Kellerautomaten (PDA)</b>	<b>54</b>
<b>5</b>	<b>Kontextfreie Sprachen – Teil 2</b>	<b>62</b>
5.1	Das Pumping Lemma für kontextfreie Sprachen . . . . .	62
5.2	Abschlusseigenschaften für kontextfreie Sprachen . . . . .	64
5.3	Deterministisch kontextfreie Sprachen . . . . .	64
<b>6</b>	<b>Turingmaschinen</b>	<b>68</b>
6.1	Turingmaschine (informell) . . . . .	68
6.2	Formalisierung der Turing-Maschine (TM) . . . . .	68
6.3	Techniken zur TM Programmierung . . . . .	71
6.4	Das Gesetz von Church-Turing (Churchsche These) . . . . .	73
	<b>Liste der Definitionen</b>	<b>74</b>
	<b>Liste der Sätze</b>	<b>75</b>
	<b>Abbildungsverzeichnis</b>	<b>76</b>
	<b>Abkürzungsverzeichnis</b>	<b>76</b>

Inhaltsverzeichnis	3
<b>Anmerkungsverzeichnis</b>	<b>78</b>

# 1 Vorspann: Sprachen

Vorlesung:  
18.10.2017

**Def. 1.1:** Ein *Alphabet* ist eine endliche Menge von *Zeichen*. ◇

Zeichen sind hier beliebige abstrakte Symbole.

**Bsp.:** für Alphabete, die in dieser Vorlesung, im täglichen Umgang mit Computern oder in der Forschung an unserem Lehrstuhl eine Rolle spielen

- $\{a, \dots, z\}$
- $\{0, 1\}$
- $\{\text{rot}, \text{gelb}, \text{grün}\}$  (Ampelfarben)
- Die Menge aller ASCII-Symbole
- Die Menge aller Statements eines Computerprogramms

Wir verwenden typischerweise den griechischen Buchstaben  $\Sigma$  als Namen für ein Alphabet und die lateinischen Buchstaben  $a, b, c$  als Namen für Zeichen.<sup>1</sup> Im Folgenden sei  $\Sigma$  immer ein beliebiges Alphabet.<sup>2</sup>

**Def. 1.2:** Wir nennen eine endliche Folge von Elementen aus  $\Sigma$  ein *Wort* und schreiben solch eine Folge immer ohne Trennsymbole wie z.B. Komma.<sup>3</sup> Die leere Folge nennen wir das *leere Wort*; als Konvention stellen wir das leere Wort mit dem griechischen Buchstaben  $\varepsilon$  dar.<sup>4</sup> Wir bezeichnen die Menge aller Wörter mit  $\Sigma^*$  und die Menge aller nicht leeren Wörter mit  $\Sigma^+$ . Die *Länge* eines Wortes,  $|\cdot| : \Sigma^* \rightarrow \mathbb{N}$ , ist die Anzahl der Elemente der Folge. ◇

Wir verwenden typischerweise  $u, v, w$  als Namen für Wörter.

**Bsp.:** für Wörter über  $\Sigma = \{a, \dots, z\}$

- **rambo** (Länge 5)
- **eis, ies** (beide Länge 3 aber ungleich)
- $\varepsilon$  (Länge 0)

<sup>1</sup>Dies ist eine Konventionen analog zu den folgenden, die Sie möglicherweise in der Schule befolgten: Verwende  $n, m$  für natürliche Zahlen. Verwende  $\alpha, \beta$  für Winkel in Dreiecken. Verwende  $A$  für Matrizen.

<sup>2</sup>Dieser Satz dient dazu, dass die Autoren dieses Skripts nicht jede Definition mit “Sei  $\Sigma$  ein Alphabet...” beginnen müssen.

<sup>3</sup>Wir schreiben also z.B. **einhorn** statt **e,i,n,h,o,r,n**.

<sup>4</sup>Eine analoge Konvention, die sie aus der Schule kennen: Verwende immer  $\pi$  als Symbol für die Kreiszahl.

Wörter lassen sich „verkett“/„hintereinanderreihen“. Die entsprechende Operation heißt *Konkatenation*, geschrieben „ $\cdot$ “ (wie Multiplikation).

**Def. 1.3** (Konkatenation von Wörtern): Die *Konkatenation*,  $\cdot : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ , ist für  $u = u_1 \dots u_n \in \Sigma^*$  und  $v = v_1 \dots v_m \in \Sigma^*$  definiert durch  $u \cdot v = u_1 \dots u_n v_1 \dots v_m$   $\diamond$

**Bsp.:**

- $\text{eis} \cdot \text{rambo} = \text{eisrambo}$
- $\text{rambo} \cdot \varepsilon = \text{rambo} = \varepsilon \cdot \text{rambo}$

Eigenschaften von „ $\cdot$ “:

- assoziativ
- $\varepsilon$  ist neutrales Element
- *nicht* kommutativ

Der Konkatenationsoperator „ $\cdot$ “ wird oft weggelassen (ähnlich wie der Multiplikationsoperator in der Arithmetik). Ebenso können durch die Assoziativität Klammern weggelassen werden.

$w_1 w_2 w_3$  steht also auch für  $w_1 \cdot w_2 \cdot w_3$ , für  $(w_1 \cdot w_2) \cdot w_3$  und für  $w_1 \cdot (w_2 \cdot w_3)$

Bemerkung: Die Zeichenfolge  $\text{rambo}\varepsilon$  ist *kein* Wort. Diese Zeichenfolge ist lediglich eine Notation für eine Konkatenationsoperation, die ein Wort der Länge 5 (nämlich  $\text{rambo}$ ) beschreibt.

Wörter lassen sich außerdem *potenzieren*:

**Def. 1.4:** Die *Potenzierung* von Wörtern,  $\cdot^{\cdot} : \Sigma^* \times \mathbb{N} \rightarrow \Sigma^*$ , ist induktiv definiert durch

1.  $w^0 = \varepsilon$
2.  $w^{n+1} = w \cdot w^n$   $\diamond$

**Bsp.:**  $\text{eis}^3 \stackrel{(2.)}{=} \text{eis} \cdot \text{eis}^2 \stackrel{\text{zweimal}}{=} \stackrel{(2.)}{=} \text{eis} \cdot \text{eis} \cdot \text{eis} \cdot \text{eis}^0 \stackrel{(1.)}{=} \text{eis} \cdot \text{eis} \cdot \text{eis} \cdot \varepsilon = \text{eiseiseis}$

**Def. 1.5:** Eine *Sprache* über  $\Sigma$  ist eine Menge  $L \subseteq \Sigma^*$ .  $\diamond$

**Bsp.:**

- $\{\mathbf{eis}, \mathbf{rambo}\}$
- $\{w \in \{0, 1\}^* \mid w \text{ ist Binärcodierung einer Primzahl}\}$
- $\{\}$  (die „leere Sprache“)
- $\{\varepsilon\}$  (ist verschieden von der leeren Sprache)
- $\Sigma^*$

Sämtliche Mengenoperationen sind auch Sprachoperationen, insbesondere Schnitt ( $L_1 \cap L_2$ ), Vereinigung ( $L_1 \cup L_2$ ), Differenz ( $L_1 \setminus L_2$ ) und Komplement ( $\overline{L_1} = \Sigma^* \setminus L_1$ ).

Weitere Operationen auf Sprachen sind Konkatenation und Potenzierung, sowie der *Kleene-Abschluss*.

**Def. 1.6** (Konkatenation und Potenzierung von Sprachen): Seien  $U, V \subseteq \Sigma^*$ . Dann ist die *Konkatenation* von  $U$  und  $V$  definiert durch

$$U \cdot V = \{uv \mid u \in U, v \in V\}$$

und die *Potenzierung* von  $U$  induktiv definiert durch

1.  $U^0 = \{\varepsilon\}$
2.  $U^{n+1} = U \cdot U^n$  ◇

**Bsp.:**

- $\{\mathbf{eis}, \varepsilon\} \cdot \{\mathbf{rambo}\} = \{\mathbf{eisrambo}, \mathbf{rambo}\}$
- $\{\mathbf{eis}, \varepsilon\} \cdot \{\} = \{\}$
- $\{\}^0 = \{\varepsilon\}$
- $\{\}^4 = \{\}$
- $\{\mathbf{eis}, \varepsilon\}^2 = \{\varepsilon, \mathbf{eis}, \mathbf{eiseis}\}$

Wie bei der Konkatenation von Wörtern dürfen wir den Konkatenationsoperator auch weglassen.

**Def. 1.7** (Kleene-Abschluss, Kleene-Stern): Sei  $U \subseteq \Sigma^*$ . Der *Kleene-Abschluss* ist definiert als

1.  $U^* = \bigcup_{n \in \mathbb{N}} U^n \quad [\ni \varepsilon]$
2.  $U^+ = \bigcup_{n \geq 1} U^n$  ◇

## 2 Reguläre Sprachen und endliche Automaten

Vorlesung:  
20.10.17

Wie können wir potentiell unendlich große Mengen von Wörtern darstellen? Eine Lösung für dieses Problem sahen wir bereits im vorherigen Kapitel, als wir die (unendlich große) Menge der binär codierten Primzahlen mit Hilfe der folgenden Zeile darstellten.

$$L_{\text{prim}} = \{w \in \{0,1\}^* \mid w \text{ ist Binärcodierung einer Primzahl}\}$$

Ein weiteres Beispiel ist die folgende Zeile.

$$L_{\text{even}} = \{w \in \{0,1\}^* \mid \text{die Anzahl der Einsen in } w \text{ ist gerade.}\}$$

Eine häufig interessante Fragestellung für ein gegebenes Wort  $w$  und eine Sprache  $L$  ist: „Ist  $w$  in  $L$  enthalten?“ (Also: „Gilt  $w \in L$ ?“) Wir nennen dieses Entscheidungsproblem das *Wortproblem*. Eine konkrete Instanz des Wortproblems wäre z.B. „1100101  $\in L_{\text{prim}}$ ?“ oder „1100101  $\in L_{\text{even}}$ ?“

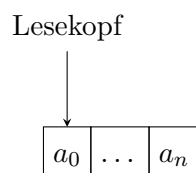
Die obige Darstellung der unendlichen Mengen  $L_{\text{prim}}$  und  $L_{\text{even}}$  ist zwar sehr kompakt, wir können daraus aber nicht direkt ein Vorgehen zur Lösung des Wortproblems ableiten. Wir müssen zunächst verstehen, was die Begriffe „Binärcodierung“, „Primzahl“ oder „gerade Anzahl“ bedeuten und für  $L_{\text{prim}}$  und  $L_{\text{even}}$  jeweils einen Algorithmus zur Entscheidung entwickeln.

In diesem Kapitel werden wir mit *endlichen Automaten* einen weiteren Formalismus kennenlernen, um (potentiell unendlich große) Mengen von Wörtern darzustellen. Ein Vorteil dieser Darstellung ist, dass es einen einheitlichen und effizienten Algorithmus für das Wortproblem gibt. Wir werden aber auch sehen, dass sich nicht jede Sprache (z.B.  $L_{\text{prim}}$ ) mit Hilfe eines endlichen Automaten darstellen lässt.

### 2.1 Endliche Automaten

Wir beschreiben zunächst informell die Bestandteile eines endlichen Automaten:

**Endliches Band** (read-only; jede Zelle enthält ein  $a_i \in \Sigma$ ; der Inhalt des Bands ist das *Eingabewort* bzw. die *Eingabe*)



**Abb. 1:** Endliches Band

**Lesekopf**

- Der *Lesekopf* zeigt auf ein Feld des Bands, oder hinter das letzte Feld.
- Er bewegt sich feldweise nach rechts; andere Bewegungen (Vor- bzw. Zurückspulen) sind nicht möglich.
- Wenn er hinter das letzte Zeichen zeigt, *stoppt* der Automat. Er muss sich nun „entscheiden“ ob er das Wort *akzeptiert* oder nicht.

**Zustände**  $q$  aus *endlicher* Zustandsmenge  $Q$

**Startzustand**  $q^{\text{init}} \in Q$

**Akzeptierende Zustände**  $F \subseteq Q$

**Transitionsfunktion** Im Zustand  $q$  beim Lesen von  $a$  gehe in Zustand  $\delta(q) = q'$ .

Der endliche Automat akzeptiert eine Eingabe, falls er in einem akzeptierenden Zustand stoppt.

**Bsp. 2.1:** Aufgabe:

„Erkenne alle Stapel von Macarons, in denen höchstens ein grünes Macaron vorkommt.“



5

Ein passendes Alphabet wäre  $\Sigma = \{\text{grün}, \text{nicht-grün}\}$ . Wir definieren die folgenden Zustände. (Die Metapher hier ist: „wenn ich mehr als ein grünes Macaron esse, wird mir übel, und das wäre nicht akzeptabel“.)

Zustand	Bedeutung
$q_0$	„alles gut“
$q_1$	„mir wird schon flau“
$q_2$	„mir ist übel“

Der Startzustand ist  $q_0$ . Akzeptierende Zustände sind  $q_0$  und  $q_1$ . Die Transitionsfunktion  $\delta$  ist durch die folgende Tabelle gegeben.

<sup>5</sup> Von links nach rechts:  
 By Mariajudit - Own work, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=48726001>  
 By Michelle Naherny - Own work, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=44361114>  
 By Keven Law - originally posted to Flickr as What's your Colour???, CC BY-SA 2.0, <https://commons.wikimedia.org/w/index.php?curid=6851868>



	grün	nicht-grün	
$q_0$	$q_1$	$q_0$	wechsle nach $q_1$ falls <b>grün</b> , ansonsten verweile
$q_1$	$q_2$	$q_1$	wechsle nach $q_2$ falls <b>grün</b> , ansonsten verweile
$q_2$	$q_2$	$q_2$	verweile, da es nichts mehr zu retten gibt

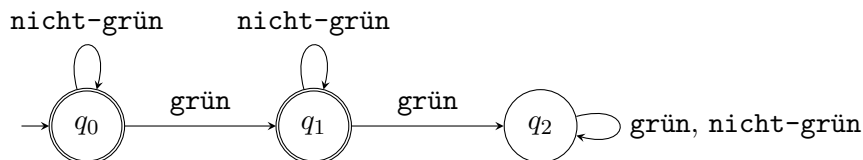
**Def. 2.1** (DEA): Ein *deterministischer endlicher Automat (DEA)*, ( $\text{DFA} \triangleq$  deterministic finite automaton) ist ein 5-Tupel

$$\mathcal{A} = (\Sigma, Q, \delta, q^{\text{init}}, F).$$

Dabei ist

- $\Sigma$  ein Alphabet,
- $Q$  eine *endliche* Menge, deren Elemente wir *Zustände* nennen,
- $\delta : Q \times \Sigma \rightarrow Q$  eine Funktion, die wir *Transitionsfunktion* nennen,
- $q^{\text{init}} \in Q$  ein Zustand, den wir *Startzustand* nennen und
- $F \subseteq Q$  eine Teilmenge der Zustände, deren Elemente wir *akzeptierende Zustände* nennen.  $\diamond$

DEAs lassen sich auch graphisch darstellen. Dabei gibt man für den Automaten einen gerichteten Graphen an. Die Knoten des Graphen sind die Zustände, und mit Zeichen beschriftete Kanten zeigen, welchen Zustandsübergang die Transitionsfunktion für das nächste Zeichen erlaubt. Der Startzustand ist mit einem unbeschrifteten Pfeil markiert, akzeptierende Zustände sind doppelt eingekreist. Hier ist die graphische Darstellung von  $A_{\text{Macaron}}$  aus Beispiel 2.1:



Die folgenden beiden Definitionen erlauben uns mit Hilfe eines DEA eine Sprache zu charakterisieren.

**Def. 2.2:** Die *induktive Erweiterung* von  $\delta : Q \times \Sigma \rightarrow Q$  auf Wörter,  $\tilde{\delta} : Q \times \Sigma^* \rightarrow Q$ , ist (induktiv) definiert durch

1.  $\tilde{\delta}(q, \varepsilon) = q$  (Wortende erreicht)
2.  $\tilde{\delta}(q, aw) = \tilde{\delta}(\delta(q, a), w)$  (Rest im Folgezustand verarbeiten)  $\diamond$

**Def. 2.3:** Sei  $\mathcal{A} = (\Sigma, Q, \delta, q^{\text{init}}, F)$ . Ein Wort  $w \in \Sigma^*$  wird von  $\mathcal{A}$  *akzeptiert*, falls  $\tilde{\delta}(q^{\text{init}}, w) \in F$ . Die *von  $\mathcal{A}$  akzeptierte Sprache*, geschrieben  $L(\mathcal{A})$ , ist die Menge aller Wörter, die von  $\mathcal{A}$  akzeptiert werden. D.h.,

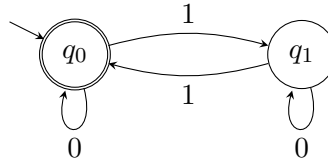
$$L(\mathcal{A}) = \{w \in \Sigma^* \mid \tilde{\delta}(q^{\text{init}}, w) \in F\}.$$

Eine durch einen DEA akzeptierte Sprache heißt *regulär*.  $\diamond$

**Bsp. 2.2:** Ein möglicher DEA für die Sprache

$$L_{\text{even}} = \{w \in \{0, 1\}^* \mid \text{die Anzahl der Einsen in } w \text{ ist gerade.}\}$$

aus der Einleitung dieses Kapitels hat die folgende graphische Repräsentation.



Frage: Gegeben seien zwei reguläre Sprachen  $L_1, L_2$  über einem gemeinsamen Alphabet  $\Sigma$ ; ist dann auch der Schnitt  $L_1 \cap L_2$  eine reguläre Sprache? Wir beantworten diese Frage mit dem folgenden Satz.

**Satz 2.1:** Reguläre Sprachen sind abgeschlossen unter der Schnittoperation. (D.h. für zwei gegebene reguläre Sprachen  $L_1, L_2$  über  $\Sigma$  ist auch der Schnitt  $L_1 \cap L_2$  eine reguläre Sprache.)

BEWEIS: <sup>6</sup> Da  $L_1$  und  $L_2$  regulär sind, gibt es zwei DEAs  $A_1 = (\Sigma, Q_1, \delta_1, q_1^{\text{init}}, F_1)$  und  $A_2 = (\Sigma, Q_2, \delta_2, q_2^{\text{init}}, F_2)$  mit  $L_1 = L(A_1)$  und  $L_2 = L(A_2)$ . Wir konstruieren nun zunächst den *Produktautomaten für Schnitt*  $A_\cap = (\Sigma, Q_\cap, \delta_\cap, q_\cap^{\text{init}}, F_\cap)$  wie folgt.

$$\begin{aligned} Q_\cap &= Q_1 \times Q_2 \\ \delta_\cap((q_1, q_2), a) &= (\delta_1(q_1, a), \delta_2(q_2, a)), \quad \text{für alle } a \in \Sigma \\ q_\cap^{\text{init}} &= (q_1^{\text{init}}, q_2^{\text{init}}) \\ F_\cap &= F_1 \times F_2 \end{aligned}$$

Anschließend zeigen wir, dass  $L(A_\cap) = L(A_1) \cap L(A_2)$  gilt. Hierfür zeigen wir zunächst via Induktion über die Länge von  $w$ , dass für alle  $w \in \Sigma^*$ , für alle  $q_1 \in Q_1$  und für alle  $q_2 \in Q_2$  die folgende Gleichung gilt.

$$\tilde{\delta}_\cap((q_1, q_2), w) = (\tilde{\delta}_1(q_1, w), \tilde{\delta}_2(q_2, w))$$

<sup>6</sup>Dieser erste Beweis ist außergewöhnlich detailliert. Im den folgenden Beweisen werden wir einfache Umformungen zusammenfassen.

Der Induktionsanfang für  $n = 0$  folgt dabei direkt aus Def. 2.2, da  $\varepsilon$  das einzige Wort der Länge 0 ist.

$$\tilde{\delta}_\cap((q_1, q_2), \varepsilon) = (q_1, q_2)$$

Den Induktionsschritt  $n \rightsquigarrow n+1$  zeigen wir mit Hilfe der folgenden Umformungen, wobei  $a \in \Sigma$  ein beliebiges Zeichen und  $w \in \Sigma^n$  ein beliebiges Wort der Länge  $n$  ist.

$$\begin{aligned} \tilde{\delta}_\cap((q_1, q_2), aw) &\stackrel{\text{Def. 2.2}}{=} \tilde{\delta}_\cap(\delta_\cap((q_1, q_2), a), w) \\ &\stackrel{\text{Def. } \delta_\cap}{=} \tilde{\delta}_\cap((\delta_1(q_1, a), \delta_2(q_2, a)), w) \\ &\stackrel{\text{I.V.}}{=} (\tilde{\delta}_1(\delta_1(q_1, a), w), \tilde{\delta}_2(\delta_2(q_2, a), w)) \\ &\stackrel{\text{Def. 2.2}}{=} (\tilde{\delta}_1(q_1, aw), \tilde{\delta}_2(q_2, aw)) \end{aligned}$$

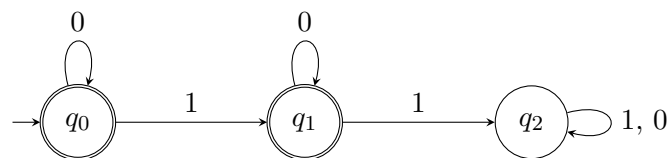
Schließlich zeigen wir  $L(A_\cap) = L(A_1) \cap L(A_2)$  mit Hilfe der folgenden Umformungen für ein beliebiges  $w \in \Sigma^*$ .

$$\begin{aligned} w \in L(A_\cap) &\stackrel{\text{Def. 2.3}}{\text{gdw}} \tilde{\delta}_\cap(q_\cap^{\text{init}}, w) \in F_\cap \\ &\stackrel{\text{Def. } q_\cap^{\text{init}}}{\text{gdw}} \tilde{\delta}_\cap((q_1^{\text{init}}, q_2^{\text{init}}), w) \in F_\cap \\ &\stackrel{\text{gdw}}{=} (\tilde{\delta}_1(q_1^{\text{init}}, w), \tilde{\delta}_2(q_2^{\text{init}}, w)) \in F_\cap \\ &\stackrel{\text{Def. } F_\cap}{\text{gdw}} \tilde{\delta}_1(q_1^{\text{init}}, w) \in F_1 \text{ und } \tilde{\delta}_2(q_2^{\text{init}}, w) \in F_2 \\ &\stackrel{\text{Def. 2.3}}{\text{gdw}} w \in L(A_1) \text{ und } w \in L(A_2) \end{aligned}$$

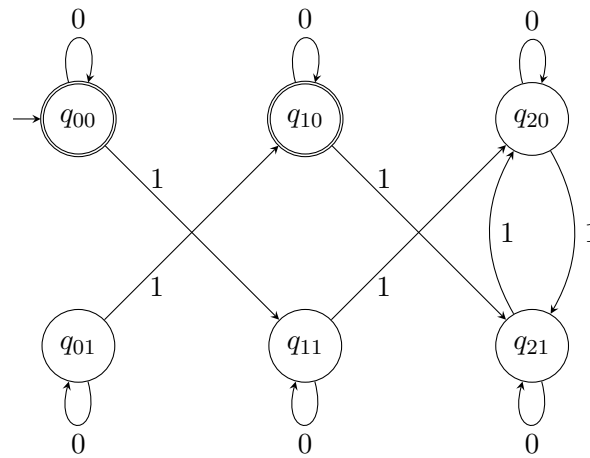
□

Vorlesung:  
25.10.17

Im folgenden Beispiel sei  $A_1$  der DEA über dem Alphabet  $\Sigma = \{0, 1\}$ , dessen graphische Repräsentation nahezu mit  $A_{\text{Macaron}}$  identisch ist.



**Bsp. 2.3:** Der Produktautomat für Schnitt von  $A_1$  und  $A_{\text{even}}$  hat die folgende graphische Repräsentation, wobei wir um Platz zu sparen „ $q_{ij}$ “ statt „ $(q_i, q_j)$ “ schreiben.



## 2.2 Minimierung endlicher Automaten

Beobachtung: Der Zustand  $q_{01}$  im Beispiel 2.3 scheint nutzlos. Wir charakterisieren diese „Nutzlosigkeit“ formal wie folgt.

**Def. 2.4:** Ein Zustand  $q \in Q$  heißt *erreichbar*, falls ein  $w \in \Sigma^*$  existiert, sodass  $\hat{\delta}(q^{\text{init}}, w) = q$ .  $\diamond$

Bemerkung: Die Menge der erreichbaren Zustände kann mit dem folgenden Verfahren in  $O(|Q| * |\Sigma|)$  berechnet werden.

- Fasse  $\mathcal{A}$  als Graph auf.
- Wende Tiefensuche an und markiere dabei alle besuchten Zustände.
- Die markierten Zustände bilden die Menge der erreichbaren Zustände.

Beobachtung: Auch nach dem Entfernen der nicht erreichbaren Zustände  $q_{01}$  und  $q_{10}$  scheint der DEA aus Bsp. 2.3 unnötig groß zu sein. Das Verhalten des DEA in den Zuständen  $q_{11}$ ,  $q_{20}$  und  $q_{21}$  ist sehr ähnlich. Wir charakterisieren diese „Ähnlichkeit“ formal wie folgt.

**Def. 2.5:** Wir nennen zwei Zustände  $p, q \in Q$  eines DEA *äquivalent*, geschrieben  $p \equiv q$ , falls

$$\forall w \in \Sigma^* : \tilde{\delta}(p, w) \in F \text{ gdw } \tilde{\delta}(q, w) \in F \quad \diamond$$

**Bsp. 2.4:** Für Bsp. 2.3 gilt: Die Zustände  $q_{11}$ ,  $q_{20}$  und  $q_{21}$  sind paarweise äquivalent. Die Zustände  $q_{00}$  und  $q_{10}$  sind äquivalent. Keine weiteren Zustandspaare sind äquivalent.

Geschrieben als Menge von Paaren sieht die Relation  $\equiv \subseteq Q \times Q$  also wie folgt aus:

$$\{(q_{00}, q_{10}), (q_{10}, q_{00}), (q_{11}, q_{20}), (q_{20}, q_{11}), (q_{20}, q_{21}), (q_{21}, q_{20}), (q_{21}, q_{11}), (q_{11}, q_{21})\}$$

### 2.2.1 Exkurs: Äquivalenzrelationen

Sie haben Äquivalenzrelationen bereits in „Mathematik II für Studierende der Informatik“<sup>7</sup> kennengelernt. Dieser kurze Exkurs wiederholt die für unsere Vorlesung relevanten Definitionen. Sei  $X$  eine beliebige Menge. Eine *Relation*  $R$  über  $X$  ist eine Teilmenge des Produkts  $X \times X$  (d.h.  $R \subseteq X \times X$ ).

Eine Relation  $R \subseteq X \times X$  heißt

- *reflexiv*, wenn  $\forall x \in X: (x, x) \in R$ ,
- *symmetrisch*, wenn  $\forall x, y \in X: (x, y) \in R \Rightarrow (y, x) \in R$ ,
- *transitiv*, wenn  $\forall x, y, z \in X: (x, y) \in R \wedge (y, z) \in R \Rightarrow (x, z) \in R$ .

**Bsp. 2.5:** Im Folgenden interessieren wir uns nur Relationen, die alle drei Eigenschaften erfüllen, aber die folgenden Beispiele sollen helfen, sich mit diesen Eigenschaften vertraut zu machen.

	reflexiv	symmetrisch	transitiv
„gewinnt“ bei Schere, Stein, Papier	nein	nein	nein
$(\mathbb{N}, <)$	nein	nein	ja
$(\mathbb{N}, \neq)$	nein	ja	nein
die leere Relation	nein	ja	ja
$\{(a, b) \in \mathbb{Z} \times \mathbb{Z} \mid a - b \leq 3\}$	ja	nein	nein
$(\mathbb{N}, \leq)$	ja	nein	ja
direkte genetische Verwandtschaft	ja	ja	nein
logische Äquivalenz von Formeln	ja	ja	ja

Bemerkung: Wir können kein Beispiel für eine nicht leere, symmetrische, transitive Relation finden, die nicht reflexiv ist. Für nicht leere Relationen folgt Reflexivität bereits aus Symmetrie und Transitivität:  $(a, b) \in R \xRightarrow{\text{sym}} (b, a) \in R \xRightarrow{\text{trans}} (a, a) \in R$ .

<sup>7</sup><http://home.mathematik.uni-freiburg.de/junker/ss17/matheII.html>

**Def. 2.6:** Eine Äquivalenzrelation  $R$  ist eine Relation, die reflexiv, symmetrisch und transitiv ist.

Für eine Äquivalenzrelation  $R$  und ein  $x \in X$  nennen wir die Menge  $\{y \in X \mid (y, x) \in R\}$  die *Äquivalenzklasse* von  $x$  und verwenden die Notation  $[x]_R$  für diese Menge. Wenn aus dem Kontext klar ist, welche Relation gemeint ist, dürfen wir das Subskript  $\cdot_R$  auch weglassen und schreiben nur  $[x]$ .

Wenn wir eine Äquivalenzklasse mit Hilfe der Notation  $[x]_R$  beschreiben, nennen wir  $x$  den *Repräsentanten* dieser Äquivalenzklasse.

Wir nennen die Anzahl der Äquivalenzklassen von  $R$  den *Index* von  $R$ . ◇

Zwei Fakten über eine beliebige Äquivalenzrelation  $R$  (ohne Beweis).

**Fakt 1** Die Äquivalenzklassen von  $R$  sind paarweise disjunkt.

**Fakt 2** Die Vereinigung aller Äquivalenzklassen ist die Menge  $X$ .

Hiermit endet der Exkurs zu Äquivalenzrelationen; wir wollen mit diesem Wissen die oben definierte Relation  $\equiv \subseteq Q \times Q$  genauer analysieren.

**Lemma 2.2:** Die Relation „ $\equiv$ “ ist eine *Äquivalenzrelation*.

BEWEIS: Die Relation  $\equiv$  ist offensichtlich reflexiv. Die Symmetrie und Transitivität von  $\equiv$  folgt aus der Symmetrie und Transitivität der logischen Interpretation von „genau dann, wenn“ (gdw). □

**Bsp. 2.6:** Für den DEA aus Bsp. 2.3 hat die Relation  $\equiv$  drei Äquivalenzklassen.<sup>8</sup>

$$\begin{aligned} [q_{00}] &= \{q_{00}, q_{10}\}, \\ [q_{01}] &= \{q_{01}\}, \\ [q_{11}] &= \{q_{11}, q_{20}, q_{21}\} \end{aligned}$$

Idee: „Verschmelze“ alle Zustände aus einer Äquivalenzklasse zu einem einzigen Zustand. Bedenken: Bei einem DEA hat jeder Zustand hat für jedes Zeichen einen Nachfolger. Wenn wir Zustände verschmelzen, könnte es mehrere Nachfolger geben und das Resultat wäre kein wohldefinierter DEA mehr.

Das folgende Lemma zeigt, dass unsere Bedenken nicht gerechtfertigt sind. Sind zwei Zustände äquivalent, so sind auch für jedes Zeichen ihre Nachfolger äquivalent.

---

<sup>8</sup>Den Zustand  $q_{11}$  als Repräsentanten für die dritte Äquivalenzklasse zu wählen ist eine völlig willkürliche Entscheidung. Wir könnten genauso gut  $q_{20}$  oder  $q_{21}$  wählen.

**Lemma 2.3:** Für alle  $p, q \in Q$  gilt:

$$p \equiv q \quad \Rightarrow \quad \forall a \in \Sigma : \delta(p, a) \equiv \delta(q, a)$$

BEWEIS:

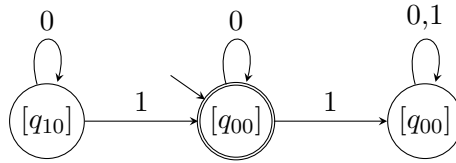
$$\begin{aligned} p \equiv q & \quad \text{gdw} \quad \forall w \in \Sigma^* : \tilde{\delta}(p, w) \in F \Leftrightarrow \tilde{\delta}(q, w) \in F \\ & \quad \text{gdw} \quad (p \in F \Leftrightarrow q \in F) \wedge \forall a \in \Sigma : \forall w \in \Sigma^* : \tilde{\delta}(p, aw) \in F \Leftrightarrow \tilde{\delta}(q, aw) \in F \\ & \quad \text{impliziert} \quad \forall a \in \Sigma : \forall w \in \Sigma^* : \tilde{\delta}(\delta(p, a), w) \in F \Leftrightarrow \tilde{\delta}(\delta(q, a), w) \in F \\ & \quad \text{gdw} \quad \forall a \in \Sigma : \delta(p, a) \equiv \delta(q, a) \end{aligned} \quad \square$$

Wir formalisieren das „Verschmelzen“ von Zuständen wie folgt.

**Def. 2.7:** Der Äquivalenzklassenautomat  $\mathcal{A}_{\equiv} = (Q_{\equiv}, \Sigma, \delta_{\equiv}, q_{\equiv}^{\text{init}}, F_{\equiv})$  zu einem DEA  $\mathcal{A} = (Q, \Sigma, \delta, q^{\text{init}}, F)$  ist bestimmt durch:

$$\begin{aligned} Q_{\equiv} &= \{[q] \mid q \in Q\} & \delta_{\equiv}([q], a) &= [\delta(q, a)] \\ q_{\equiv}^{\text{init}} &= [q^{\text{init}}] & F_{\equiv} &= \{[q] \mid q \in F\} \end{aligned} \quad \diamond$$

**Bsp. 2.7:** Der Äquivalenzklassenautomat  $\mathcal{A}_{\equiv}$  zum DEA aus Bsp. 2.3 hat das folgende Zustandsdiagramm.



**Satz 2.4:** Der Äquivalenzklassenautomat ist wohldefiniert und  $L(\mathcal{A}_{\equiv}) = L(\mathcal{A})$ .

BEWEIS:

1. Wohldefiniert: Es gilt zu zeigen, dass  $\delta_{\equiv}([q], a) = [\delta(q, a)]$  nicht abhängig von der Wahl des Repräsentanten  $q \in [q]$  ist. Dies folgt direkt aus Lemma 2.3.
2.  $L(\mathcal{A}) = L(\mathcal{A}_{\equiv})$ : Zunächst zeigen wir via Induktion über die Länge von  $w$ , dass für alle  $w \in \Sigma^*$  und alle  $q \in Q$  die folgende Äquivalenz gilt.

$$\tilde{\delta}(q, w) \in F \Leftrightarrow \tilde{\delta}_{\equiv}([q], w) \in F_{\equiv}$$

$$\text{I.A. } (n = 0, \text{ also } w = \varepsilon): \tilde{\delta}(q, \varepsilon) = q \in F \Leftrightarrow \tilde{\delta}_{\equiv}([q], \varepsilon) = [q] \in F_{\equiv}$$

I.S.:  $(n \rightsquigarrow n + 1)$

$$\begin{aligned}
 \tilde{\delta}(q, aw) \in F &\iff \tilde{\delta}(\delta(q, a), w) \in F \\
 &\stackrel{I.V.}{\iff} \tilde{\delta}_{\equiv}([\delta(q, a)], w) \in F_{\equiv} \\
 &\iff \tilde{\delta}_{\equiv}(\delta_{\equiv}([q], a), w) \in F_{\equiv} \\
 &\iff \tilde{\delta}_{\equiv}([q], aw) \in F_{\equiv}
 \end{aligned}$$

Mit Hilfe dessen zeigen wir nun  $L(\mathcal{A}) = L(\mathcal{A}_{\equiv})$ . Sei  $w \in \Sigma^*$ .

$$\begin{aligned}
 w \in L(\mathcal{A}) &\iff \tilde{\delta}(q^{\text{init}}, w) \in F \\
 &\iff \tilde{\delta}_{\equiv}([q^{\text{init}}], w) \in F_{\equiv} \\
 &\iff w \in L(\mathcal{A}_{\equiv})
 \end{aligned}$$

□

In den Übungen werden wir ein Verfahren mit Laufzeit  $O(|Q|^4 \cdot |\Sigma|)$  zur Konstruktion des Äquivalenzklassenautomaten kennenlernen. Es gibt aber auch schnellere Verfahren. Mit dem Algorithmus von Hopcroft kann  $\mathcal{A}_{\equiv}$  in  $O(|Q||\Sigma| \log |Q|)$  erzeugt werden.

Wir werden später ( $\rightarrow$  Satz von Myhill-Nerode) sehen, dass  $\mathcal{A}_{\equiv}$  der kleinste DEA ist, der  $L(\mathcal{A})$  akzeptiert.

**Def. 2.8:** Eine Äquivalenzrelation  $R \subseteq \Sigma^* \times \Sigma^*$  heißt *rechtskongruent*, falls

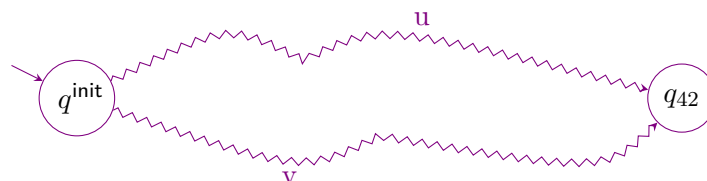
Vorlesung:  
27.10.17

$$(u, v) \in R \quad \Rightarrow \quad \forall w \in \Sigma^* : (u \cdot w, v \cdot w) \in R.$$

◇

**Motivation:**<sup>9</sup> In weiteren Verlauf dieses Unterkapitels wollen wir reguläre Sprachen charakterisieren. Informell ist die Idee dieser Charakterisierung die folgende:

Angenommen wir möchten für eine gegebene Sprache  $L$  einen DEA  $\mathcal{A}$  konstruieren. Angenommen wir haben bereits einen ersten Entwurf gemacht und in diesem führen sowohl das Wort  $u \in \Sigma^*$  als auch das Wort  $v \in \Sigma^*$  in den Zustand  $q_{42}$  (Formal:  $\tilde{\delta}(q^{\text{init}}, u) = \tilde{\delta}(q^{\text{init}}, v) = q_{42}$ ).



**Abb. 2:** Schematische Darstellung eines DEA, mit  $\tilde{\delta}(q^{\text{init}}, u) = \tilde{\delta}(q^{\text{init}}, v) = q_{42}$

<sup>9</sup>Dieser lilafarbene Abschnitt wurde aufgrund der Fragestunde vom 22.12.2017 nachträglich eingefügt.



- Angenommen es gibt ein Wort  $w \in \Sigma^*$ , sodass die Äquivalenz  $uw \in L \Leftrightarrow vw \in L$  nicht gilt dann wissen wir dass unser Entwurf noch einen Fehler enthält und wir dafür sorgen müssen, dass der DEA nicht sowohl von  $u$  als auch von  $v$  in  $q_{42}$  überführt wird.
- Angenommen es gilt für alle Wörter  $w \in \Sigma^*$  dass  $uw \in L \Leftrightarrow vw \in L$ , dann wissen wir dass es eine gute Idee war den Automaten mit  $u$  und  $v$  in den gleichen Zustand zu führen.

Angenommen wir finden nicht nur für zwei Wörter  $u, v$  ein Wort  $w$  sodass die Äquivalenz  $uw \in L \Leftrightarrow vw \in L$  nicht gilt sondern wir finden soch ein  $w$  für jedes Paar aus einer unendlich großen Menge von Wörtern  $U$ .

$$(\text{Formal: } \forall u, v \in U : u \neq v \Rightarrow \exists w^{uv} \in \Sigma^* : \neg(uw^{uv} \in L \Leftrightarrow vw^{uv} \in L))$$

Dann wissen wir dass all unsere Reparaturversuche hoffnungslos sind, denn der DEA  $\mathcal{A}$  bräuchte für jedes  $u \in U$  einen eigenen Zustand doch ein DEA darf nur endlich viele Zustände haben. Die Sprache  $L$  kann also nicht regulär sein.

Im weiteren Verlauf des Unterkapitels werden wir genau diese Überlegungen formal ausarbeiten.

**Bsp. 2.8:** Für einen DEA  $\mathcal{A}$  definiere

$$R_{\mathcal{A}} = \{(u, v) \mid \tilde{\delta}(q^{\text{init}}, u) = \tilde{\delta}(q^{\text{init}}, v)\}.$$

**Beobachtung 1**  $R_{\mathcal{A}}$  ist eine Äquivalenzrelation.

Dies folgt daraus, dass „ $=$ “ eine Äquivalenzrelation ist.

**Beobachtung 2**  $R_{\mathcal{A}}$  ist rechtskongruent.

Dies wird in den Übungen bewiesen.

**Beobachtung 3** Wir haben pro Zustand, der von  $q^{\text{init}}$  erreichbar ist, genau eine Äquivalenzklasse. Der Index von  $R_{\mathcal{A}}$  ist also die Anzahl der erreichbaren Zustände.

Für den DEA aus Bsp. 2.3 hat  $R_{\mathcal{A}}$  die folgenden Äquivalenzklassen.

$$\begin{aligned} [\varepsilon] &= \{0^n \mid n \in \mathbb{N}\} \\ [1] &= \{0^n 10^m \mid n, m \in \mathbb{N}\} \\ [11] &= \{w \mid \text{Anzahl von Einsen in } w \text{ ist gerade und } \geq 2\} \\ [111] &= \{w \mid \text{Anzahl von Einsen in } w \text{ ist ungerade und } \geq 2\} \end{aligned}$$

**Def. 2.9:** Für eine Sprache  $L \subseteq \Sigma^*$  ist die *Nerode-Relation* wie folgt definiert.

$$R_L = \{(u, v) \mid \forall w \in \Sigma^* : uw \in L \Leftrightarrow vw \in L\} \quad \diamond$$

**Beobachtung 1** Die Nerode-Relation ist eine Äquivalenzrelation.

Dies folgt daraus, dass „ $\Leftrightarrow$ “ (Bimplikation, „genau dann, wenn“) eine Äquivalenzrelation ist.

**Beobachtung 2** Die Nerode-Relation ist rechtskongruent.

BEWEIS: Sei  $(u, v) \in R_L$ . Zeige  $\forall w \in \Sigma^* : (uw, vw) \in R_L$ . Wir führen diesen Beweis via Induktion über die Länge von  $w$ .

I.A. ( $n = 0$ ) Für  $w = \varepsilon$  ist  $(u\varepsilon, v\varepsilon) = (u, v) \in R_L$ .

I.S. ( $n \rightsquigarrow n + 1$ ) Betrachte mit  $w = w'a$  ein beliebiges Wort der Länge  $n$ . Nach Induktionsvoraussetzung ist dann auch  $(uw', vw') \in R_L$ .

$$\begin{array}{ll}
 (uw', vw') \in R_L & \begin{array}{l} \text{Def. } R_L \\ \text{gdw} \end{array} \quad \forall z \in \Sigma^* : uw'z \in L \Leftrightarrow vw'z \in L \\
 & \begin{array}{l} \text{zerlege } z=az' \\ \text{impliziert} \end{array} \quad \forall a \in \Sigma, z' \in \Sigma^* : uw'az' \in L \Leftrightarrow vw'az' \in L \\
 & \begin{array}{l} \text{Def. } R_L \\ \text{impliziert} \end{array} \quad (uw'a, vw'a) \in R_L \quad \square
 \end{array}$$

**Bsp. 2.9:** Sei  $\Sigma = \{0, 1\}$ . Die Sprache  $L = \{w \in \Sigma^* \mid \text{vorletztes Zeichen von } w \text{ ist } 1\}$  hat die folgenden Äquivalenzklassen bezüglich der Nerode-Relation.

$$\begin{aligned}
 [\varepsilon] &= \{w \mid w \text{ endet mit } 00\} \cup \{\varepsilon, 0\} \\
 [1] &= \{w \mid w \text{ endet mit } 01\} \cup \{1\} \\
 [10] &= \{w \mid w \text{ endet mit } 10\} \\
 [11] &= \{w \mid w \text{ endet mit } 11\}
 \end{aligned}$$

**Bsp. 2.10:** Für ein beliebiges Alphabet  $\Sigma$  gilt:

1. Die Sprache  $L = \{\varepsilon\}$  hat genau zwei Äquivalenzklassen bezüglich der Nerode-Relation. Eine Äquivalenzklasse ist  $\{\varepsilon\}$ , die andere ist  $\Sigma^+$ .
2. Die Sprache  $L = \{\}$  hat genau eine Äquivalenzklasse (nämlich  $\Sigma^*$ ) bezüglich der Nerode-Relation.

**Bsp. 2.11:** Sei  $\Sigma = \{0, 1\}$ . Die Sprache  $L_{\text{centered}} = \{0^n 10^n \mid n \in \mathbb{N}\}$  hat bezüglich der Nerode-Relation die folgende Menge von Äquivalenzklassen:

$$\{[w'] \mid w' \text{ ist Präfix eines Worts } w \in L_{\text{centered}}\} \cup \{[11]\}$$

Dabei gilt, dass für je zwei verschiedene  $k \in \mathbb{N}$  auch die Äquivalenzklassen  $[0^k 1]$  verschieden sind. Somit gibt es unendlich viele Äquivalenzklassen.

Bemerkung: Die Äquivalenzklasse  $[11]$  enthält alle Wörter, die kein Präfix eines Worts aus  $L_{\text{centered}}$  sind.

**Bsp.:** Die Sprache  $L = \{w \in \{a, b\}^* \mid \#_a(w) > \#_b(w)\}$  hat bezüglich der Nerode-Relation die folgende Menge von Äquivalenzklassen:<sup>10</sup>

$$\{[w] \mid \#_a(w) - \#_b(w) = k, k \in \mathbb{Z}\}$$

**Satz 2.5** (Myhill und Nerode): Die folgenden Aussagen sind äquivalent:

1.  $L \subseteq \Sigma^*$  wird von einem DEA akzeptiert.
2.  $L$  ist Vereinigung die von Äquivalenzklassen einer rechtskongruenten Äquivalenzrelation mit *endlichem* Index.
3. Die Nerode-Relation  $R_L$  hat *endlichen* Index.

BEWEIS: Wir beweisen die paarweise Äquivalenz in drei Schritten:

$$(1) \Rightarrow (2), \quad (2) \Rightarrow (3) \quad \text{und} \quad (3) \Rightarrow (1)$$

**(1)  $\Rightarrow$  (2)** Sei  $\mathcal{A}$  ein DEA mit

$$L(\mathcal{A}) = \{w \mid \tilde{\delta}(q^{\text{init}}, w) \in F\} = \bigcup_{q \in F} \{w \mid \tilde{\delta}(q^{\text{init}}, w) = q\}.$$

Nun sind  $\{w \mid \tilde{\delta}(q^{\text{init}}, w) = q\}$  genau die Äquivalenzklassen der Relation  $R_{\mathcal{A}}$  aus Bsp. 2.8, einer rechtskongruenten Äquivalenzrelation. Der Index ist die Anzahl der erreichbaren Zustände und somit endlich:  $\text{Index}(R_{\mathcal{A}}) \leq |Q| < \infty$ .

**(2)  $\Rightarrow$  (3)** Sei  $R$  einer rechtskongruente Äquivalenzrelation mit endlichem Index, sodass  $L$  die Vereinigung von  $R$ -Äquivalenzklassen ist.

Es genügt zu zeigen, dass die Nerode-Relation  $R_L$  eine Obermenge von  $R$  ist.<sup>11</sup>

$$\begin{aligned} (u, v) \in R &\Rightarrow u \in L \Leftrightarrow v \in L, \quad \text{da } L \text{ Vereinigung von Äquivalenzklassen ist} \\ &\Rightarrow \forall w \in \Sigma^* : uw \in L \Leftrightarrow vw \in L, \quad \text{da } R \text{ rechtskongruent} \\ &\Rightarrow (u, v) \in R_L, \quad \text{nach Definition der Nerode-Relation} \end{aligned}$$

Es gilt also  $R \subseteq R_L$  und somit  $\text{Index}(R_L) \leq \text{Index}(R) < \infty$ .

<sup>10</sup>Dieses Beispiel wurde aufgrund der Fragestunde vom 22.12.2017 nachträglich eingefügt.

<sup>11</sup>Zur Erklärung: Falls  $R \subseteq R_L$ , dann gilt  $\text{Index}(R) \geq \text{Index}(R_L)$ . Intuitiv: Je mehr Elemente eine Äquivalenzrelation  $R$  enthält, desto mehr Elemente sind bzgl. dieser Relation äquivalent, d.h. desto weniger unterschiedliche Äquivalenzklassen gibt es.

**(3)  $\Rightarrow$  (1)** Gegeben  $R_L$ , konstruiere  $\mathcal{A} = (Q, \Sigma, \delta, q^{\text{init}}, F)$

- $Q = \{[w]_{R_L} \mid w \in \Sigma^*\}$  endlich, da  $\text{Index}(R_L)$  endlich
- $\delta([w], a) = [wa]$  wohldefiniert, da  $R_L$  rechtskongruent
- $q^{\text{init}} = [\varepsilon]$
- $F = \{[w] \mid w \in L\}$

Wir wollen nun  $L(\mathcal{A}) = L$  zeigen. Dafür beweisen wir zunächst via Induktion über die Länge von  $w$  die folgende Eigenschaft.

$$\forall w \in \Sigma^* : \forall v \in \Sigma^* : \tilde{\delta}([v], w) = [v \cdot w]$$

I.A. ( $w = \varepsilon$ ):  $\tilde{\delta}([v], \varepsilon) = [v] = [v \cdot \varepsilon]$

I.S. Sei  $w = aw'$  beliebiges Wort der Länge  $n + 1$ .

$$\begin{aligned} \tilde{\delta}([v], aw') &= \tilde{\delta}(\delta([v], a), w') \\ &= \tilde{\delta}([v \cdot a], w') \\ &\stackrel{\text{I.V.}}{=} [v \cdot a \cdot w'] \\ &= [v \cdot \underbrace{aw'}_{=w}] \end{aligned}$$

Nun zeigen wir  $L(\mathcal{A}) = L$  wie folgt:

$$\begin{aligned} w \in L(\mathcal{A}) &\text{ gdw } \tilde{\delta}([\varepsilon], w) \in F \\ &\text{ gdw } [w] \in F, \quad (\text{via Induktion gezeigte Eigenschaft für } v = \varepsilon) \\ &\text{ gdw } w \in L \end{aligned} \quad \square$$

**Korollar 2.5:** Der im Beweisschritt (3)  $\Rightarrow$  (1) konstruierte Automat  $\mathcal{A}$  ist ein minimaler Automat (bzgl. der Zustandsanzahl) für eine reguläre Sprache  $L$ .  $\diamond$

Vorlesung:  
3.11.16

BEWEIS: Sei  $\mathcal{A}'$  ein beliebiger DEA mit  $L(\mathcal{A}') = L$ .

Aus „1  $\Rightarrow$  2“ wissen wir, dass  $\text{Index}(R_{\mathcal{A}'}) \leq |Q'|$  gilt.

Aus „2  $\Rightarrow$  3“ wissen wir, dass  $R_{\mathcal{A}'} \subseteq R_L$  und somit  $\text{Index}(R_L) \leq \text{Index}(R_{\mathcal{A}'})$  gilt.

In „3  $\Rightarrow$  1“ definieren wir  $A$ , sodass  $|Q| = \text{Index}(R_L) \leq \text{Index}(R_{\mathcal{A}'}) \leq |Q'|$  gilt.

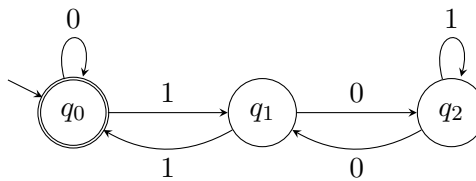
Für einen beliebigen DEA  $\mathcal{A}'$  ist  $|Q|$  also nie größer als  $|Q'|$ .  $\square$

### 2.3 Pumping Lemma (PL) für reguläre Sprachen

Welche interessanten Eigenschaften haben reguläre Sprachen?

Notation: Sei  $\text{bin} : \{0, 1\}^* \rightarrow \mathbb{N}$  die Decodierung von Bitstrings in natürliche Zahlen; z.B.  $\text{bin}(101) = 5$ ,  $\text{bin}(\varepsilon) = 0$ .

**Bsp. 2.12:** Betrachte den folgenden DEA, der die Sprache der Binärcodierung von durch drei teilbaren Zahlen akzeptiert:  $L = \{w \in \{0, 1\}^* \mid \text{bin}(w) \equiv_3 0\}$



Beobachtungen:

- Es gilt offensichtlich, dass  $11 \in L$ .
- Es gilt auch, dass  $1001 \in L$ .
- Der Automat hat eine Schleife bei  $\tilde{\delta}(q_1, 00) = q_1$ , die mehrfach „abgelaufen“ werden kann, ohne die Akzeptanz zu beeinflussen.
- Also gilt auch  $100001 \in L$ .
- Im Allgemeinen gilt  $\forall i \in \mathbb{N} : 1(00)^i 1 \in L$ .

Verdacht: Alle „langen“ Wörter lassen sich in der Mitte „aufpumpen“. Wir formalisieren diesen Verdacht im folgenden Lemma.

**Lemma 2.6** (Pumping Lemma): Sei  $L$  eine reguläre Sprache. Dann gilt:

$$\begin{aligned}
 &\exists n \in \mathbb{N}, n > 0 : \quad \forall z \in L, |z| \geq n : \\
 &\quad \exists u, v, w \in \Sigma^* : \\
 &\quad z = uvw, |uv| \leq n, |v| \geq 1 \\
 &\text{und } \forall i \in \mathbb{N} : uv^i w \in L
 \end{aligned}$$

BEWEIS: Sei  $\mathcal{A} = (\Sigma, Q, \delta, q^{\text{init}}, F)$  ein beliebiger DEA für  $L$ . Wähle  $n = |Q|$  und  $z \in L$  beliebig mit  $|z| \geq n$ .

Beim Lesen von  $z$  durchläuft  $\mathcal{A}$  genau  $\overbrace{|z|}^{\geq n+1} + 1$  Zustände. Somit gibt es mindestens einen Zustand  $q \in Q$ , der mehrmals besucht wird (Schubfachprinzip).

Wähle den Zustand  $q$ , der als Erster zweimal besucht wird.

$$\begin{aligned} \text{Nun gilt: } \exists u : \tilde{\delta}(q^{\text{init}}, u) = q & \quad u \text{ Präfix von } z \\ \exists v : \tilde{\delta}(q, v) = q & \quad uv \text{ Präfix von } z \\ \exists w : \tilde{\delta}(q, w) \in F & \quad uvw = z \\ |v| \geq 1 & \\ |uv| \leq n & \quad \text{da } q \text{ als Erster zweimal besucht wird} \end{aligned}$$

$$\begin{aligned} \text{Es folgt für beliebiges } i \in \mathbb{N} : \tilde{\delta}(q^{\text{init}}, uv^i w) &= \tilde{\delta}(q, v^i w) \\ &= \tilde{\delta}(q, w) \quad \text{denn } \forall i : \tilde{\delta}(q, v^i) = q \\ &\in F \end{aligned} \quad \square$$

**Bsp. 2.13:** Die Sprache  $L_{\text{centered}} = \{0^n 10^n \mid n \in \mathbb{N}\}$  ist nicht regulär.

Wir geben hierfür einen Widerspruchsbeweis mit Hilfe des Pumping Lemmas PL.

Sei  $n$  die Konstante aus dem PL. Wähle  $z = 0^n 10^n$ . (Gültige Wahl, da  $|z| = 2n + 1 \geq n$ )  
Laut PL existieren  $u, v, w$ , sodass  $z = uvw$  mit  $|v| \geq 1, |uv| \leq n$  und  $\forall i \in \mathbb{N} : uv^i w \in L$ .  
Nach Wahl von  $z$  gilt nun

- $uv = 0^m$  mit  $m \leq n$
- $v = 0^k$  mit  $k \geq 1$
- $w = 0^{n-m} 10^n$

Betrachte  $uv^2w = 0^{m-k} 0^{2k} 0^{n-m} 10^n = 0^{n+k} 10^n \notin L$ . Widerspruch! Somit ist  $L$  nicht regulär.

Zur Illustration:

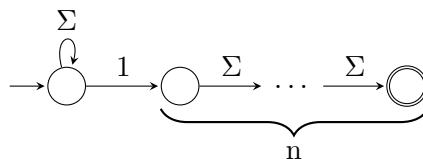
$$\begin{array}{c} \underbrace{0 \dots 0}_n \underbrace{10 \dots 0}_n \\ | \text{---} u \text{---} | \text{---} v \text{---} | \text{---} w \text{---} | \end{array}$$

## 2.4 Nichtdeterministischer endlicher Automat (NEA)

Aufgabe: Konstruiere für eine natürliche Zahl  $n \in \mathbb{N}$  einen DEA für die folgende Sprache.

$$L_n = \{w \in \{0, 1\}^* \mid \text{das } n\text{-letzte Symbol von } w \text{ ist } 1\}$$

Naiver Lösungsversuch:



**Abb. 3:** Nichtdeterministischer Automat für  $L_n$

Problem: Das Zustandsdiagramm beschreibt keinen DEA: Der Startzustand hat zwei ausgehende Kanten für 1.

Untersuche die Sprache mit Hilfe der Nerode-Relation. Beobachtung: Je zwei Wörter der Länge  $n$  sind in unterschiedlichen Äquivalenzklassen. Es gibt also mindestens  $2^n$  Äquivalenzklassen; aus Korollar 2.5 wissen wir, dass ein minimaler DEA, der  $L_n$  akzeptiert, mindestens  $2^n$  Zustände haben muss.

Idee: Definiere eine neue Art von Automaten, bei dem ein Zustand pro Zeichen mehrere Nachfolger haben darf.

**Def. 2.10 (NEA):** Ein *nichtdeterministischer endlicher Automat (NEA)*, ( $\text{NFA} \triangleq \text{non-deterministic finite automaton}$ ) ist ein 5-Tupel

$$\mathcal{N} = (\Sigma, Q, \delta, q^{\text{init}}, F).$$

Dabei ist

- $\Sigma$  ein Alphabet,
- $Q$  eine *endliche* Menge, deren Elemente wir *Zustände* nennen,
- $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$  eine Funktion, die wir *Transitionsfunktion* nennen,
- $q^{\text{init}} \in Q$  ein Zustand, den wir *Startzustand* nennen und
- $F \subseteq Q$  eine Teilmenge der Zustände, deren Elemente wir *akzeptierende Zustände* nennen.  $\diamond$

Bemerkung: Die Definition des NEA unterscheidet sich vom DEA also nur in der Transitionsfunktion. Beim DEA ist der Bildbereich der Transitionsfunktion die Menge der Zustände  $Q$ . Beim NEA ist der Bildbereich die Potenzmenge  $\mathcal{P}(Q)$  der Zustandsmenge  $Q$ . Analog zu DEAs werden wir auch NEAs mit Hilfe eines Zustandsdiagramms beschrieben. Zum Beispiel beschreibt Abb. 3 für jedes  $n \in \mathbb{N}$  einen NEA für die Sprache  $L_n$ .

Im Folgenden sei  $\mathcal{N}$  immer ein NEA.

**Def. 2.11:** Wir nennen eine Folge von Zuständen  $q_0 q_1 \dots q_n$  einen *Lauf von  $\mathcal{N}$  über  $w = a_1 \dots a_n$* , falls  $q_i \in \delta(q_{i-1}, a_i)$  für alle  $i$  mit  $1 \leq i \leq n$ . Wir nennen einen Lauf *initial*, falls  $q_0 = q^{\text{init}}$ . Wir nennen einen Lauf *akzeptierend*, falls  $q_n \in F$ .  $\diamond$

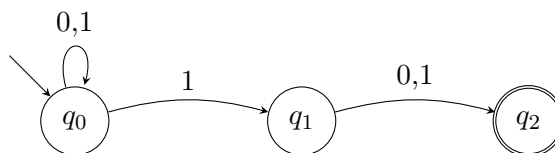
**Def. 2.12:** Ein Wort  $w \in \Sigma^*$  wird von  $\mathcal{N}$  *akzeptiert*, falls  $\mathcal{N}$  einen initialen und akzeptierenden Lauf über  $w$  hat. Die von  $\mathcal{N}$  akzeptierte Sprache ist die Menge der von  $\mathcal{N}$  akzeptierten Wörter, d.h.  $L(\mathcal{N}) = \{w \in \Sigma^* \mid \exists \text{ initialer, akzeptierender Lauf von } \mathcal{N} \text{ über } w\}$ .

◇

**Bsp. 2.14:** Der NEA für die Sprache

$$L_2 = \{w \in \{0,1\}^* \mid \text{das zweitletzte Zeichen von } w \text{ ist } 1\}$$

hat die folgende graphische Repräsentation.



Bemerkung: Die Frage, ob ein gegebenes Wort  $w$  akzeptiert wird (das „Wortproblem“), lässt sich für NEAs nicht mehr so leicht beantworten wie wir es von DEAs gewohnt sind. Ein sinnvolles Vorgehen scheint, jeden initialen Lauf zu betrachten, doch z.B. für das Wort 111 hat obiger NEA bereits drei verschiedene initiale Läufe:  $q_0q_0q_0$ ,  $q_0q_0q_1$ ,  $q_0q_0q_2$ .

Bemerkung: Die Definitionen von NEA und DEA in der Literatur sind nicht einheitlich. Es gibt äquivalente NEA-Definitionen, die statt der Transitionsfunktion  $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$  eine Transitionsrelation  $\delta \subseteq Q \times \Sigma \times Q$  verwenden. Es gibt alternative NEA-Definitionen, die eine Menge von Startzuständen erlauben. Alternativ könnte man auch zunächst den NEA einführen und den DEA als Spezialfall dessen definieren (Spezialfall: Das Bild der Transitionsfunktion ist einelementig für alle  $q \in Q$  und  $a \in \Sigma$ ).

Bemerkung: Zu jedem DEA  $\mathcal{A} = (\Sigma, Q, \delta, q^{\text{init}}, F)$  gibt es einen NEA, der die gleiche Sprache akzeptiert. Beispiel: der NEA  $\mathcal{N} = (\Sigma, Q, \delta_{\text{NEA}}, q^{\text{init}}, F)$  mit  $\delta_{\text{NEA}}(q, a) = \{\delta(q, a)\}$ , der sich von  $\mathcal{A}$  nur in der Transitionsfunktion unterscheidet.

**Satz 2.7** (Rabin und Scott): Zu jedem NEA  $\mathcal{N}$  mit  $n$  Zuständen gibt es einen DEA  $\mathcal{A}_{\mathcal{P}}$  mit  $2^n$  Zuständen, sodass  $L(\mathcal{A}_{\mathcal{P}}) = L(\mathcal{N})$  gilt.

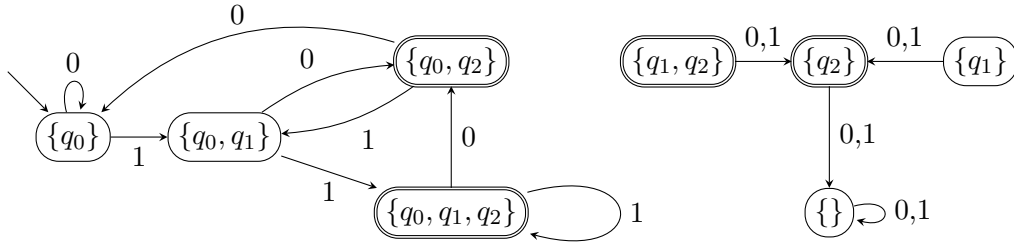
Zur Vorbereitung des Beweises machen wir zunächst die folgende Definition.



**Def. 2.13** (Potenzmengenautomat): Für einen gegebenen NEA  $\mathcal{N} = (\Sigma, Q, \delta, q^{\text{init}}, F)$  ist der Potenzmengenautomat  $\mathcal{A}_{\mathcal{P}}$  wie folgt definiert.

$$\begin{aligned} Q_{\mathcal{P}} &= \mathcal{P}(Q) \\ \delta_{\mathcal{P}}(p, a) &= \bigcup_{q \in p} \{\delta(q, a)\} \\ q_{\mathcal{P}}^{\text{init}} &= \{q^{\text{init}}\} \\ F_{\mathcal{P}} &= \{p \in Q_{\mathcal{P}} \mid p \cap F \neq \emptyset\} \end{aligned} \quad \diamond$$

**Bsp. 2.15:** Der Potenzmengenautomat für den NEA aus Bsp. 2.14 hat das folgende Zustandsdiagramm.



Die vier Zustände auf der rechten Seite sind nicht erreichbar.

**BEWEIS** (von Satz 2.7): Zeige  $L(\mathcal{A}_{\mathcal{P}}) = L(\mathcal{N})$ . Dafür beweisen wir zunächst via Induktion über die Länge von  $w$  die folgende Eigenschaft:

$\forall w \in \Sigma^* \forall p \in Q_{\mathcal{P}} \setminus \{\{\}\} \forall q \in Q :$

$$q \in \tilde{\delta}_{\mathcal{P}}(p, w) \Leftrightarrow \exists \underbrace{q_0, q_1, \dots, q_n}_{\text{Lauf}} \in Q, \text{ sodass } n = |w|, q_0 \in p \text{ und } q_n = q$$

I.A. ( $n = 0$ , also  $w = \varepsilon$ ): Gilt trivialerweise, da  $p \neq \{\}$ .

I.S. ( $n \rightsquigarrow n + 1$ ): Sei  $w = aw'$  ein beliebiges Wort der Länge  $n + 1$ .

$$\begin{aligned} q \in \tilde{\delta}_{\mathcal{P}}(p, w) &\Leftrightarrow q \in \tilde{\delta}_{\mathcal{P}}(\delta_{\mathcal{P}}(p, a), w') \\ &\stackrel{\text{I.V.}}{\Leftrightarrow} \exists \underbrace{q_1, q_2, \dots, q_{n+1}}_{\text{Lauf}} \in Q, \text{ sodass } n = |w'|, q_1 \in \delta_{\mathcal{P}}(p, a) \text{ und } q_{n+1} = q \\ &\Leftrightarrow \exists \underbrace{q_1, q_2, \dots, q_{n+1}}_{\text{Lauf}} \in Q, \text{ sodass } n = |w'|, \exists q_0 \in p : q_1 \in \delta(q_0, a) \text{ und } q_{n+1} = q \\ &\Leftrightarrow \exists \underbrace{q_0, q_1, q_2, \dots, q_{n+1}}_{\text{Lauf}} \in Q, \text{ sodass } n + 1 = |w|, q_0 \in p \text{ und } q_{n+1} = q \end{aligned}$$

Mit Hilfe dieser Eigenschaft zeigen wir nun die Gleichheit  $L(\mathcal{A}_{\mathcal{P}}) = L(\mathcal{N})$ .

$$\begin{aligned}
 w \in L(\mathcal{A}_{\mathcal{P}}) &\Leftrightarrow \tilde{\delta}_{\mathcal{P}}(q_{\mathcal{P}}^{\text{init}}, w) \in F_{\mathcal{P}} \\
 &\Leftrightarrow \exists q_f \in \tilde{\delta}_{\mathcal{P}}(q_{\mathcal{P}}^{\text{init}}, w) \cap F \\
 &\Leftrightarrow \exists \underbrace{q_0, q_1, \dots, q_n}_{\text{Lauf}} \in Q, \text{ sodass } n = |w|, q_0 \in q^{\text{init}} \text{ und } q_n \in F \\
 &\Leftrightarrow \exists \text{ initialer, akzeptierender Lauf von } \mathcal{N} \text{ über } w \\
 &\Leftrightarrow w \in L(\mathcal{N}) \quad \square
 \end{aligned}$$

Bemerkung: Es gelten also die folgenden Äquivalenzen.

$$L \text{ regulär} \stackrel{\text{Def. 2.3}}{\iff} L = L(\mathcal{A}) \text{ für einen DEA } \mathcal{A} \iff L = L(\mathcal{N}) \text{ für einen NEA } \mathcal{N}$$

Bemerkung: NEAs sind eine exponentiell kompaktere Repräsentation von regulären Sprachen im folgenden Sinne:

1. Es gibt mit  $L_n$  ( $n$ -letztes Zeichen) eine Menge von Sprachen, die sich durch einen NEA mit  $n + 1$  Zuständen darstellen lassen, aber bei denen ein minimaler DEA mindestens  $2^n$  Zustände hat. (Siehe Übungsblatt 3, Aufgabe 2).
2. Zu jedem NEA mit  $n$  Zuständen gibt es einen DEA mit  $2^n$  Zuständen, der die gleiche Sprache akzeptiert. (Satz 2.7).
3. Zu jedem DEA mit  $n$  Zuständen gibt es einen NEA mit  $n$  Zuständen, der die gleiche Sprache akzeptiert.

### 2.4.1 $\varepsilon$ -Übergänge

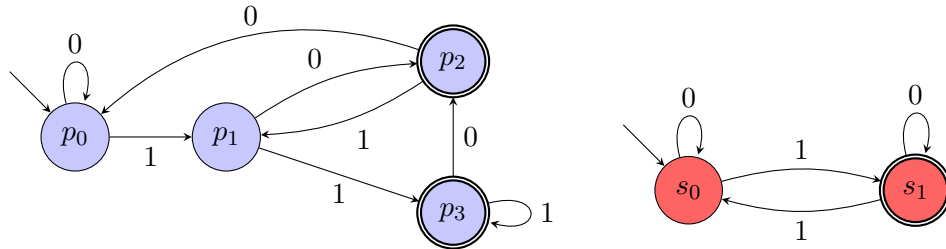
In diesem Unterkapitel führen wir mit dem  $\varepsilon$ -NEA ein weiteres Automatenmodell ein. Wir wollen  $\varepsilon$ -NEAs zunächst durch die folgende Fragestellung und anschließende Diskussion motivieren.

Frage: Gegeben zwei reguläre Sprachen  $L_1, L_2$ , ist auch die Konkatenation  $L_1 \cdot L_2$  eine reguläre Sprache?

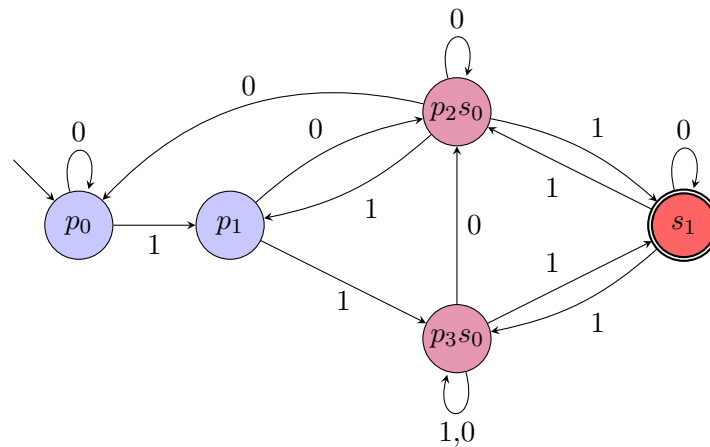
Idee: Gegeben DEA  $\mathcal{A}_1$  mit  $L(\mathcal{A}_1) = L_1$  und DEA  $\mathcal{A}_2$  mit  $L(\mathcal{A}_2) = L_2$ , konstruiere NEA für  $L_1 \cdot L_2$  durch „Hintereinanderschalten“ von  $\mathcal{A}_1$  und  $\mathcal{A}_2$ ; immer wenn wir in einem akzeptierenden Zustand von  $\mathcal{A}_1$  sind, erlauben wir, in  $\mathcal{A}_2$  zu „wechseln“.

Erste, naive (und inkorrekte) Umsetzung dieser Idee: Verschmelze akzeptierende Zustände von  $\mathcal{A}_1$  mit dem Startzustand von  $\mathcal{A}_2$ . Wir betrachten die folgenden Automaten, um zu sehen, dass diese Umsetzung nicht zielführend ist.

**Bsp. 2.16:** Links: DEA  $\mathcal{A}_1$ , der Automat aus Bsp. 2.14 eingeschränkt auf die erreichbaren Zustände. Rechts: DEA  $\mathcal{A}_2$  mit der Sprache  $\{w \in \{0,1\}^* \mid \text{Anzahl 1 in } w \text{ ungerade}\}$ .



Unten: NEA  $\mathcal{N}_{\text{naiv}}$  aus der naiven und inkorrekten Konstruktion für die Konkatination.



Dieser NEA akzeptiert nun auch das Wort  $w = 11011$ . Allerdings ist  $w$  nicht in der Konkatination  $L(\mathcal{A}_1) \cdot L(\mathcal{A}_2)$ , denn es gibt keine Zerlegung  $w = w_1 \cdot w_2$ , sodass sowohl das Präfix  $w_1$  von  $\mathcal{A}_1$  als auch das Suffix  $w_2$  von  $\mathcal{A}_2$  akzeptiert wird.

Das „Verschmelzen“ von  $p_2$  (bzw.  $p_3$ ) mit  $s_0$  war also keine gute Idee. Was uns aber helfen würde: ein Zustandsübergang, der es uns erlaubt, von Zustand  $p_2$  (bzw.  $p_3$ ) in den Zustand  $s_0$  zu gehen, ohne dabei ein Zeichen zu lesen.

Wir nennen solch einen Zustandsübergang  $\varepsilon$ -Übergang und definieren einen Automaten, der solche Zustandsübergänge haben kann, wie folgt.

**Def. 2.14 ( $\varepsilon$ -NEA):** Ein *nichtdeterministischer endlicher Automat mit  $\varepsilon$ -Übergängen* ist ein 5-Tupel

$$\mathcal{B} = (\Sigma, Q, \delta, q^{\text{init}}, F)$$

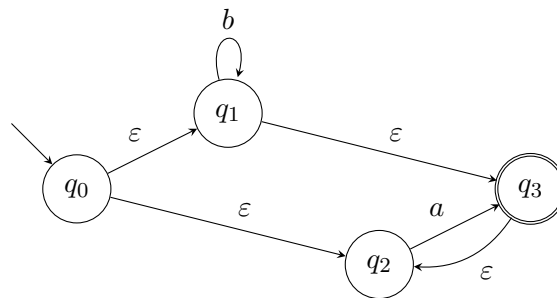
wobei  $\Sigma$ ,  $Q$ ,  $q^{\text{init}}$ ,  $F$  wie bei NEAs (bzw. DEAs) definiert sind und die Transitionsfunktion den folgenden Typ hat.

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q)$$

◇

**Bsp. 2.17:** Zustandsdiagramm eines  $\varepsilon$ -NEA über dem Alphabet  $\Sigma = \{a, b\}$ .

Vorlesung:  
10.11.17



Im Folgenden sei  $\mathcal{B}$  immer ein  $\varepsilon$ -NEA.

Wie bei den bisher definierten Automaten wollen wir mit Hilfe eines  $\varepsilon$ -NEA eine Sprache definieren. Wir benötigen dafür zunächst zwei weitere Definitionen.

Der  $\varepsilon$ -Abschluss ist eine Abbildung, die jedem Zustand  $q$  die Menge der Zustände zuordnet, die von  $q$  über  $\varepsilon$ -Übergänge erreichbar sind. Wir definieren diese Abbildung formal wie folgt. Dabei verwenden wir den Abbildungsnamen  $\text{ecl}$ , um an den englischen Begriff „ $\varepsilon$  closure“ zu erinnern.

**Def. 2.15:** Der  $\varepsilon$ -Abschluss  $\text{ecl}_{\mathcal{B}} : Q \rightarrow \mathcal{P}(Q)$  ist die kleinste Abbildung, die für alle  $q, q', q'' \in Q$  die folgenden Eigenschaften erfüllt:

$$\begin{aligned} q &\in \text{ecl}_{\mathcal{B}}(q) \\ q' \in \text{ecl}_{\mathcal{B}}(q) \text{ und } q'' \in \delta(q', \varepsilon) &\Rightarrow q'' \in \text{ecl}_{\mathcal{B}}(q) \end{aligned} \quad \diamond$$

Offensichtlich kann immer eine endliche explizite Repräsentation von  $\text{ecl}_{\mathcal{B}}$  berechnet werden: Starte in jedem Zustand einmal und folge mit Breitensuche allen  $\varepsilon$ -Kanten im Zustandsdiagramm.

**Bsp.:** Für den  $\varepsilon$ -NEA aus Bsp. 2.17 sieht  $\text{ecl}_{\mathcal{B}}$  wie folgt aus:

$q$	$q_0$	$q_1$	$q_2$	$q_3$
$\text{ecl}(q)$	$\{q_0, q_1, q_2, q_3\}$	$\{q_1, q_2, q_3\}$	$\{q_2\}$	$\{q_2, q_3\}$

Als Nächstes definieren wir eine dreistellige Relation, die uns für je zwei Zustände sagt, welche Wörter den Automaten vom ersten Zustand in den zweiten Zustand überführen. Der Name der Relation „reach“ soll dabei an das englische Wort „reachability“ erinnern.

**Def. 2.16:** Die *Erreichbarkeitsrelation*  $\text{reach}_{\mathcal{B}} \subseteq Q \times \Sigma^* \times Q$  ist die kleinste Relation, die für alle  $q, q', q'', q''' \in Q$  und für alle  $w \in \Sigma^*$  die folgenden Eigenschaften erfüllt:

$$\begin{aligned} q' \in \text{ecl}_{\mathcal{B}}(q) &\Rightarrow (q, \varepsilon, q') \in \text{reach}_{\mathcal{B}} \\ q' \in \text{ecl}_{\mathcal{B}}(q), q'' \in \delta(q', a) \text{ und } (q'', w, q''') \in \text{reach}_{\mathcal{B}} &\Rightarrow (q, aw, q''') \in \text{reach}_{\mathcal{B}} \quad \diamond \end{aligned}$$

Für den  $\varepsilon$ -NEA aus Bsp. 2.17 können wir  $\text{reach}_{\mathcal{B}}$  mit Hilfe der folgenden Tabelle angeben. Dabei bedeutet der Eintrag von einer Sprache  $L$  in Zeile  $q_i$  und Spalte  $q_j$ , dass für alle  $w \in L$  das Tripel  $(q_i, w, q_j)$  in  $\text{reach}_{\mathcal{B}}$  enthalten ist.

$\text{reach}_{\mathcal{B}}$	$q_0$	$q_1$	$q_2$	$q_3$
$q_0$	$\{\varepsilon\}$	$\{b\}^*$	$\{b\}^* \cdot \{a\}^*$	$\{b\}^* \cdot \{a\}^*$
$q_1$	$\{\}$	$\{b\}^*$	$\{b\}^* \cdot \{a\}^*$	$\{b\}^* \cdot \{a\}^*$
$q_2$	$\{\}$	$\{\}$	$\{a\}^*$	$\{a\} \cdot \{a\}^*$
$q_3$	$\{\}$	$\{\}$	$\{a\}^*$	$\{a\}^*$

**Def. 2.17:** Ein Wort  $w \in \Sigma^*$  wird von  $\mathcal{B}$  *akzeptiert*, wenn  $(q^{\text{init}}, w, q_f) \in \text{reach}_{\mathcal{B}}$  für ein  $q_f \in F$ . Die von  $\mathcal{B}$  akzeptierte Sprache  $L(\mathcal{B})$  ist die Menge der von  $\mathcal{B}$  akzeptierten Wörter, d.h.  $L(\mathcal{B}) = \{w \in \Sigma^* \mid \exists q \in F : (q^{\text{init}}, w, q) \in \text{reach}_{\mathcal{B}}\}$ .  $\diamond$

Offensichtlich gibt es zu jedem NEA  $\mathcal{N}$  einen  $\varepsilon$ -NEA  $\mathcal{B}$ , der die gleiche Sprache akzeptiert. Die Konstruktion ist dabei einfach: Erweitere die Transitionsfunktion um  $\delta(q, \varepsilon) = \{\}$  für alle  $q \in Q$ . Für die Sprachgleichheit zeigen wir via Induktion über die Länge von  $w$ , dass für alle Wörter  $w \in \Sigma^*$  gilt:

$$\exists \text{ Lauf } q_0, q_1, \dots, q_n \text{ von } \mathcal{N} \text{ über } w \Leftrightarrow (q_0, w, q_n) \in \text{reach}_{\mathcal{B}}$$

Der folgende Satz zeigt uns, dass auch die umgekehrte Richtung gilt.

**Satz 2.8:** Zu jedem  $\varepsilon$ -NEA  $\mathcal{B}$  gibt es einen NEA  $\mathcal{N}$ , sodass  $L(\mathcal{N}) = L(\mathcal{B})$  gilt.

Zur Vorbereitung des Beweises machen wir zunächst die folgende Definition.

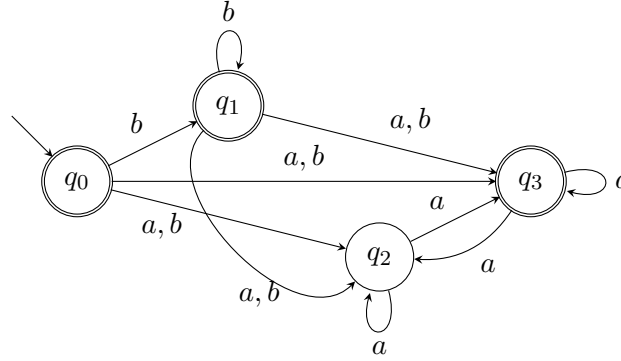
**Def. 2.18** ( $\varepsilon$ -freier Automat): Für einen gegebenen  $\varepsilon$ -NEA  $\mathcal{B} = (\Sigma, Q, \delta, q^{\text{init}}, F)$  definieren wir den NEA  $\mathcal{N} = (\Sigma, Q, \delta_{\mathcal{N}}, q^{\text{init}}, F_{\mathcal{N}})$  mit

$$\delta_{\mathcal{N}}(q, a) = \bigcup_{q' \in \text{ecl}_{\mathcal{B}}(q)} \{q''' \mid \exists q'' : q'' \in \delta(q', a) \text{ und } q''' \in \text{ecl}_{\mathcal{B}}(q'')\}$$

$$F_{\mathcal{N}} = \{q \in Q \mid \exists q_f \in F : q_f \in \text{ecl}_{\mathcal{B}}(q)\}$$

und nennen diesen NEA den  $\varepsilon$ -freien Automaten von  $\mathcal{B}$ .  $\diamond$

**Bsp. 2.18:** Der  $\varepsilon$ -freie Automat für den  $\varepsilon$ -NEA aus Bsp. 2.17 hat das folgende Zustandsdiagramm.



BEWEIS (von Satz 2.8:  $\varepsilon$ -Eliminierung): Zeige  $L(\mathcal{N}) = L(\mathcal{B})$ . Dabei verwenden wir die folgende Eigenschaft, die wir via Induktion über die Länge von  $w$  in den Übungen zeigen werden.

$\forall w \in \Sigma^+ \forall q, q' \in Q :$

$$(q, w, q') \in \text{reach}_{\mathcal{B}} \Leftrightarrow \exists \underbrace{q_0, q_1, \dots, q_n}_{\text{Lauf}} \in Q, \text{ sodass } n = |w|, q_0 = q \text{ und } q_n = q'$$

Mit Hilfe dieser Eigenschaft können wir nun leicht für  $w \neq \varepsilon$  zeigen:

$$\begin{aligned} w \in L(\mathcal{B}) &\stackrel{\text{Def. 2.3}}{\Leftrightarrow} \exists q_f \in F : (q^{\text{init}}, w, q_f) \in \text{reach}_{\mathcal{B}} \\ &\Leftrightarrow \exists q_f \in F_{\mathcal{N}} : \underbrace{q_0, q_1, \dots, q_n}_{\text{Lauf}} \in Q, \text{ sodass } n = |w|, q_0 = q^{\text{init}} \text{ und } q_n = q_f \\ &\stackrel{\text{Def. 2.17}}{\Leftrightarrow} w \in L(\mathcal{N}) \end{aligned}$$

Der Fall  $w = \varepsilon$  folgt mit  $\varepsilon \in L(\mathcal{B}) \Leftrightarrow \exists q \in F \cap \text{ecl}_{\mathcal{B}}(q^{\text{init}}) \Leftrightarrow q^{\text{init}} \in F_{\mathcal{N}} \Leftrightarrow \varepsilon \in L(\mathcal{N})$ .  $\square$

Mit Hilfe der  $\varepsilon$ -NEAs greifen wir nun die zu Beginn von Abschnitt 2.4.1 aufgeworfene Fragestellung wieder auf und zeigen, dass für je zwei reguläre Sprachen auch die Konkatenation regulär ist.

Wir geben hierfür zunächst eine Konstruktion an.

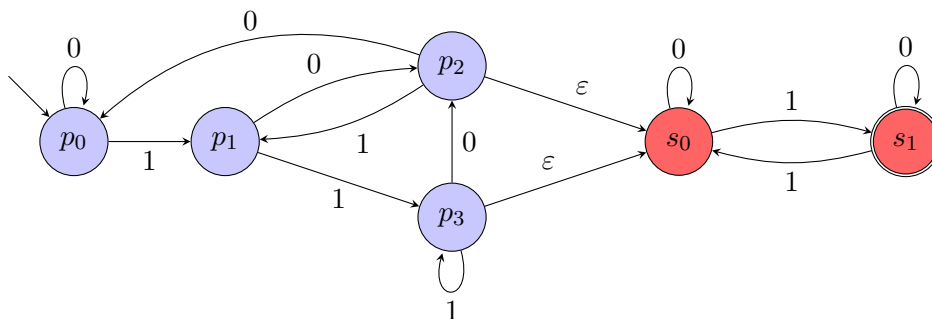
**Def. 2.19:** Gegeben zwei  $\varepsilon$ -NEA  $\mathcal{B}_i = (\Sigma, Q_i, \delta_i, q_i^{\text{init}}, F_i)$ ,  $i = 1, 2$ , definieren wir den

$\varepsilon$ -NEA für Konkatination  $\mathcal{B} = (\Sigma, Q, \delta, q^{\text{init}}, F)$  wie folgt.

$$\begin{aligned}
 Q &= Q_1 \cup Q_2 \\
 \delta(q, x) &= \begin{cases} \delta_1(q, x) & q \in Q_1 \wedge (q \notin F_1 \vee x \neq \varepsilon) \\ \delta_1(q, x) \cup \{q_2^{\text{init}}\} & q \in F_1 \wedge x = \varepsilon \\ \delta_2(q, x) & q \in Q_2 \end{cases} \\
 q^{\text{init}} &= q_1^{\text{init}} \\
 F &= F_2
 \end{aligned}$$

◇

**Bsp. 2.19:** Der  $\varepsilon$ -NEA für Konkatination für die beiden Automaten aus Bsp. 2.16 hat das folgende Zustandsdiagramm.



**Lemma 2.9:** Die vom  $\varepsilon$ -NEA für Konkatination akzeptierte Sprache ist  $L(\mathcal{B}_1) \cdot L(\mathcal{B}_2)$ .

BEWEIS: Zeige via Induktion über die Länge von  $w_1$ , dass  $\forall w_1, w_2 \in \Sigma^*, \forall q_1 \in Q_1, \forall q'_1 \in F, \forall q'_2 \in Q_2$  die folgende Eigenschaft gilt.

$$(q_1, w_1, q'_1) \in \text{reach}_{\mathcal{B}_1} \text{ und } (q^{\text{init}}, w_2, q'_2) \in \text{reach}_{\mathcal{B}_2} \Leftrightarrow (q_1, w, q'_2) \in \text{reach}_{\mathcal{B}}$$

Mit dieser Eigenschaft folgt leicht für alle  $w_1, w_2 \in \Sigma^*$ :

$$w_1 \in L(\mathcal{B}_1) \text{ und } w_2 \in L(\mathcal{B}_2) \Leftrightarrow w_1 \cdot w_2 \in L(\mathcal{B}) \quad \square$$

## 2.5 Abschlusseigenschaften

**Def. 2.20:** Eine Menge  $X$  heißt *abgeschlossen* unter Operation  $f : X^n \rightarrow X$ , falls  $\forall x_1, \dots, x_n \in X : f(x_1, \dots, x_n) \in X$ . ◇

Vorlesung:  
15.11.17

Zum Beispiel sind die natürlichen Zahlen abgeschlossen unter Addition, aber nicht abgeschlossen unter Subtraktion.

Im Folgenden schreiben wir *REG* für die Menge aller regulären Sprachen.

**Lemma 2.10:** Die Menge  $REG$  der regulären Sprachen ist abgeschlossen unter Komplement.

BEWEIS: Sei  $L$  eine reguläre Sprache. Dann gibt es (per Definition) einen DEA  $\mathcal{A} = (\Sigma, Q, \delta, q^{\text{init}}, F)$  der  $L$  akzeptiert. Wir konstruieren den DEA  $\overline{\mathcal{A}} = (\Sigma, Q, \delta, q^{\text{init}}, Q \setminus F)$  für  $\overline{L}$ , bei dem ein Zustand genau dann akzeptierend ist, wenn der Zustand in  $\mathcal{A}$  nicht akzeptierend ist. Man kann leicht zeigen, dass  $L(\overline{\mathcal{A}}) = \overline{L}$ ; somit ist auch  $\overline{L}$  regulär.  $\square$

**Lemma 2.11:** Die Menge  $REG$  der regulären Sprache ist abgeschlossen unter dem Stern-Operator.

BEWEIS: Sei  $L$  eine reguläre Sprache. Dann gibt es einen  $\varepsilon$ -NEA  $\mathcal{B} = (\Sigma, Q, \delta, q^{\text{init}}, F)$ , der  $L$  akzeptiert. Wir konstruieren einen NEA  $\mathcal{B}_* = (\Sigma, Q \cup \{q_*^{\text{init}}\}, \delta_*, q_*^{\text{init}}, F \cup \{q_*^{\text{init}}\})$  für  $L^*$ , indem wir

- einen neuen akzeptierenden Startzustand einführen,
- vom neuen Startzustand einen  $\varepsilon$ -Übergang zum alten Startzustand hinzufügen und
- von jedem akzeptierenden Zustand einen  $\varepsilon$ -Übergang zum alten Startzustand hinzufügen.

$$\delta_*(q, x) = \begin{cases} \delta(q, x) & q \notin F \text{ oder } x \neq \varepsilon \\ \delta(q, x) \cup \{q^{\text{init}}\} & q \in F \text{ und } x = \varepsilon \\ \{\} & q = q_*^{\text{init}} \text{ und } x \in \Sigma \\ \{q^{\text{init}}\} & q = q_*^{\text{init}} \text{ und } x = \varepsilon \end{cases}$$

Man kann via Induktion über die Länge von Wörtern zeigen, dass  $L(\mathcal{B}_*) = L^*$  gilt.  $\square$

Bemerkung: Die Menge der regulären Sprachen  $REG$  ist unter den folgenden Operationen abgeschlossen.

$\cap$	(Durchschnitt)	Satz 2.1
$\cup$	(Vereinigung)	Präsenzübungen erste Woche, Aufgabe 3
$\overline{\phantom{x}}$	(Komplement)	Lemma 2.10
$\cdot$	(Konkatenation)	Lemma 2.9
$*$	(Kleene-Stern)	Lemma 2.11

## 2.6 Reguläre Ausdrücke

### 2.6.1 Motivation

Problemstellung: Sie haben auf Ihrem Rechner irgendwo eine Textdatei mit Notizen zu dieser Vorlesung, können sich aber gerade nicht an den Pfad erinnern. Sie wissen aber



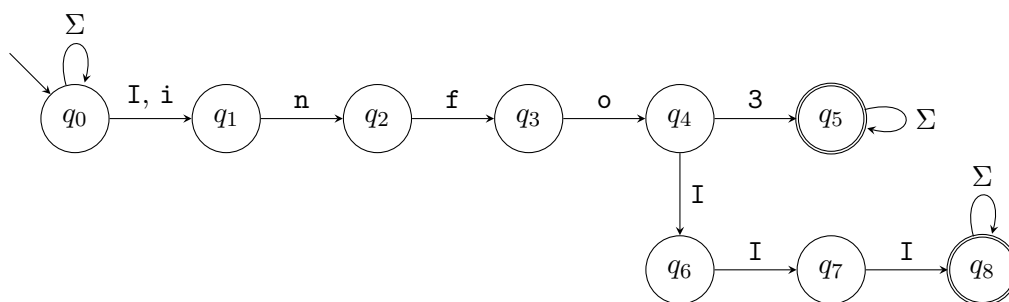
noch, dass die Zeichenkette `info3` oder `infoIII` enthalten ist. Möglicherweise war das `i` am Anfang aber auch groß geschrieben. Da Sie mehrere Dateien mit dieser Zeichenkette haben, soll der Rechner jeweils Dateinamen und die entsprechenden Zeilen ausgeben.

Auf Systemen, auf denen die GNU Tools installiert sind, kann dieses Problem mit dem folgenden Befehl gelöst werden.

```
grep -r "\(I|i\)nfo\(III\|3\)" /
```

Konzeptuell soll Ihr Rechner hier Instanzen des Wortproblems lösen. Ihre Sprache besteht aus allen Zeichenketten, die `info3` in den oben beschriebenen Varianten enthalten. Die Wörter, die getestet werden sollen, sind die Zeilen aller Dateien auf dem Rechner.

Eine Umsetzung: Gib dem Rechner den folgenden NEA, welchen er dann in einen DEA konvertiert, um das Wortproblem effizient zu lösen.



Problem: Dem Rechner einen Graphen als Eingabe zu übermitteln ist unkomfortabel oder zeitaufwändig. Wir führen deshalb einen weiteren textbasierten Formalismus für reguläre Sprachen ein. Dieser hat große Ähnlichkeiten zum dem in der Praxis verwendeten Input von `grep`.

**Def. 2.21:** Die Menge  $RE(\Sigma)$  der *regulären Ausdrücke über  $\Sigma$*  ist induktiv definiert durch:

- $\emptyset \in RE(\Sigma)$
- $\varepsilon \in RE(\Sigma)$
- $a \in RE(\Sigma)$  für alle  $a \in \Sigma$
- falls  $r, s \in RE(\Sigma)$ 
  - $(r + s) \in RE(\Sigma)$
  - $(r \cdot s) \in RE(\Sigma)$
  - $r^* \in RE(\Sigma)$

◇

Konventionen: Wir möchten nicht immer alle Klammern schreiben müssen. Wir führen deshalb die folgenden Präzedenzregeln ein: „ $*$ “ bindet stärker als „ $\cdot$ “ und „ $\cdot$ “ bindet stärker als „ $+$ “. Nach Definition der Semantik werden wir außerdem sehen, dass „ $\cdot$ “ und „ $+$ “ assoziativ sind. Unsere Konvention ist: Sofern mit Hilfe dieser Regeln der reguläre Ausdruck zweifelsfrei rekonstruiert werden kann, dürfen wir Klammern und „ $\cdot$ “ weglassen. Wir schreiben z.B.  $110 + 0$  statt  $((1 \cdot (1 \cdot 0)) + 0)$ .

**Bsp. 2.20:**

1.  $(A + \dots + Z + a + \dots + z + 0 + \dots + 9)^* (I + i) \text{ n f o } (3 + I I I) (A + \dots + Z + a + \dots + z + 0 + \dots + 9)^*$  ist ein regulärer Ausdruck<sup>12</sup> über dem Alphabet  $\Sigma = \{A, \dots, Z, a, \dots, z, 0, \dots, 9\}$ .
2.  $(\emptyset \varepsilon)^* + a$  ist ein regulärer Ausdruck über  $\Sigma = \{a\}$ .
3.  $\emptyset \varepsilon \emptyset \varepsilon$  ist ein regulärer Ausdruck über jedem Alphabet.
4.  $aaaabbbbbbb$  ist ein regulärer Ausdruck über  $\Sigma = \{a, b\}$ .
5.  $a^4 b^7$  ist **kein** regulärer Ausdruck
6.  $(0 + 1 \cdot (01^* 0)^* \cdot 1)^*$  ist ein regulärer Ausdruck über  $\Sigma = \{0, 1\}$ .

Wir geben regulären Ausdrücken mit Hilfe der folgenden Definition eine Semantik.

**Def. 2.22:** Die durch einen regulären Ausdruck *beschriebene Sprache*  $\llbracket \cdot \rrbracket : RE(\Sigma) \rightarrow \mathcal{P}(\Sigma^*)$  ist induktiv definiert durch:

$$\begin{aligned}
 \llbracket \emptyset \rrbracket &= \emptyset \\
 \llbracket \varepsilon \rrbracket &= \{\varepsilon\} \\
 \llbracket a \rrbracket &= \{a\} \quad a \in \Sigma \\
 \llbracket (r + s) \rrbracket &= \llbracket r \rrbracket \cup \llbracket s \rrbracket \\
 \llbracket (r \cdot s) \rrbracket &= \llbracket r \rrbracket \cdot \llbracket s \rrbracket \\
 \llbracket r^* \rrbracket &= \llbracket r \rrbracket^*
 \end{aligned}
 \quad \diamond$$

**Bsp. 2.21:**

$$\llbracket a + (\emptyset \varepsilon)^* \rrbracket = \llbracket a \rrbracket \cup (\llbracket \emptyset \rrbracket \cdot \llbracket \varepsilon \rrbracket)^* = \{a\} \cup (\emptyset)^* = \{a\} \cup \{\varepsilon\} = \{a, \varepsilon\}$$

**Bsp. 2.22:** Die aus Bsp. 2.14 bekannte Sprache

$$L_2 = \{w \in \{0, 1\}^* \mid \text{das zweitletzte Zeichen von } w \text{ ist } 1\}$$

wird durch den regulären Ausdruck  $(0 + 1)^* 1 (0 + 1)$  beschrieben.

<sup>12</sup>Wir verwenden hier  $A + \dots + Z$  als Abkürzung für die Disjunktion aus 26 Zeichen. Der eigentliche reguläre Ausdruck besteht aus der Disjunktion; die drei Punkte kommen darin nicht vor.

**Satz 2.12** (Kleene):  $L$  ist regulär  $\Leftrightarrow L$  ist Sprache eines regulären Ausdrucks.

BEWEIS (Kleene,  $\Leftarrow$ ): Betrachte zu einem regulärem Ausdruck  $r \in RE(\Sigma)$  die durch diesen erzeugte Sprache  $L = \llbracket r \rrbracket$ . Zeige via strukturelle Induktion über den Aufbau regulärer Ausdrücke, dass  $\llbracket r \rrbracket$  regulär ist.

- I.A.: •  $r = \emptyset$ ,  $\llbracket r \rrbracket = \emptyset$  ist regulär  
 •  $r = \varepsilon$ ,  $\llbracket r \rrbracket = \{\varepsilon\}$  ist regulär  
 •  $r = a$ ,  $\llbracket r \rrbracket = \{a\}$  ist regulär (NEA:  $\rightarrow \bigcirc \xrightarrow{a} \bigcirc \bigcirc$ )

I.V.: Für  $i \in \{1, 2\}$  gilt:  $\llbracket r_i \rrbracket$  ist regulär.

- I.S.: •  $r = r_1 + r_2$ ,  $\llbracket r \rrbracket = \llbracket r_1 \rrbracket \cup \llbracket r_2 \rrbracket$  ist regulär nach I.V. und Abschluss unter Vereinigung (gezeigt in Präsenzübungen, Aufgabe 3)  
 •  $r = r_1 \cdot r_2$ ,  $\llbracket r \rrbracket = \llbracket r_1 \rrbracket \cdot \llbracket r_2 \rrbracket$  ist regulär nach I.V. und Lemma 2.9  
 •  $r = r_1^*$ ,  $\llbracket r \rrbracket = \llbracket r_1 \rrbracket^*$  ist regulär nach I.V. und Lemma 2.11  $\square$

Im verbleibenden Teil des Abschnitts zu regulären Ausdrücken werden wir zeigen, dass wir für jeden DEA einen regulären Ausdruck mit der gleichen Sprache konstruieren können. (Richtung „ $\Rightarrow$ “ von Satz 2.12). Um unsere Idee zu beschreiben, benötigen wir zunächst die folgende Definition.

**Def. 2.23:** Sei  $\mathcal{A} = (Q, \Sigma, \delta, q^{\text{init}}, F)$  ein DEA. Für einen Zustand  $q \in Q$  ist die *Sprache des Zustands*  $L_q = \{w \in \Sigma^* \mid \tilde{\delta}(q, w) \in F\}$  die Sprache der Wörter, die von Zustand  $q$  aus in einen akzeptierenden Zustand führen.  $\diamond$

Im Folgenden betrachten wir einen beliebigen DEA und nummerieren dessen Zustände  $Q = \{q_0, q_1, \dots, q_n\}$ . Wir leiten nun ein Gleichungssystem zwischen den Sprachen  $L_{q_i}$  her.

$$L_{q_i} = \{w \in \Sigma^* \mid \tilde{\delta}(q_i, w) \in F\}$$

Zunächst teilen wir  $L_{q_i}$  in den Teil, der (potentiell) das leere Wort enthält, und den Teil, der die nicht-leeren Wörter enthält.

$$= \{\varepsilon \mid q_i \in F\} \cup \bigcup_{a \in \Sigma} \{a\} \{w' \in \Sigma^* \mid \tilde{\delta}(\delta(q_i, a), w') \in F\}$$

Die nicht-leeren Wörter hängen von den Sprachen der Folgezustände ab

$$= \{\varepsilon \mid q_i \in F\} \cup \bigcup_{a \in \Sigma} \{a\} L_{\delta(q_i, a)}$$

Anstatt die Vereinigung über die Transitionen  $a$  und Folgezustandssprachen  $L_{q_j}$  zu bilden, lassen sich die nicht-leeren Wörter von  $L_{q_i}$  auch als Vereinigung über alle Zustände mit entsprechend gewählten *Koeffizienten*  $A_{ij}$  formulieren; die Zustände, die keine Folgezustände sind, haben den Koeffizienten  $\emptyset$ .

$$= \{\varepsilon \mid q_i \in F\} \cup \bigcup_{j=0}^n \underbrace{\{a \in \Sigma \mid \delta(q_i, a) = q_j\}}_{A_{ij} \neq \varepsilon} L_{q_j}$$

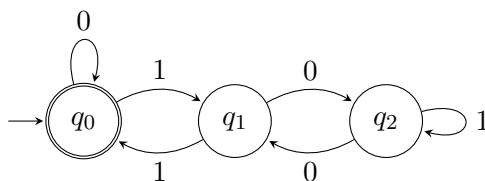
Diese Gleichungen lassen sich analog als Gleichungen von regulären Ausdrücken  $r_i$  formulieren:

$$r_i = N(q_i) + \sum_{j=0}^n R_{ij} r_j$$

wobei  $\llbracket r_i \rrbracket = L_{q_i}$  und  $R_{ij} = \sum \{a \in \Sigma \mid \delta(q_i, a) = q_j\}$  mit  $\varepsilon \notin \llbracket R_{ij} \rrbracket$  und

$$N(q_i) = \begin{cases} \varepsilon & q_i \in F \\ \emptyset & q_i \notin F \end{cases}$$

**Bsp. 2.23:** Wir wollen diese Gleichungen zunächst an einem Beispiel betrachten und verwenden dafür den aus Bsp. 2.12 bekannten DEA, der die Sprache der Binärcodierung von durch drei teilbaren Zahlen akzeptiert:  $L = \{w \in \{0, 1\}^* \mid \text{bin}(w) \equiv_3 0\}$ .



**Abb. 4:** DEA „modulo 3“

Wir erhalten das folgende Gleichungssystem mit drei Unbekannten.

$$r_0 = \varepsilon + 0 \cdot r_0 + 1 \cdot r_1$$

$$r_1 = 1 \cdot r_0 + 0 \cdot r_2$$

$$r_2 = 0 \cdot r_1 + 1 \cdot r_2$$

Wir kennen bisher kein systematisches Verfahren zum Lösen solcher Gleichungen und betrachten deshalb das folgende Lemma.

**Lemma 2.13** (Ardens Lemma):

Für die Gleichung  $X = A \cdot X \cup B$  über den Sprachen  $A, B, X \subseteq \Sigma^*$  gilt:

1. Die Sprache  $A^*B$  ist eine Lösung für  $X$ .
2. Falls  $\varepsilon \notin A$ , so ist diese Lösung eindeutig.

BEWEIS:

1. Wir können leicht nachrechnen, dass  $A^*B$  tatsächlich eine Lösung ist.

$$\underbrace{A^*B}_X = (AA^* \cup \{\varepsilon\})B = A \cdot \underbrace{(A^*B)}_X \cup B$$

2. Sei  $\varepsilon \notin A$ . Aus dem Beweis für 1. wissen wir außerdem, dass jede Lösung eine Obermenge von  $A^*B$  sein muss. Wir führen einen Widerspruchsbeweis, dass keine Lösung ein Wort  $x \in \Sigma^*$  mit  $x \notin A^*B$  enthalten kann.

Angenommen es gäbe Wörter  $x$  mit dieser Eigenschaft. Dann gibt es auch Wörter minimaler Länge mit dieser Eigenschaft. Sei  $w$  solch ein Wort minimaler Länge. Da alle Lösungen von der Form  $\underbrace{A^n X}_{\ni w} \cup \underbrace{A^{n-1}B \cup \dots \cup AB \cup B}_{\not\ni w}$  sind, hat  $w$  die Form

$w = u_1 \dots u_n w'$  mit  $u_1, \dots, u_n \in A$  und  $w' \in X$ . Fallunterscheidung:

- $w' \in A^*B \Rightarrow w \in A^n A^*B \subseteq A^*B$  – Widerspruch!
- $w' \notin A^*B \Rightarrow w'$  ist bereits Element einer Lösung. Widerspruch zur minimalen Länge von  $w$ .

Die Sprache  $A^*B$  ist also die einzige Lösung für  $X$ . □

Wir können Ardens Lemma wie folgt auch für reguläre Ausdrücke formulieren:

**Korollar 2.14:** Seien  $r_X, r_A, r_B$  reguläre Ausdrücke mit  $\varepsilon \notin \llbracket r_A \rrbracket$ , sodass die folgende Gleichung gilt:

$$\llbracket r_X \rrbracket = \llbracket r_A \cdot r_X + r_B \rrbracket$$

Dann ist der reguläre Ausdruck

$$r_A^* r_B$$

eine Lösung für  $r_X$ , welche die Gleichung erfüllt. Außerdem erzeugen alle anderen Lösungen die gleiche Sprache wie  $r_A^* r_B$ . ◇

Wir haben nun alle Hilfsmittel, um den Beweis für die fehlende Richtung zu führen.

BEWEIS (Kleene,  $\Rightarrow$ ): Sei  $\mathcal{A} = (\Sigma, Q, \delta, q^{\text{init}}, F)$  der oben diskutierte DEA mit  $Q = \{q_0, q_1, \dots, q_n\}$  und  $q^{\text{init}} = q_0$ , für dessen Sprache wir einen regulären Ausdruck konstruieren möchten.

Mit Hilfe von Ardens Lemma (und weiteren Rechenregeln für Sprachen) können wir nun das Gleichungssystem (TODO refer to) iterativ lösen. Wir beginnen mit Gleichung  $r_n$ :

$$\begin{aligned} r_n &= N(q_n) + \sum_{j=0}^n R_{nj} r_j \\ &= N(q_n) + \underbrace{\left( \sum_{j=0}^{n-1} R_{nj} r_j \right)}_{r_B} + \underbrace{R_{nn} r_n}_{r_A} \end{aligned}$$

Wie oben angedeutet ist nach dem Herausziehen des  $n$ -ten Summenglieds Ardens Lemma anwendbar (beachte:  $\varepsilon \notin \llbracket R_{nn} \rrbracket$ ); wir erhalten:

$$r_n := R_{nn}^* \left( N(q_n) + \sum_{j=0}^{n-1} R_{nj} r_j \right)$$

Dieses Ergebnis in  $r_0, \dots, r_{n-1}$  eingesetzt ergibt:

$$r_i = N(q_i) + \left( \sum_{j=0}^{n-1} R_{ij} r_j \right) + R_{in} \underbrace{R_{nn}^* \left( N(q_n) + \sum_{j=0}^{n-1} R_{nj} r_j \right)}_{r_n}$$

(Ausmultiplizieren von  $R_{in} R_{nn}^*$ )

$$= N(q_i) + \left( \sum_{j=0}^{n-1} R_{ij} r_j \right) + R_{in} R_{nn}^* N(q_n) + \sum_{j=0}^{n-1} R_{in} R_{nn}^* R_{nj} r_j$$

(Zusammenlegen der Summen und Ausklammern von  $r_j$ )

$$= N(q_i) + R_{in} R_{nn}^* N(q_n) + \sum_{j=0}^{n-1} (R_{ij} + R_{in} R_{nn}^* R_{nj}) r_j$$

Nach diesen Umformungen ergeben sich  $\varepsilon$ -freie Koeffizienten  $R_{nj} + R_{in} R_{nn}^* R_{nj}$  für  $r_j$  und wir können mit dem Auflösen der Summe von  $n-1$  analog zu  $n$  fortfahren. Am Ende erhalten wir einen regulären Ausdruck als Lösung für  $r_0$ . Per Konstruktion gilt  $\llbracket r_0 \rrbracket = L_{q_0} = L_{q^{\text{init}}} = L(\mathcal{A})$ .  $\square$

**Bsp. 2.24:** Wir betrachten noch einmal Bsp. 2.23 und lösen das Gleichungssystem auf die im Beweis beschriebene Weise.

$$\begin{aligned} r_0 &= \varepsilon + 0 \cdot r_0 + 1 \cdot r_1 \\ r_1 &= 1 \cdot r_0 + 0 \cdot r_2 \\ r_2 &= \underbrace{0 \cdot r_1}_B + \underbrace{1}_{A} \cdot r_2 \end{aligned}$$

Ardens Lemma auf  $r_2$  anwenden:

$$r_2 = 1^* \cdot 0 \cdot r_1$$

Einsetzen in  $r_1$ :

$$r_1 = \underbrace{1 \cdot r_0}_B + \underbrace{0 \cdot 1^* \cdot 0 \cdot r_1}_A$$

Ardens Lemma auf  $r_1$  anwenden:

$$r_1 = (01^*0)^* \cdot 1 \cdot r_0$$

Einsetzen in  $r_0$ :

$$\begin{aligned} r_0 &= \underline{\varepsilon} + 0 \cdot r_0 + 1 \cdot (01^*0)^* \cdot 1 \cdot r_0 \\ &= \underbrace{\underline{\varepsilon}}_B + \underbrace{(0 + 1 \cdot (01^*0)^* \cdot 1)}_A \cdot r_0 \end{aligned}$$

Ardens Lemma auf  $r_0$  anwenden:

$$\begin{aligned} r_0 &= (0 + 1 \cdot (01^*0)^* \cdot 1)^* \cdot \underline{\varepsilon} \\ &= (0 + 1(01^*0)^*1)^* \end{aligned}$$

### 3 Grammatiken und kontextfreie Sprachen

**Def. 3.1:** Eine *Grammatik* ist ein 4-Tupel  $(\Sigma, N, P, S)$  mit folgenden Komponenten:

- $\Sigma$  ist ein Alphabet, dessen Elemente wir in diesem Kontext auch *Terminalsymbole* nennen.
- $N$  ist eine endliche Menge, deren Elemente wir *Nichtterminalsymbole* oder *Variablen* nennen.
- $P \subseteq (N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$  ist eine endliche Relation, deren Elemente wir *Regeln* oder *Produktionen* nennen.
- $S \in N$  ist ein Nichtterminalsymbol, das wir *Startsymbol* nennen.  $\diamond$

**Bsp. 3.1:**  $\mathcal{G} = (\Sigma, N, P, S)$  mit<sup>13</sup>

$$\begin{aligned}\Sigma &= \{0, 1\} \\ N &= \{S\} \\ P &= \{S \rightarrow 1S0S \\ &\quad S \rightarrow 0S1S \\ &\quad S \rightarrow \varepsilon\}\end{aligned}$$

**Def. 3.2** (Ableitungsrelation, Ableitung, Sprache einer Grammatik): Sei  $\mathcal{G} = (\Sigma, N, P, S)$  eine Grammatik. Die *Ableitungsrelation*<sup>14</sup> zu  $\mathcal{G}$  ist

$$\cdot \vdash_{\mathcal{G}} \cdot \subseteq (N \cup \Sigma)^* \times (N \cup \Sigma)^*$$

mit  $\alpha \vdash_{\mathcal{G}} \beta$  gdw  $\alpha = \gamma_1 \alpha' \gamma_2$ ,  $\beta = \gamma_1 \beta' \gamma_2$  und  $\alpha' \rightarrow \beta' \in P$

Eine Folge  $\alpha = \alpha_0, \dots, \alpha_n = \beta$  heißt *Ableitung von  $\beta$  aus  $\alpha$  in  $n$  Schritten*, geschrieben  $\alpha \xrightarrow{n}_{\mathcal{G}} \beta$ , gdw  $\alpha_i \vdash_{\mathcal{G}} \alpha_{i+1}$  für  $0 \leq i < n$ . Jedes solche  $\alpha_i$  heißt *Satzform von  $\mathcal{G}$* .

Die *Ableitung von  $\beta$  aus  $\alpha$* , geschrieben  $\alpha \xrightarrow{*}_{\mathcal{G}} \beta$ , existiert gdw ein  $n \in \mathbb{N}$  existiert, sodass  $\alpha \xrightarrow{n}_{\mathcal{G}} \beta$ . Damit ist „ $\xrightarrow{*}_{\mathcal{G}}$ “ die reflexive, transitive Hülle von „ $\vdash_{\mathcal{G}}$ “.

Ein Wort  $w \in \Sigma^*$  wird von  $\mathcal{G}$  *erzeugt*, wenn  $S \xrightarrow{*}_{\mathcal{G}} w$  gilt. Die von  $\mathcal{G}$  *erzeugte Sprache* ist definiert als:

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

Wir nennen zwei Grammatiken *äquivalent*, wenn sie die gleiche Sprache erzeugen.  $\diamond$

<sup>13</sup> $P$  ist eine „normale“ binäre Relation, doch wir verwenden statt „ $(x, y) \in P$ “ meist „ $x \rightarrow y$ “, also einen Pfeil und Infix-Notation, um die Lesbarkeit zu erhöhen.

<sup>14</sup>Analog zur Relation  $P$  verwenden wir auch für die Ableitungsrelation wieder Infix-Notation, also  $\alpha \vdash \beta$  statt  $(\alpha, \beta) \in \vdash$ .



**Bsp. 3.2:** Wir betrachten nochmal die Grammatik aus Bsp. 3.1. Es gilt:  $1001 \in L(\mathcal{G})$ .

$$S \vdash_{\mathcal{G}} 1S0S \vdash_{\mathcal{G}} 10S \vdash_{\mathcal{G}} 100S1S \vdash_{\mathcal{G}} 100S1 \vdash_{\mathcal{G}} 1001$$

Außerdem gilt:  $L(\mathcal{G})$  ist die Sprache der Wörter über  $\{0, 1\}$ , die gleich viele Nullen wie Einsen haben:

$$L = \{w \in \Sigma^* \mid \#_0(w) = \#_1(w)\}$$

Vorlesung:  
22.11.17

Die Funktion  $\#_a(w)$  berechnet hierbei die Anzahl der Vorkommen von  $a \in \{0, 1\}$  in  $w$ .

Dass  $L(\mathcal{G}) \subseteq L$ , lässt sich via Induktion über die Länge der Ableitung von  $S \vdash_{\mathcal{G}}^* w$  zeigen. Der Beweis wird als Übung dem Leser überlassen.

Wir zeigen  $L \subseteq L(\mathcal{G})$ . Dazu zeigen wir „Wenn  $w \in L$ , dann  $w \in L(\mathcal{G})$ “ via Induktion über die Länge von  $w$ . Hierzu definieren wir noch die Hilfsfunktion  $d : \Sigma^* \rightarrow \mathbb{N}$ :

$$\begin{aligned} d(\varepsilon) &= 0 \\ d(1w) &= d(w) + 1 \\ d(0w) &= d(w) - 1 \end{aligned}$$

Per Induktion über  $|w|$  mit  $w \in \Sigma^*$  lässt sich leicht zeigen, dass  $L = \{w \in \Sigma^* \mid d(w) = 0\}$  und  $d(v \cdot w) = d(v) + d(w)$ .

Wir zeigen nun via Induktion über  $n$  die folgende Eigenschaft:

$$\forall n' < n : \forall w \in \Sigma^* : \text{falls } |w| = n' \text{ und } w \in L, \text{ dann } w \in L(\mathcal{G})$$

I.A.:  $n = 0$ :  $w = \varepsilon$ . Es gilt  $\varepsilon \in L$ , da  $\#_0(\varepsilon) = \#_1(\varepsilon) = 0$ .

I.S.:  $n \rightsquigarrow n + 1$ :  $|w| = n > 1$ ,  $w = aw'$ ,  $a \in \{0, 1\}$ . Beachte:  $|w|$  ist gerade für alle  $w \in L$ .

Betrachte  $a = 0$  (der Fall für  $a = 1$  funktioniert analog).

Da  $0 = d(w) = d(0w') = d(w') - 1$ , ist  $d(w') = 1$ .

Wir zeigen zunächst, dass wir  $w'$  in  $w_1 1 w_2$  mit  $d(w_1) = 0$  und  $d(w_2) = 0$  zerlegen können:

Sei  $w' = a_1 \dots a_n$ . Betrachte die Folge  $d_0, \dots, d_n$  mit  $d_0 = 0$  und  $d_i = d(a_1 \dots a_i)$  für  $1 \leq i \leq n$ . Wähle  $0 \leq i < n$  maximal, sodass für alle  $0 \leq j \leq i$  gilt:  $d_j < 1$ .<sup>15</sup>

Da  $i$  maximal ist, folgt  $d_{i+1} \geq 1$ . Da  $d_{j+1} - d_j \leq 1$  für alle  $0 \leq j \leq i$ , folgt  $d_{i+1} - d_i \leq 1$  und damit auch  $d_i = 0$ ,  $d_{i+1} = 1$  und  $a_{i+1} = 1$ . Setze  $w_1 = a_1 \dots a_i$ .

Es gilt also  $w' = a_1 \dots a_i a_{i+1} w_2 = w_1 w_2$  mit  $d(w_1) = 0$ .

Da  $d(v \cdot w) = d(v) + d(w)$  und  $d(w') = d(w_1 1 w_2) = 1$ , folgt  $d(w_2) = 0$ .

<sup>15</sup>Wir wissen, dass  $i < n$ , da  $d_n = d(w') = 1$ .

Da  $|w_1| < n$  und  $|w_2| < n$ , folgt nach I.V., dass  $S \vdash_{\mathcal{G}}^* w_1$  und  $S \vdash_{\mathcal{G}}^* w_2$ .

Es folgt mit den Produktionsregeln  $S \vdash_{\mathcal{G}} 0S1S \vdash_{\mathcal{G}}^* 0w_11S \vdash_{\mathcal{G}}^* 0w_11w_2$ .

**Bsp. 3.3:**  $\mathcal{G} = (\Sigma, N, P, S)$  mit

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

$$N = \{S, B, C\}$$

$$P = \{S \rightarrow aSBC, S \rightarrow aBC, CB \rightarrow BC, aB \rightarrow ab, bB \rightarrow bb, bC \rightarrow bc, cC \rightarrow cc\}$$

Es gilt z.B.  $aaabbbccc \in L(\mathcal{G})$ .

Außerdem gilt  $L(\mathcal{G}) = \{a^n b^n c^n \mid n \geq 1\}$ . (Ohne Beweis)

Die Chomsky-Hierarchie teilt die Grammatiken in vier Typen unterschiedlicher Mächtigkeit ein.

**Def. 3.3** (Chomsky-Hierarchie):

- Jede Grammatik ist eine *Typ-0-Grammatik*.
- Eine Grammatik ist *Typ-1* oder *kontextsensitiv*, falls alle Regeln expansiv sind, d.h., für alle Regeln  $\alpha \rightarrow \beta \in P$  ist  $|\alpha| \leq |\beta|$ . Ausnahme: falls  $S$  nicht in einer rechten Regelseite auftritt, dann ist  $S \rightarrow \varepsilon$  erlaubt.
- Eine Grammatik heißt *Typ-2* oder *kontextfrei*, falls alle Regeln die Form  $A \rightarrow \alpha$  mit  $A \in N$  und  $\alpha \in (N \cup \Sigma)^*$  haben.
- Eine Grammatik heißt *Typ-3* oder *regulär*, falls alle Regeln die folgende Form haben:

$$\begin{aligned} A &\rightarrow w & w &\in \Sigma^* \\ \text{oder } A &\rightarrow aB & a &\in \Sigma, B \in N \end{aligned}$$

Eine Sprache heißt *Typ- $i$ -Sprache*, falls es eine *Typ- $i$ -Grammatik* für sie gibt.  $\diamond$

**Beobachtung 3.1:** Jede *Typ- $(i+1)$ -Sprache* ist auch eine *Typ- $i$ -Sprache*.

- Jede *Typ-3-Grammatik* ist eine *Typ-2-Grammatik*.
- Wir werden im folgenden Unterkapitel zeigen, dass jede *Typ-2-Grammatik* in eine äquivalente  $\varepsilon$ -freie<sup>16</sup> *Typ-2-Grammatik* transformiert werden kann. Diese erfüllt dann auch alle Bedingungen einer *Typ-1-Grammatik*.
- Jede *Typ-1-Grammatik* ist auch eine *Typ-0-Grammatik*.

Im Lauf der Vorlesung werden wir die folgende Aussage zeigen:

Sei  $\mathcal{M}_i$  die Menge der *Typ- $i$  Sprachen*; dann gilt:  $\mathcal{M}_3 \subsetneq \mathcal{M}_2 \subsetneq \mathcal{M}_1 \subsetneq \mathcal{M}_0$ .

<sup>16</sup>Auf keiner rechten Seite steht  $\varepsilon$ ; Ausnahme: Startsymbol  $S$ ; vgl. Elimination von  $\varepsilon$ -Produktionen.

### 3.1 Kontextfreie Sprachen

Die Sprache aus Bsp. 3.1 ist kontextfrei. Weitere Beispiele sind:

**Bsp. 3.4:** Arithmetische Ausdrücke ohne Klammern:  $\mathcal{G} = (\{E\}, \{a, +, *\}, P, E)$  mit

$$P = \{E \rightarrow a \mid E + E \mid E * E\}.$$
<sup>17</sup>

**Bsp. 3.5:** Arithmetische Ausdrücke mit Klammern:  $\mathcal{G} = (\{E\}, \{a, +, *, (, )\}, P, E)$  mit

$$P = \{E \rightarrow a \mid (E + E) \mid (E * E)\}.$$

**Bsp. 3.6:** Syntax von Programmiersprachen. Wir zeigen hier nur exemplarisch Produktionen für einige typische Bestandteile einer Programmiersprache.<sup>18</sup> Wörter in spitzen Klammern sind hier Nichtterminalsymbole. Terminalsymbole sind grau unterlegt.

$$\begin{aligned} \langle \text{Stmt} \rangle &\rightarrow \langle \text{Var} \rangle \text{ = } \langle \text{Exp} \rangle \\ &\mid \langle \text{Stmt} \rangle \text{ ; } \langle \text{Stmt} \rangle \\ &\mid \text{ if } ( \langle \text{Exp} \rangle ) \langle \text{Stmt} \rangle \text{ else } \langle \text{Stmt} \rangle \\ &\mid \text{ while } ( \langle \text{Exp} \rangle ) \langle \text{Stmt} \rangle \end{aligned}$$

Für den arithmetischen Ausdruck ohne Klammern  $a + a$  gibt es in der Grammatik aus Bsp. 3.4 zwei verschiedene Ableitungen:

- $E \vdash_{\mathcal{G}} E + E \vdash_{\mathcal{G}} a + E \vdash_{\mathcal{G}} a + a$
- $E \vdash_{\mathcal{G}} E + E \vdash_{\mathcal{G}} E + a \vdash_{\mathcal{G}} a + a$

Beide Ableitungen unterscheiden sich nur durch die Reihenfolge, in der Variablen ersetzt werden. Die folgende Definition erlaubt es uns, beide Ableitungen durch das gleiche Objekt, den sogenannten Ableitungsbaum, darzustellen.

**Def. 3.4:** Sei  $\mathcal{G} = (\Sigma, N, P, S)$  eine kontextfreie Grammatik. Wir definieren die Menge der *in A beginnenden Ableitungsbäume von  $\mathcal{G}$* ,  $\text{Abl}(A)$ , für  $A \in N$  als Menge von beschrifteten, geordneten Bäumen induktiv durch:

Falls  $\pi = A \rightarrow w_0 A_1 w_1 \dots A_n w_n \in P$  mit  $A_i \in N$ ,  $w_0, w_i \in \Sigma^*$ ,  $1 \leq i \leq n$  und  $\mathcal{T}_i \in \text{Abl}(A_i)$ , dann ist

<sup>17</sup>Notation: Wenn wir mehrere Regeln mit gleicher linker Seite wie z.B. „ $A \rightarrow \alpha$ “ und „ $A \rightarrow \beta$ “ haben, dann dürfen wir diese auch mit Hilfe eines senkrechten Striches als „ $A \rightarrow \alpha \mid \beta$ “ schreiben.

<sup>18</sup> Sie finden am Ende der Java Language Specification <https://docs.oracle.com/javase/specs/> auf ca. 25 Seiten die kontextfreie Grammatik für diese Programmiersprache.

$$\begin{array}{c} \pi \\ \swarrow \quad \searrow \\ \mathcal{T}_1 \quad \dots \quad \mathcal{T}_n \end{array} \in \text{Abl}(A)$$

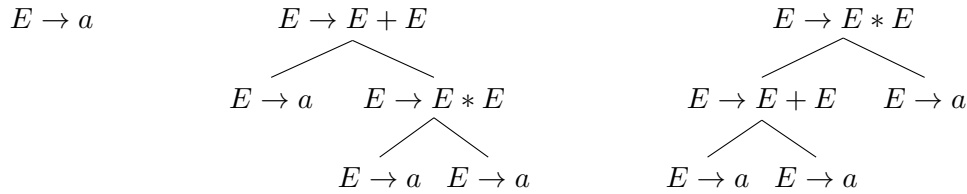
Da es manchmal aufwendig ist, Bäume zu zeichnen, verwenden wir alternativ die folgende Notation:  $\pi(\mathcal{T}_1, \dots, \mathcal{T}_n)$

Das *abgeleitete Wort* zu einem  $\mathcal{T} \in \text{Abl}(A)$ ,  $Y(\mathcal{T})$ <sup>19</sup>, ist definiert durch

$$Y \left( \begin{array}{c} \pi \\ \swarrow \quad \searrow \\ \mathcal{T}_1 \quad \dots \quad \mathcal{T}_n \end{array} \right) = w_0 Y(\mathcal{T}_1) w_1 \dots Y(\mathcal{T}_n) w_n$$

wobei  $\pi = A \rightarrow w_0 A_1 w_1 \dots A_n w_n \in P$ . ◇

**Bsp. 3.7:** Für die Grammatik aus Bsp. 3.4 sind die folgenden drei Beispiele in  $E$  beginnende Ableitungsbäume.



**Lemma 3.1:** Sei  $\mathcal{G} = (\Sigma, N, P, S)$  eine kontextfreie Grammatik.

$$w \in L(\mathcal{G}) \quad \text{gdw} \quad \exists \mathcal{T} \in \text{Abl}(S) \text{ mit } Y(\mathcal{T}) = w$$

(Ohne Beweis.)

**Def. 3.5:**

- Eine Grammatik  $\mathcal{G}$  heißt *eindeutig*, falls es für jedes Wort  $w \in L(\mathcal{G})$  genau einen Ableitungsbaum gibt.
- Eine kontextfreie Sprache  $L$  heißt *eindeutig*, falls es eine eindeutige kontextfreie Grammatik gibt, die  $L$  erzeugt. ◇

Die Grammatik aus Bsp. 3.4 ist also nicht eindeutig, denn sowohl für den zweiten als auch für den dritten Ableitungsbaum aus Bsp. 3.7 ist das abgeleitete Wort  $a + a * a$ .

Im Folgenden verwenden wir auch die Abkürzung CFG für „kontextfreie Grammatik“ (engl. context-free grammar).

<sup>19</sup>Wir verwenden  $Y(\mathcal{T})$ , um an das englische Wort „yield“ zu erinnern.

### 3.2 Die Chomsky-Normalform für kontextfreie Sprachen

Vorlesung:  
24.11.2017

Wir lernen in diesem Unterkapitel eine spezielle Form von kontextfreien Grammatiken (Chomsky-Normalform) kennen, für die gilt:

- Das Wortproblem ( $w \in L(\mathcal{G})?$ ) lässt sich mit Hilfe eines einfachen Algorithmus entscheiden.
- Jede kontextfreie Grammatik lässt sich „effizient“ in diese Normalform transformieren.

Wir stellen im Folgenden vier Transformationen (SEP, BIN, DEL, UNIT) vor und wollen dabei jeweils den Zeitaufwand und die Größe der resultierenden CFG analysieren.

Dafür definieren wir die Größe einer Grammatik wie folgt.

$$|\mathcal{G}| = \sum_{A \in N} \sum_{A \rightarrow \alpha \in P} |A\alpha|$$

Die Größe ist also genau die Anzahl an Zeichen aus  $\Sigma \cup N$ , die man benötigt, um die Regeln der Grammatik aufzuschreiben.

**Def. 3.6:** Eine CFG heißt *separiert*, wenn jede Regel eine der folgenden beiden Formen hat.

$$\begin{aligned} A &\rightarrow A_1 \dots A_n && \text{für } A \in N, A_i \in N, n \geq 0 \\ A &\rightarrow a && \text{für } A \in N, a \in \Sigma \end{aligned} \quad \diamond$$

**Lemma 3.2 (SEP):** Zu jeder CFG gibt es eine äquivalente separierte CFG.

BEWEIS: Sei  $\mathcal{G} = (\Sigma, N, P, S)$  eine CFG. Konstruiere  $\mathcal{G}' = (\Sigma, N', P', S)$  mit

- $N' = N \dot{\cup} \{Y_a \mid a \in \Sigma\}$ <sup>20</sup>
- $P' = \{Y_a \rightarrow a \mid a \in \Sigma\} \cup \{A \rightarrow \beta[a \rightarrow Y_a] \mid A \rightarrow \beta \in P\}$ .

Die Schreibweise  $\beta[a \rightarrow Y_a]$  bedeutet hier, dass alle  $a \in \Sigma$ , die in  $\beta$  vorkommen, durch  $Y_a$  ersetzt werden.

Offenbar gilt  $L(\mathcal{G}) = L(\mathcal{G}')$  und  $\mathcal{G}'$  ist separiert (ohne Beweis). □

**Bemerkung:** SEP berechnet in  $O(|\mathcal{G}|^2)$  eine Grammatik der Größe  $O(|\mathcal{G}|)$ .

- Laufe für jedes Terminalsymbol einmal über alle Regeln.
- Für jedes neue Nichtterminalsymbol fügen wir eine Regel der Größe 2 hinzu.

<sup>20</sup> „ $\dot{\cup}$ “ steht für „disjunkte Vereinigung“

**Lemma 3.3** (BIN): Zu jeder CFG gibt es eine äquivalente CFG, bei der für alle Regeln  $A \rightarrow \alpha$  gilt, dass  $|\alpha| \leq 2$ .

BEWEIS: Ersetze jede Regel der Form

$$A \rightarrow X_1 X_2 \dots X_n, \quad n \geq 3$$

mit  $X_i \in N \cup \Sigma$ ,  $1 \leq i \leq n$ , durch die Regeln

$$\begin{aligned} A &\rightarrow X_1 \langle X_2 \dots X_n \rangle \\ \langle X_2 \dots X_n \rangle &\rightarrow X_2 \langle X_3 \dots X_n \rangle \\ &\vdots \\ \langle X_{n-1} X_n \rangle &\rightarrow X_{n-1} X_n \end{aligned}$$

Dabei sind  $\langle X_2 \dots X_n \rangle, \dots, \langle X_{n-1} \dots X_n \rangle$  neue Nichtterminalsymbole.  $\square$

**Bemerkung:** BIN berechnet in  $O(|\mathcal{G}|)$  eine Grammatik der Größe  $O(|\mathcal{G}|)$ .

- Laufe einmal über alle Regeln.
- Für jede Regel der Größe  $n + 1$ ,  $n \geq 3$  fügen wir höchstens  $n - 1$  neue Regeln der Größe 3 hinzu.

Wir definieren die Menge der Nichtterminalsymbole, aus denen das leere Wort abgeleitet werden kann, formal wie folgt:

**Def. 3.7:** Sei  $\mathcal{G} = (\Sigma, N, P, S)$  eine CFG. Definiere die Menge  $\text{Nullable}(\mathcal{G}) \subseteq N$  als

$$\text{Nullable}(\mathcal{G}) = \{A \in N \mid A \vdash^* \mathcal{G}\varepsilon\}. \quad \diamond$$

**Satz 3.4:** Es gibt einen Algorithmus, der  $\text{Nullable}(\mathcal{G})$  in  $O(|\mathcal{G}|^3)$  berechnet.

BEWEIS: Definiere  $M_i$  als die Menge der Nichtterminalsymbole, aus denen sich  $\varepsilon$  mit einem Ableitungsbaum der Höhe  $< i$  ableiten lässt:<sup>21</sup>

$$\begin{aligned} M_0 &= \emptyset \\ M_{i+1} &= M_i \cup \{A \mid A \rightarrow \alpha \in P \text{ und } \alpha \in M_i^*\} \end{aligned}$$

Es gilt für alle  $i \in \mathbb{N}$ , dass  $M_i \subseteq N$ . Da  $|N|$  endlich ist, existiert ein  $m \in \mathbb{N}$ , sodass

$$M_m = M_{m+1} = \bigcup_{i \in \mathbb{N}} M_i$$

Wir können  $\bigcup_{i \in \mathbb{N}} M_i$  mit dem folgenden Verfahren in  $O(|\mathcal{G}|^3)$  berechnen.

<sup>21</sup> Im Folgenden ist mit  $*$  bei  $M_i^*$  und  $M^*$  der Kleene-Stern gemeint.

```

M = {}
done = false
while (not done):
    done = true
    foreach A → a ∈ P:
        if (A ∉ M ∧ a ∈ M*):
            M = M ∪ A
            done = false
return M

```

Wir zeigen nun, dass  $\text{Nullable}(\mathcal{G}) = \bigcup_{i \in \mathbb{N}} M_i$ .

„ $\supseteq$ “ Wir zeigen via Induktion über die Höhe des Ableitungsbaums  $i$ , dass

$$\forall i \in \mathbb{N} : M_i \subseteq \text{Nullable}(\mathcal{G})$$

**IA**  $i = 0$ :  $M_0 = \emptyset \subseteq \text{Nullable}(\mathcal{G})$

**IS**  $i \rightsquigarrow i + 1$

Wenn  $A \in M_{i+1}$ , dann

- gilt entweder  $A \in M_i \subseteq \text{Nullable}(\mathcal{G})$  nach IV oder
- es existiert  $A \rightarrow A_1 \dots A_n \in P$  mit  $A_j \in M_i$  für  $1 \leq j \leq n$ . Nach IV existieren Ableitungen

$$A_j \vdash^* \mathcal{G} \varepsilon$$

sodass auch eine Ableitung von  $A$  nach  $\varepsilon$  existiert:

$$A \vdash_{\mathcal{G}} A_1 \dots A_n \vdash^* \underbrace{\mathcal{G} \varepsilon \dots \varepsilon}_{n \text{ mal}} = \varepsilon$$

Also gilt  $A \in \text{Nullable}(\mathcal{G})$ .

„ $\subseteq$ “ Wenn  $A \in \text{Nullable}(\mathcal{G})$ , dann existiert  $m \in \mathbb{N}$ , sodass  $A \vdash^* \mathcal{G} \varepsilon$  mit einem Ableitungsbaum der Höhe  $m$ . Wir zeigen via Induktion über  $m$ , dass  $A \in M_{m+1} \subseteq \bigcup_{i \in \mathbb{N}} M_i$ .

**IA**  $m = 1$ . Die Ableitung ist  $A \vdash_{\mathcal{G}} \varepsilon$ , sodass  $A \in M_1$ .

**IS**  $m > 1$ . Die Wurzel des Ableitungsbaum muss mit  $A \rightarrow A_1 \dots A_n$  ( $n > 0$ ) beschriftet sein und die Kinder sind jeweils Ableitungsbäume für  $\varepsilon$  in  $\text{Abl}(A_i)$  der Höhe  $m_i \leq m - 1$ , wobei  $1 \leq i \leq n$ .

Es gilt nun nach IV, dass  $A_i \in M_{m_i} \subseteq M_{m-1}$ . Somit ist  $A_i \in M_{m-1}$  und damit, nach Definition,  $A \in M_m$ .  $\square$

**Korollar 3.5:** Man kann zu einer CFG  $\mathcal{G}$  berechnen, ob  $\varepsilon \in L(\mathcal{G})$ .  $\diamond$

BEWEIS: Prüfe, ob  $S \in \text{Nullable}(\mathcal{G})$ .  $\square$

**Lemma 3.6 (DEL):** Zu jeder CFG  $\mathcal{G} = (\Sigma, N, P, S)$  gibt es eine äquivalente CFG  $\mathcal{G}' = (\Sigma, N', P', S')$ , bei der die einzige  $\varepsilon$ -Regel  $S' \rightarrow \varepsilon$  ist (falls  $\varepsilon \in L(\mathcal{G})$ ) und bei der  $S'$  auf keiner rechten Seite einer Regel vorkommt.

BEWEIS:

1. Erweitere  $\mathcal{G}$  um ein neues Startsymbol  $S' \notin N$  mit  $S' \rightarrow S$  als neue Regel. Dieser Schritt stellt sicher, dass  $S'$  auf keiner rechten Regelseite vorkommt.
2. Wende erst SEP, dann BIN an. Nun hat jede rechte Regelseite die Form  $a \in \Sigma$  oder  $\varepsilon$  oder  $A$  oder  $BC$ .
3. Für alle Regeln  $A \rightarrow BC \in P$ :
  - Falls  $B \in \text{Nullable}(\mathcal{G})$  füge  $A \rightarrow C$  hinzu.
  - Falls  $C \in \text{Nullable}(\mathcal{G})$  füge  $A \rightarrow B$  hinzu.
4. Falls  $S \in \text{Nullable}(\mathcal{G})$ , füge  $S' \rightarrow \varepsilon$  hinzu
5. Entferne alle Regeln  $A \rightarrow \varepsilon$  für  $A \neq S'$ .  $\square$

**Bemerkung:** DEL kann in  $O(|\mathcal{G}|^3)$  berechnet werden und die Größe der resultierenden Grammatik ist  $O(|\mathcal{G}|)$ .

- Die höchsten Kosten entstehen beim Berechnen von Nullable.
- In Schritt 3 verdoppelt sich die Anzahl der Regeln höchstens.

**Def. 3.8:** Sei  $\mathcal{G} = (\Sigma, N, P, S)$  eine Grammatik. Eine *Kettenregel* von  $\mathcal{G}$  ist eine Regel in  $P$  der Form  $A \rightarrow B$ , wobei  $A, B \in N$ .  $\diamond$

**Lemma 3.7 (UNIT):** Zu jeder CFG  $\mathcal{G} = (\Sigma, N, P, S)$  gibt es eine äquivalente CFG  $\mathcal{G}' = (N, \Sigma, P', S)$  ohne Kettenregeln.

BEWEIS: Setze zu Anfang  $P' = P$  und eliminiere alle Kettenregeln mit folgendem Algorithmus:

1. Betrachte den gerichteten Graphen  $G$  mit Knoten  $N$  und Kanten für jede Kettenregel, d.h.,  $\{(A, B) \mid A \rightarrow B \in P'\}$ .
2. Suche, z.B. mittels Tiefensuche, einen Zyklus in  $G$ . Wenn kein Zyklus gefunden wurde, fahre mit Schritt 4 fort.
3. Wurde der Zyklus  $A_1 \rightarrow A_2 \rightarrow \dots \rightarrow A_k \rightarrow A_1$  gefunden, dann ersetze in  $P'$  alle Vorkommen von  $A_j$  mit  $j > 1$  durch  $A_1$  (auf linker *und* rechter Regelseite). Entferne dann alle Regeln der Form  $A \rightarrow A$ . Fahre fort mit Schritt 1.



4. Der Graph  $G$  ist ein gerichteter azyklischer Graph (DAG) (engl. *directed acyclic graph*). Sortiere die Knoten von  $G$  topologisch als  $A_1, \dots, A_n$ , sodass  $A_n$  auf keiner linken Seite einer Kettenregel vorkommt.
5. **for**  $j = n - 1, \dots, 1$ 
  - for each**  $A_j \rightarrow A_k \in P'$ 
    - (wegen topologischer Sortierung gilt  $k > j$ )
    - entferne  $A_j \rightarrow A_k$  aus  $P'$
    - füge  $A_j \rightarrow BC$  zu  $P'$  hinzu, falls  $A_k \rightarrow BC \in P'$ ,  $B, C \in N$ .
    - füge  $A_j \rightarrow a$  zu  $P'$  hinzu, falls  $A_k \rightarrow a \in P'$ ,  $a \in \Sigma$ .

Die äußere Schleife hat die Invariante, dass am Ende jedes Durchlaufs für  $n \geq i \geq j$  keine Kettenregel  $A_i \rightarrow \dots$  in  $P'$  existiert. Beim Verlassen der Schleife ist  $j = 1$  und es existieren überhaupt keine Kettenregeln mehr.  $\square$

**Bemerkung:** UNIT kann in  $O(|\mathcal{G}|^3)$  berechnet werden und die Größe der resultierenden Grammatik ist  $O(|\mathcal{G}|^2)$ .

- Die innere Schleife in Schritt 5 wird nur einmal pro Regel ausgeführt.
- In Schritt 5 kommen keine neuen Kettenregeln hinzu.
- Es gibt nur höchstens  $|P'|$  neue Regeln in der inneren Schleife.

**Def. 3.9:** Eine CFG  $\mathcal{G} = (\Sigma, N, P, S)$  ist in Chomsky-Normalform (CNF), falls jede Regel die Form  $A \rightarrow a$ ,  $A \rightarrow BC$ , oder  $S \rightarrow \varepsilon$  hat, wobei  $A, B, C \in N$ ,  $a \in \Sigma$ . Falls  $S \rightarrow \varepsilon \in P$ , dann darf  $S$  auf keiner rechten Seite einer Regel vorkommen.  $\diamond$

Vorlesung:  
29.11.2017

**Satz 3.8:** Zu jeder CFG existiert eine äquivalente CFG in CNF.

BEWEIS: Wir erhalten eine äquivalente CFG in CNF, indem wir der Reihe nach SEP, BIN, DEL und UNIT anwenden.<sup>22</sup>  $\square$

**Bemerkung:** Die Transformation in CNF benötigt Zeit  $O(|\mathcal{G}|^3)$ . Die Größe der CNF-Grammatik ist  $O(|\mathcal{G}|^2)$ .

**Lemma 3.9:** Die Menge der Typ-2-Sprachen ist eine Teilmenge der Typ-1-Sprachen.

BEWEIS: Zu jeder Typ-2-Grammatik existiert eine äquivalente  $\varepsilon$ -freie Typ-2-Grammatik. Diese ist nach Definition auch eine Typ-1-Grammatik.  $\square$

<sup>22</sup>Es genügt sogar nur DEL und dann UNIT anzuwenden, da DEL schon SEP und BIN ausführt.

### 3.3 Wortproblem und Leerheitsproblem für kontextfreie Sprachen

**Satz 3.10:** Es gibt einen Algorithmus, der für eine beliebige kontextfreie Sprache  $L$  das Wortproblem „ $w \in L$ “ in  $O(n^3)$  Schritten und mit  $O(n^2)$  Speicherplatz entscheidet.

BEWEIS (CYK-Algorithmus): Sei  $\mathcal{G}$  eine CFG in CNF und sei  $w = a_1 \dots a_n \in \Sigma^*$ . Der CYK-Algorithmus berechnet bei Eingabe von  $\mathcal{G}$  und  $w$ , ob  $w \in L(\mathcal{G})$ .

Idee: Definiere für  $1 \leq i < j \leq n$ :

$$M_{ij} = \{A \in N \mid A \vdash_{\mathcal{G}}^* a_i \dots a_j\}$$

Es gilt für  $i = j$

$$M_{ii} = \{A \mid A \rightarrow a_i\}$$

Es gilt für  $i < j$

$$\begin{aligned} M_{ij} &= \{A \mid A \rightarrow BC \wedge BC \vdash_{\mathcal{G}}^* a_i \dots a_j\} \\ &= \{A \mid A \rightarrow BC \wedge \exists k : i \leq k < j \quad B \vdash_{\mathcal{G}}^* a_i \dots a_k \wedge C \vdash_{\mathcal{G}}^* a_{k+1} \dots a_j\} \\ &= \{A \mid A \rightarrow BC \wedge \exists k : i \leq k < j \quad B \in M_{ik} \wedge C \in M_{(k+1)j}\} \end{aligned}$$

Es gilt:

- Wir können die  $M_{ij}$  als eine  $(n \times n)$ -Matrix  $M$  mit Einträgen in  $\mathcal{P}(N)$  darstellen.
- Der Wert von  $M_{ij}$  hängt nur von  $M_{i'j'}$  mit  $i' > i, j' \leq j$  oder  $i' \geq i, j' < j$  ab.
- $w \in L$  genau dann, wenn  $S \vdash^* w$  genau dann, wenn  $S \in M_{1n}$ .

Wir können die  $M_{ij}$  mit dem folgenden Verfahren ermitteln:

```

CYK( $\mathcal{G}, a_1, \dots, a_n$ )
 $M = n \times n$ -Matrix mit (initial)  $M_{ij} = \emptyset$  für alle  $i, j$  //  $O(n^2)$ 
for  $i=1 \dots n$  do //  $O(|\mathcal{G}|) \cdot O(n)$ 
     $M_{ii} = \{A \mid A \rightarrow a_i\}$ 
for  $i=n-1 \dots 1$  do
    for  $j=i+1 \dots n$  do
        for  $k=i \dots j-1$  do //  $O(|\mathcal{G}|) \cdot O(n^3)$ 
             $M_{ij} = M_{ij} \cup \{A \mid A \rightarrow BC, B \in M_{ik}, C \in M_{(k+1)j}\}$ 
return  $S \in M_{1n}$ 

```

□

**Bsp. 3.8:**  $L = \{a^n b^n c^m \mid n, m \geq 1\}$  mit Grammatik (CFG) mit folgenden Regeln.

$$\begin{aligned} S &\rightarrow XY \\ X &\rightarrow ab \mid aXb \\ Y &\rightarrow c \mid cY \end{aligned}$$

In CNF:  $S \rightarrow XY$

$$X \rightarrow AB \mid AZ$$

$$Z \rightarrow XB$$

$$Y \rightarrow CY \mid c$$

$$A \rightarrow a, B \rightarrow b, C \rightarrow c$$

Beispiel der Matrix für  $w = aaabbbcc$ . Zellen, die mit „•“ markiert sind, sind leer. Die Reihenfolge, in der die nicht-diagonalen, nicht leeren Zellen eingetragen wurden, ist mit kleinen Zahlen markiert. Beispielsweise wurde  $M_{35} = \{Z\}$  mit Zahl „3“ nach  $M_{78} = \{Y\}$  mit Zahl „1“ eingetragen. Da  $S \in M_{1n} = \{S\}$ , gilt  $w \in L$ .

	A	•	•	•	•	$X_6$	$S_7$	$S_8$
a	A	•	•	$X_4$	$Z_5$	•	•	
	a	A	$X_2$	$Z_3$	•	•	•	
		a	B	•	•	•	•	
			b	B	•	•	•	
				b	B	•	•	
					b	$C, Y$	$Y_1$	
						c	$C, Y$	
							c	

**Satz 3.11:** Es gibt einen Algorithmus, der das Leerheitsproblem<sup>23</sup> für kontextfreie Sprachen in  $O(|\mathcal{G}|)$  entscheidet.

BEWEIS: Sei  $\mathcal{G} = (\Sigma, N, P, S)$  eine CFG für  $L$ . Wir definieren die Menge  $M$  der co-erreichbaren Nichtterminalsymbole.

$$M = \{A \mid A \vdash_{\mathcal{G}}^* w, w \in \Sigma^*\}$$

Offensichtlich gilt:  $L = \emptyset \Leftrightarrow S \notin M$ .

<sup>23</sup>Gefragt ist, ob  $L(\mathcal{G}) = \{w \in \Sigma^* \mid S \vdash_{\mathcal{G}}^* w\} \stackrel{?}{=} \emptyset$ .

Wir können  $M$  induktiv berechnen. Wir definieren hierfür:

$$\begin{aligned} M_0 &= \{A \mid A \rightarrow w \in P, w \in \Sigma^*\} \\ M_{i+1} &= M_i \cup \{A \mid A \rightarrow \alpha \in P, \alpha \in (\Sigma \cup M_i)^*\} \end{aligned}$$

Offensichtlich gilt für alle  $i \in \mathbb{N}$   $M_i \subseteq M_{i+1}$  und  $M_i \subseteq M$ . Da  $M \subseteq N$  endlich ist, gibt es ein  $n$ , sodass  $M_n = M_{n+1}$  (hier ohne Beweis).

Wir können daher  $M$  analog zu  $\text{Nullable}(\mathcal{G})$  (siehe Satz 3.4) in  $O(|\mathcal{G}|)$  berechnen.  $\square$

**Bem.:** Wenn wir in einer Grammatik  $\mathcal{G}$  ein nicht co-erreichbares Nichtterminalsymbol (und alle zugehörigen Regeln) entfernen, ändert sich  $L(\mathcal{G})$  nicht. Wir können eine gegebene Grammatik also optimieren, indem wir alle nicht co-erreichbaren Nichtterminalsymbole entfernen.

### 3.4 Einschub: Typ-3-Sprachen

**Satz 3.12:**  $L$  ist regulär  $\Leftrightarrow L$  ist Typ-3-Sprache.

Vorlesung:  
1.12.2017

BEWEIS: „ $\Rightarrow$ “ Sei  $\mathcal{A} = (\Sigma, Q, \delta, q^{\text{init}}, F)$  DEA für  $L$ .

Konstruiere Typ-3-Grammatik  $\mathcal{G} = (\Sigma, N, P, S)$  mit

$$\begin{aligned} N &= Q \\ S &= q^{\text{init}} \\ P &= \{q \rightarrow aq' \mid q, q' \in Q, a \in \Sigma, \delta(q, a) = q'\} \cup \{q \rightarrow \varepsilon \mid q \in F\} \end{aligned}$$

Es gilt  $L(\mathcal{G}) = L(\mathcal{A})$ . (ohne Beweis)

„ $\Leftarrow$ “ Sei  $\mathcal{G} = (\Sigma, N, P, S)$  Typ-3-Grammatik. Dann erhalten wir durch Anwenden von BIN eine äquivalente Grammatik, in der jede Regel eine der folgenden Formen hat:

$$\begin{aligned} A &\rightarrow \varepsilon && \text{für } A \in N \\ A &\rightarrow a && \text{für } A \in N, a \in \Sigma \\ A &\rightarrow a_1 a_2 && \text{für } A \in N, a_1, a_2 \in \Sigma \\ A &\rightarrow aB && \text{für } A \in N, a \in \Sigma, B \in N \end{aligned}$$

Wir machen nun die folgenden Modifikationen.

- Wir fügen eine frische Variable  $Y_\varepsilon$  und die Regel  $Y_\varepsilon \rightarrow \varepsilon$  hinzu.
- Wir löschen jede Regel der zweiten Form und führen stattdessen die Regel  $A \rightarrow aY_\varepsilon$  ein.

- Wir löschen jede Regel der dritten Form und führen stattdessen eine frische Variable  $Y_{a_2}$  und die Regeln  $A \rightarrow a_1 Y_{a_2}$  und  $Y_{a_2} \rightarrow a_2 Y_\varepsilon$  ein.

Die resultierende Grammatik  $\mathcal{G} = (\Sigma, N', P', S)$  ist äquivalent zu  $\mathcal{G}$ . (Ohne Beweis)

Wir konstruieren nun den folgenden NEA.  $\mathcal{N} = (\Sigma, Q, \delta, q^{\text{init}}, F)$

$$\begin{aligned} Q &= N' \\ B \in \delta(A, a) &\text{ gdw } A \rightarrow aB \in P' \\ q^{\text{init}} &= S \\ F &= \{A \in N' \mid A \rightarrow \varepsilon \in P'\} \end{aligned}$$

Es gilt  $L(\mathcal{G}) = L(\mathcal{A})$ . (ohne Beweis)

□

**Lemma 3.13:**  $\mathcal{M}_3 \subsetneq \mathcal{M}_2$  (d.h., die regulären Sprachen sind eine *echte* Teilmenge der kontextfreien Sprachen).

BEWEIS: Betrachte  $L_{\text{centered}} = \{0^n 1 0^n \mid n \in \mathbb{N}\} \subseteq \{0, 1\}^*$

Sowohl aus Bsp. 2.11 als auch aus Bsp. 2.13 ist bekannt, dass  $L_{\text{centered}}$  nicht regulär ist.

Es gibt aber eine Typ-2-Grammatik für  $L_{\text{centered}}$ :

$$\mathcal{G} = (\{0, 1\}, \{S\}, \{S \rightarrow 1, S \rightarrow 0S0\}, S)$$

(ohne Beweis)

□

## 4 Kellerautomaten pushdown automaton (PDA)

TODO: Dieser rote Teil wird noch überarbeitet.

Kellerautomat  $\approx$  Endlicher Automat + Kellerspeicher von unbeschränkter Größe (Stack, push down)

**Neu:**

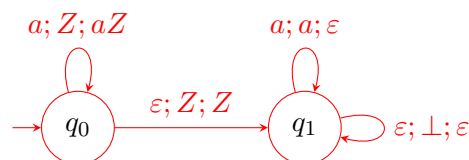
- Bei jedem Schritt darf der PDA das oberste Kellersymbol inspizieren und durch ein beliebiges Kellerwort ersetzen (das neue Kellerpräfix).
- Der PDA darf auf dem Keller rechnen, ohne in der Eingabe weiterzulesen ( $\varepsilon$ -Transition oder Spontantransition).

**Bsp. 4.1:**

$\Sigma = \{0, 1\}$	Eingabealphabet	
$\Gamma = \{0, 1, \perp\}$	Kelleralphabet	
$Q = \{q_0, q_1\}$	$\perp \hat{=}$ Kellerbodensymbol	
$\delta(q_0, a, Z) = \{(q_0, aZ)\}$	$a \in \{0, 1\}, Z \in \Gamma$	(1)
$\delta(q_0, \varepsilon, Z) = \{(q_1, Z)\}$		(2)
$\delta(q_1, a, a) = \{(q_1, \varepsilon)\}$		(3)
$\delta(q_1, \varepsilon, \perp) = \{(q_1, \varepsilon)\}$		(4)

Die Übergangsfunktion  $\delta$  bildet den aktuellen Zustand, das aktuelle Eingabesymbol und das aktuell oberste Kellersymbol auf Paare von Folgezustand und neuem Kellerpräfix ab. In diesem Beispiel ist die zurückgegebene Menge von Paaren in allen Fällen einelementig. Durch die  $\varepsilon$ -Transitionen ist der PDA aber trotzdem nichtdeterministisch.

Bei der graphischen Darstellung werden die Transitionen mit Tripeln  $(a; Z; \gamma)$  beschriftet, wobei  $a \in \Sigma$  das Eingabesymbol,  $Z \in \Gamma$  das oberste Kellersymbol und  $\gamma \in \Gamma^*$  der neue Kellerpräfix ist:



wobei hier  $a \in \Sigma$  und  $Z \in \Gamma$  gilt.

Der Automat beginnt im Startzustand  $q^{\text{init}}$  mit einem Kellerspeicher, der nur das Kellerbodensymbol  $Z^{\text{init}}$  enthält. Er akzeptiert ein Wort, wenn er alle Eingabesymbole gelesen hat und den Keller komplett leeren konnte. Anders als bei endlichen Automaten gibt es *keine* akzeptierenden Zustände.

Die akzeptierte Sprache ist hier  $L = \{ww^R \mid w \in \{0,1\}^*\}$ .

**Def. 4.1:** Ein nichtdeterministischer Kellerautomat (NPDA)<sup>24</sup> ist ein 6-Tupel

$$(\Sigma, Q, \Gamma, q^{\text{init}}, Z^{\text{init}}, \delta).$$

Dabei ist

- $\Sigma$  ein Alphabet, das wir auch *Eingabealphabet* nennen,
- $Q$  eine endliche Menge, deren Elemente wir *Zustände* nennen,
- $\Gamma$  ein Alphabet, das wir auch *Kelleralphabet* nennen,
- $q^{\text{init}} \in Q$  ein Zustand, den wir Startzustand nennen,
- $Z^{\text{init}} \in \Gamma$  ein Zeichen, das wir *Kellerbodensymbol* nennen,
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma^*)$  eine Funktion, die wir *Transitionsfunktion* nennen.  $\diamond$

Im Weiteren sei  $\mathcal{K} = (\Sigma, Q, \Gamma, q^{\text{init}}, Z^{\text{init}}, \delta)$  ein NPDA.

**Def. 4.2:** Die Menge der *Konfigurationen* von  $\mathcal{K}$  ist  $\text{Konf}(\mathcal{K}) = Q \times \Sigma^* \times \Gamma^*$ .

Die *Schrittrelation* von  $\mathcal{K}$

$$\triangleright \subseteq \text{Konf}(\mathcal{K}) \times \text{Konf}(\mathcal{K})$$

ist definiert durch

$$\begin{aligned} (q, aw, Z\gamma) &\triangleright (q', w, \beta\gamma) && \text{falls } \delta(q, a, Z) \ni (q', \beta) \\ (q, w, Z\gamma) &\triangleright (q', w, \beta\gamma) && \text{falls } \delta(q, \varepsilon, Z) \ni (q', \beta). \end{aligned}$$

Wir schreiben  $(q, w, \gamma) \triangleright^n (q', w', \gamma')$ , wenn  $\mathcal{K}$  in  $n \in \mathbb{N}$  Schritten von Konfiguration  $(q, w, \gamma)$  in Konfiguration  $(q', w', \gamma')$  gelangt.

Wir schreiben  $\triangleright^*$  für die reflexive, transitive Hülle von  $\triangleright$ . Falls  $(q, w, \gamma) \triangleright^* (q', w', \gamma')$ , so existiert also  $n \in \mathbb{N}$ , sodass  $(q, w, \gamma) \triangleright^n (q', w', \gamma')$ .

Die von  $\mathcal{K}$  *akzeptierte Sprache* ist

$$L(\mathcal{K}) = \{w \in \Sigma^* \mid \exists q \in Q : (q^{\text{init}}, w, Z^{\text{init}}) \triangleright^* (q, \varepsilon, \varepsilon)\}$$

Wir nennen eine Konfiguration der Form  $(q, \varepsilon, \varepsilon)$  mit  $q \in Q$  eine *akzeptierende Konfiguration*.  $\diamond$

**Bsp.:** Die folgenden Schritte von  $\mathcal{K}$  aus Beispiel 4.1 zeigen, dass  $w = 0110 \in L(\mathcal{K})$ :

- $$\begin{aligned}
 & (q^{\text{init}}, 0110, \perp) \\
 \triangleright & (q^{\text{init}}, 110, 0\perp) \quad (\text{„pushen“ des Eingabesymbols } 0) \\
 \triangleright & (q^{\text{init}}, 10, 10\perp) \quad (\text{„pushen“ des Eingabesymbols } 1) \\
 \triangleright & (q_1, 10, 10\perp) \quad (\varepsilon\text{-Übergang von } q^{\text{init}} \text{ nach } q_1) \\
 \triangleright & (q_1, 0, 0\perp) \quad (\text{„poppen“ des Eingabesymbols } 1) \\
 \triangleright & (q_1, \varepsilon, \perp) \quad (\text{„poppen“ des Eingabesymbols } 0) \\
 \triangleright & (q_1, \varepsilon, \varepsilon) \quad (\varepsilon\text{-Übergang zum Entfernen von } \perp)
 \end{aligned}$$

**Lemma 4.1:** Zu jeder CFG  $\mathcal{G}$  gibt es einen NPDA  $\mathcal{K}$ , sodass  $L(\mathcal{K}) = L(\mathcal{G})$ .

Zum Führen des Beweises benötigen wir das folgende Lemma.

**Lemma 4.2** (Mehr Keller – mehr Möglichkeiten.): Für alle  $q, q' \in Q$ ,  $w \in \Sigma^*$ ,  $Z \in \Gamma$  und  $n \in \mathbb{N}$  gilt:

Wenn  $(q, w, Z) \triangleright^n (q', \varepsilon, \varepsilon)$ , dann  $\forall v \in \Sigma^*, \gamma \in \Gamma^* : (q, wv, Z\gamma) \triangleright^n (q', v, \gamma)$ .

BEWEIS: Übungsaufgabe. □

BEWEIS (von Lemma 4.1): Sei  $\mathcal{G} = (\Sigma, N, P, S)$  eine kontextfreie Grammatik für  $L$  in CNF. Definiere einen NPDA  $\mathcal{K} = (\Sigma, Q, \Gamma, q^{\text{init}}, Z^{\text{init}}, \delta)$  durch:

- $Q = \{q^{\text{init}}\}$
- $\Gamma = \Sigma \cup N$
- $Z^{\text{init}} = S$
- $\delta(q^{\text{init}}, a, a) = \{(q^{\text{init}}, \varepsilon)\}$  für  $a \in \Sigma$
- $\delta(q^{\text{init}}, \varepsilon, A) = \{(q^{\text{init}}, \alpha)\}$  für  $A \rightarrow \alpha \in P$

Wir zeigen nun, dass  $L(\mathcal{K}) = L(\mathcal{G})$ .

- Beweisrichtung „ $\supseteq$ “

Sei  $w \in L(\mathcal{G})$ , dann gilt  $S \vdash^* w$ . Wir wollen zeigen dass  $(q^{\text{init}}, w, S) \triangleright^* (q^{\text{init}}, \varepsilon, \varepsilon)$  folgt und beweisen dafür via vollständige Induktion über die Länge der Ableitung die folgende stärkere Eigenschaft.

$$\forall A \in N : \text{ wenn } A \vdash^* w, \text{ dann } (q^{\text{init}}, w, A) \triangleright^* (q^{\text{init}}, \varepsilon, \varepsilon)$$

I.A.:  $n = 1$ : Es gibt nur zwei Arten von Regeln, die für Ableitungen der Länge 1 in Frage kommen:

---

<sup>24</sup>Englisch: Nondeterministic Pushdown Automaton



- $S \rightarrow \varepsilon$ . Per Konstruktion gilt  $(q^{\text{init}}, \varepsilon) \in \delta(q^{\text{init}}, \varepsilon, S)$  und somit  $(q^{\text{init}}, \varepsilon, S) \triangleright (q^{\text{init}}, \varepsilon, \varepsilon)$ .
  - $A \rightarrow a, a \in \Sigma$ . Per Konstruktion gilt  $(q^{\text{init}}, a) \in \delta(q^{\text{init}}, \varepsilon, A)$  und  $(q^{\text{init}}, \varepsilon) \in \delta(q^{\text{init}}, a, a)$ .
- Somit gilt  $(q^{\text{init}}, a, A) \triangleright (q^{\text{init}}, a, a) \triangleright (q^{\text{init}}, \varepsilon, \varepsilon)$ .

I.S.:  $n \rightsquigarrow n + 1$ : In diesem Fall verwendet der erste Ableitungsschritt eine Regel der Form  $\pi = A \rightarrow BC$  mit  $B, C \in N$ . Wir betrachten den zugehörigen Ableitungsbaum  $\mathcal{T} = \pi(\mathcal{T}_1, \mathcal{T}_2)$ ,  $\mathcal{T}_1 \in \text{Abl}(B)$ ,  $\mathcal{T}_2 \in \text{Abl}(C)$ ,  $w = Y(\mathcal{T}_1)Y(\mathcal{T}_2)$ .

Per Konstruktion gilt  $(q^{\text{init}}, BC) \in \delta(q^{\text{init}}, \varepsilon, A)$ .

Ferner gilt per I.V., dass  $(q^{\text{init}}, Y(\mathcal{T}_1), B) \triangleright^* (q^{\text{init}}, \varepsilon, \varepsilon)$  und  $(q^{\text{init}}, Y(\mathcal{T}_2), C) \triangleright^* (q^{\text{init}}, \varepsilon, \varepsilon)$ .

Mit Lemma 4.2 folgt schließlich:

$$\begin{aligned} (q^{\text{init}}, w, A) &= (q^{\text{init}}, Y(\mathcal{T}_1)Y(\mathcal{T}_2), A) \triangleright (q^{\text{init}}, Y(\mathcal{T}_1)Y(\mathcal{T}_2), BC) \\ &\triangleright^* (q^{\text{init}}, Y(\mathcal{T}_2), C) \\ &\triangleright^* (q^{\text{init}}, \varepsilon, \varepsilon). \end{aligned}$$

• Beweisrichtung „ $\subseteq$ “

Sei  $w \in L(\mathcal{K})$ , dann gilt  $(q^{\text{init}}, w, S) \triangleright^* (q^{\text{init}}, \varepsilon, \varepsilon)$ . Wir wollen  $S \vdash^* w$  folgern und beweisen dafür via vollständige Induktion über die Anzahl der Berechnungsschritte  $n$  die folgende Aussage:

$$\forall n \in \mathbb{N} : \forall w \in \Sigma^*, \alpha \in \Gamma^* : \text{ wenn } (q^{\text{init}}, w, \alpha) \triangleright^n (q^{\text{init}}, \varepsilon, \varepsilon), \text{ dann } \alpha \vdash^* w$$

**I.A.**  $n = 0$ .  $w = \varepsilon, \alpha = \varepsilon$ : Es gilt  $\varepsilon \vdash^* \varepsilon$ .

**I.S.**  $n > 0, \alpha = Z\alpha'$ : Es gilt  $(q^{\text{init}}, w, Z\alpha') \triangleright (q^{\text{init}}, w', \beta\alpha') \triangleright^{n-1} (q^{\text{init}}, \varepsilon, \varepsilon)$ ,  $\delta(q^{\text{init}}, x, Z) \ni (q^{\text{init}}, \beta), x \in \Sigma \cup \{\varepsilon\}$ .

Es gibt zwei Fälle für  $Z$ :

- $Z = a$  für  $a \in \Sigma$ . Es folgt  $\beta = \varepsilon, w = aw'$  und  $x = a$ .

Per I.V. gilt  $\alpha' \vdash^* w'$  und somit auch  $\alpha = a\alpha' \vdash^* aw' = w$ .

- $Z = A$  mit  $A \rightarrow \beta \in P$ . Es folgt  $w = w'$  und  $x = \varepsilon$ .

Per I.V. gilt  $\beta\alpha' \vdash^* w'$  und somit auch  $A\alpha \vdash^* \beta\alpha' \vdash^* w' = w$ . □

Bei einem NPDA nehmen wir in jedem Schritt ein Zeichen vom Kellerspeicher herunter, aber dürfen beliebig viele Symbole auf den Kellerspeicher zurücklegen. Was wäre, wenn die Höhe des Kellers pro Schritt höchstens um eins größer werden könnte? Verlören wir dadurch nur Komfort oder könnten wir dadurch weniger Sprachen akzeptieren?

Das folgende Lemma beantwortet diese Frage.

**Lemma 4.3:** Zu jedem NPDA gibt es einen äquivalenten NPDA, bei dem für alle  $q, q' \in Q, Z \in \Gamma, \gamma \in \Gamma^*, x \in \Sigma \cup \{\varepsilon\}$  gilt: Falls  $(q', \gamma) \in \delta(q, x, Z)$ , dann  $|\gamma| \leq 2$ .

BEWEIS: Sei  $(q', \gamma) \in \delta(q, x, Z)$  mit  $\gamma = Z_n \dots Z_1$  für  $n > 2$ :

- neue Zustände  $q_2 \dots q_{n-1}$
- Ersetze  $(q', \gamma)$  durch  $(q_2, Z_2 Z_1)$
- Definiere  $\delta(q_i, \varepsilon, Z_i) = \{(q_{i+1}, Z_{i+1} Z_i)\}$ , für  $2 \leq i < n-1$
- Definiere  $\delta(q_{n-1}, \varepsilon, Z_{n-1}) = \{(q', Z_n Z_{n-1})\}$

Wiederhole bis alle Transitionen die gewünschte Form haben. Die Sprache des NPDA ändert sich durch diese Transformation nicht (ohne Beweis).  $\square$

**Lemma 4.4:** Zu jedem NPDA  $\mathcal{K}$  gibt es eine CFG  $\mathcal{G}$ , sodass  $L(\mathcal{K}) = L(\mathcal{G})$ .

Vorlesung:  
8.12.2017

BEWEIS: Ohne Einschränkung (Lemma 4.3) sei  $\mathcal{K} = (Q, \Sigma, \Gamma, q^{\text{init}}, Z^{\text{init}}, \delta)$  ein NPDA mit  $|\gamma| \leq 2$  für alle  $(q', \gamma) \in \delta(q, x, Z), q, q' \in Q, x \in \Sigma \cup \{\varepsilon\}, Z \in \Gamma$ .

Definiere  $\mathcal{G} = (\Sigma, N, P, S)$  mit  $N = (Q \times \Gamma \times Q) \cup \{S\}$ . Bevor wir  $P$  beschreiben, wollen wir zunächst die Intuition hinter den Nichtterminalsymbolen vermitteln.

Die dieser Konstruktion zugrunde liegende Idee wird durch die folgende Eigenschaft beschrieben:

$$[q, Z, q'] \vdash^n w \text{ gdw } (q, w, Z) \triangleright^n (q', \varepsilon, \varepsilon) \quad (*)$$

In Worten: Wir können vom Nichtterminalsymbol  $[q, Z, q']$  das Wort  $w$  genau dann in  $n$  Schritten ableiten, wenn der NPDA die Konfiguration  $(q, w, Z)$  in  $n$  Schritten in eine akzeptierende Konfiguration mit Zustand  $q'$  überführen kann.

Mit dieser Intuition können wir nun die Menge der Produktionsregeln angeben. Wir definieren  $P = P_S \cup P_0 \cup P_1 \cup P_2$  mit:

$$P_S = \{S \rightarrow [q^{\text{init}}, Z^{\text{init}}, q'] \mid q' \in Q\},$$

$$P_0 = \{[q, Z, q'] \rightarrow x \mid (q', \varepsilon) \in \delta(q, x, Z), x \in \Sigma \cup \{\varepsilon\}\},$$

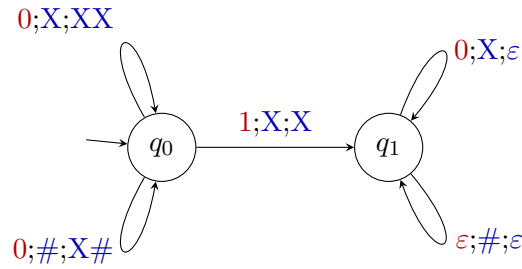
$$P_1 = \{[q, Z, q'] \rightarrow x[q'', Z', q'] \mid (q'', Z') \in \delta(q, x, Z), q' \in Q, x \in \Sigma \cup \{\varepsilon\}\},$$

$$P_2 = \{[q, Z, q'] \rightarrow x[q_1, Z_1, q_2][q_2, Z_2, q'] \mid (q_1, Z_1 Z_2) \in \delta(q, x, Z), q', q_2 \in Q, x \in \Sigma \cup \{\varepsilon\}\}$$

Dabei unterscheiden  $P_0$ ,  $P_1$  und  $P_2$  die Fälle, dass der Stack kleiner wird ( $P_0$ ), gleich hoch bleibt ( $P_1$ ) oder größer wird ( $P_2$ ).

Beispiel:

NPDA für  $L_{\text{centered3}} := \{0^n 10^n \mid n \in \mathbb{N}, n > 0\} \subseteq \{0, 1\}^*$ :



010 wird akzeptiert, weil  $(q_0, 010, \#) \triangleright (q_0, 10, X\#) \triangleright (q_1, 0, X\#) \triangleright (q_1, \varepsilon, \#) \triangleright (q_1, \varepsilon, \varepsilon)$ .

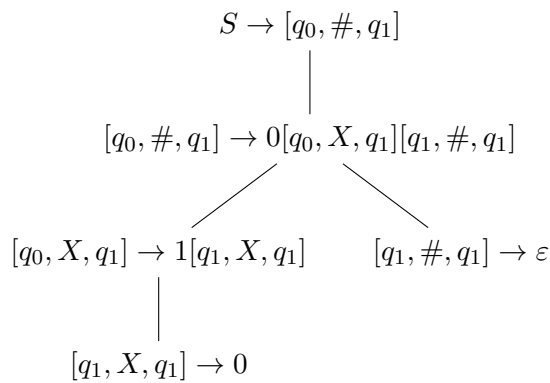
$$P_S = \{S \rightarrow [q_0, \#, q_0] \\ S \rightarrow [q_0, \#, q_1]\}$$

$$P_2 = \{[q_0, \#, q_0] \rightarrow 0[q_0, X, q_0][q_0, \#, q_0] \\ [q_0, \#, q_0] \rightarrow 0[q_0, X, q_1][q_1, \#, q_0] \\ [q_0, \#, q_1] \rightarrow 0[q_0, X, q_0][q_0, \#, q_1] \\ [q_0, \#, q_1] \rightarrow 0[q_0, X, q_1][q_1, \#, q_1] \\ [q_0, X, q_0] \rightarrow 0[q_0, X, q_0][q_0, X, q_0] \\ [q_0, X, q_0] \rightarrow 0[q_0, X, q_1][q_1, X, q_0] \\ [q_0, X, q_1] \rightarrow 0[q_0, X, q_0][q_0, X, q_1] \\ [q_0, X, q_1] \rightarrow 0[q_0, X, q_1][q_1, X, q_1]\}$$

$$P_0 = \{[q_1, X, q_1] \rightarrow 0 \\ [q_1, \#, q_1] \rightarrow \varepsilon\}$$

$$P_1 = \{[q_0, X, q_0] \rightarrow 1[q_1, X, q_0] \\ [q_0, X, q_1] \rightarrow 1[q_1, X, q_1]\}$$

Ableitungsbaum für 010:



Offensichtlich folgt aus dieser Eigenschaft  $L(\mathcal{G}) = L(\mathcal{K})$ , da wir vom Startsymbol  $S$  genau die Nichtterminalsymbole der Form  $[q^{\text{init}}, Z^{\text{init}}, q']$  ableiten können und der NPDA genau die Wörter  $w$  akzeptiert, für die  $(q^{\text{init}}, w, Z^{\text{init}})$  in eine akzeptierende Konfiguration überführt werden kann.

Es bleibt zu zeigen, dass die Eigenschaft  $(*)$  gilt. Für den Fall  $n = 1$  folgt die Eigenschaft direkt aus den  $P_0$ -Regeln (und der Tatsache, dass keine der anderen Regeln ein einzelnes Symbol oder  $\varepsilon$  ableiten kann).

- Beweisrichtung „ $\Rightarrow$ “ Wir zeigen diese Richtung via vollständige Induktion über die Anzahl der Ableitungsschritte.

**I.A.**  $n = 1$ : Folgt wie oben erwähnt aus den  $P_0$ -Regeln.

**I.S.**  $n \rightsquigarrow n + 1$ : Sei  $[q, Z, q'] \vdash \alpha \vdash^n w$  eine Ableitung für  $w$ . Für den ersten Ableitungsschritt kommen zwei Fälle in Frage:

- Fall 1:  $\alpha$  hat die Form  $x[q'', Z', q']$  ( $[q, Z, q'] \rightarrow \alpha \in P_1$ ).

Es folgt  $w = xw'$  für ein  $w'$  mit  $[q'', Z', q'] \vdash^n w'$ .

Nach I.V. gilt nun, dass  $(q'', w', Z') \triangleright^n (q', \varepsilon, \varepsilon)$ .

Nach Konstruktion gilt  $\delta(q, x, Z) \ni (q'', Z'), q'' \in Q, x \in \Sigma \cup \{\varepsilon\}$  und somit  $(q, w, Z) \triangleright (q'', w', Z') \triangleright^n (q', \varepsilon, \varepsilon)$ .

- Fall 2:  $\alpha$  hat die Form  $x[q_1, Z_1, q_2][q_2, Z_2, q']$  ( $[q, Z, q'] \rightarrow \alpha \in P_2$ ).

Es gibt also einen Ableitungsbaum  $\mathcal{T} = \pi(\mathcal{T}_1, \mathcal{T}_2)$ , mit  $\mathcal{T}_1 \in \text{Abl}([q_1, Z_1, q_2])$  und  $\mathcal{T}_2 \in \text{Abl}([q_2, Z_2, q'])$ , sodass  $w = xY(\mathcal{T}_1)Y(\mathcal{T}_2)$ .

Es gilt also  $[q_1, Z_1, q_2] \vdash^{k_1} Y(\mathcal{T}_1)$  und  $[q_2, Z_2, q'] \vdash^{k_2} Y(\mathcal{T}_2)$  für  $k_1, k_2 \in \mathbb{N}$ , sodass  $k_1 + k_2 = n$ .

Per I.V. folgt

$$(q_1, Y(\mathcal{T}_1), Z_1) \triangleright^{k_1} (q_2, \varepsilon, \varepsilon).$$

Mit Lemma 4.2 folgt auch

$$(q_1, Y(\mathcal{T}_1)Y(\mathcal{T}_2), Z_1Z_2) \triangleright^{k_1} (q_2, Y(\mathcal{T}_2), Z_2).$$

Auch hier können wir wieder I.V. anwenden und erhalten:

$$(q_2, Y(\mathcal{T}_2), Z_2) \triangleright^{k_2} (q', \varepsilon, \varepsilon)$$

Somit gilt:

$$(q, \underbrace{xY(\mathcal{T}_1)Y(\mathcal{T}_2)}_w, Z) \triangleright (q_1, Y(\mathcal{T}_1)Y(\mathcal{T}_2), Z_1Z_2) \triangleright^{k_1} (q_2, Y(\mathcal{T}_2), Z_2) \triangleright^{k_2} (q', \varepsilon, \varepsilon)$$

Beweisrichtung „ $\Leftarrow$ “ Wir zeigen diese Richtung via vollständige Induktion über die Anzahl der Berechnungsschritte des NPDA.

Vorlesung:  
13.12.2017

**I.A.**  $n = 1$ : Folgt wie oben erwähnt aus den  $P_0$ -Regeln.

**I.S.**  $n \rightsquigarrow n + 1$ :

Sei  $(q, w, Z) \triangleright^{n+1} (q', \varepsilon, \varepsilon)$  eine Berechnung für  $w$ . Da wir annehmen, dass der NPDA in einem Schritt höchstens zwei Symbole auf den Stack legen darf, kommen für den ersten Berechnungsschritt nur zwei Fälle in Frage:

- Fall 1:  $(q, w, Z) \triangleright (q'', w', Z')$  mit  $w = xw'$  für  $x \in \Sigma \cup \{\varepsilon\}$   
Aus  $(q'', w', Z') \triangleright^n (q', \varepsilon, \varepsilon)$  folgern wir mit I.V.  $[q'', Z', q'] \vdash^n w'$ . Nach Konstruktion können wir  $[q, Z, q'] \vdash x[q'', Z', q']$  ableiten.
- Fall 2:  $(q, w, Z) \triangleright (q_1, w', Z_1 Z_2)$  mit  $w = xw'$  für  $x \in \Sigma \cup \{\varepsilon\}$   
Aus  $(q_1, w', Z_1 Z_2) \triangleright^n (q', \varepsilon, \varepsilon)$  folgern wir:  $\exists k_1, k_2 \in \mathbb{N}, q_2 \in Q, w_1, w_2 \in \Sigma^*$ , sodass gilt:
  1.  $k_1 + k_2 = n$
  2.  $w' = w_1 w_2$
  3.  $(q_1, w', Z_1 Z_2) \triangleright^{k_1} (q_2, w_2, Z_2)$
  4.  $(q_2, w_2, Z_2) \triangleright^{k_2} (q', \varepsilon, \varepsilon)$

Wir wählen  $k_1$  als die kleinste Zahl, welche die obigen Eigenschaften erfüllt. Damit wissen wir, dass wir  $Z_2$  noch nie vom Keller genommen haben.

Aus 3. folgern wir  $(q_1, w_1, Z_1) \triangleright^{k_1} (q_2, \varepsilon, \varepsilon)$  und mit I.V.  $[q_1, Z_1, q_2] \vdash^{k_1} w_1$ .

Aus 4. folgern wir mit I.V.  $[q_2, Z_2, q'] \vdash^{k_2} w_2$ .

Es gibt also einen Ableitungsbaum  $\mathcal{T} = \pi(\mathcal{T}_1, \mathcal{T}_2)$ , mit  $\mathcal{T}_1 \in \text{Abl}([q_1, Z_1, q_2])$  und  $\mathcal{T}_2 \in \text{Abl}([q_2, Z_2, q'])$ , sodass  $w = xY(\mathcal{T}_1)Y(\mathcal{T}_2)$ .

Somit gibt es auch eine Ableitung

$$[q, Z, q'] \vdash x[q_1, Z_1, q_2][q_2, Z_2, q'] \vdash^{k_1+k_2} xw_1w_2. \quad \square$$

**Satz 4.5:** Die Menge der von Kellerautomaten akzeptierten Sprachen und die Menge der kontextfreien Sprachen sind identisch.

BEWEIS: Folgt direkt aus Lemma 4.1 und Lemma 4.4.  $\square$

**Bemerkung:** Zu jedem NPDA gibt es einen äquivalenten NPDA, der nur einen Zustand hat. (Folgt aus Lemma 4.4 und der Konstruktion aus dem Beweis von Lemma 4.1.)

## 5 Kontextfreie Sprachen – Teil 2

### 5.1 Das Pumping Lemma für kontextfreie Sprachen

**Satz 5.1** (Pumping Lemma für CFL): Sei  $L$  eine kontextfreie Sprache. Dann gilt:

$$\begin{aligned} \exists n \in \mathbb{N}, n > 0 : \quad & \forall z \in L, |z| \geq n : \\ & \exists u, v, w, x, y \in \Sigma^* : \\ & z = uvwxy, |vwx| \leq n, |vx| \geq 1 \\ \text{und } \forall i \in \mathbb{N} : & uv^iwx^iy \in L \end{aligned}$$

**Lemma 5.2:** In jedem Binärbaum (Baum, bei dem jeder innere Knoten zwei Nachfolger hat) mit  $\geq 2^k$  Blättern gibt es mindestens einen Pfad der Länge  $\geq k$  von einem Blatt zur Wurzel.

(ohne Beweis)

BEWEIS (von Satz 5.1): Sei  $\mathcal{G} = (\Sigma, N, P, S)$  in CNF mit  $L(\mathcal{G}) = L$ . Wähle  $n = 2^{|N|}$ .

Betrachte den Ableitungsbaum  $\mathcal{T} \in \text{Abl}(S)$  für ein Wort  $z$  mit  $|z| \geq n$ .  $\mathcal{T}$  ist ein Binärbaum mit  $|z| \geq n = 2^{|N|}$  Blättern. In einem solchen Binärbaum existiert nach Lemma 5.2 ein Pfad  $\zeta$  der Länge  $\geq |N|$ . Auf  $\zeta$  liegen  $\geq |N| + 1$  Nichtterminalsymbole. Es muss also mindestens ein Nichtterminalsymbol  $A$  mehrmals vorkommen.

Folge  $\zeta$  vom Blatt in Richtung Wurzel, bis sich ein Nichtterminalsymbol  $A$  das erste Mal wiederholt. Das geschieht nach  $\leq |N|$  Schritten. Teile  $z = uvwxy$  wie in Abb. 5a skizziert. Es gilt:

- $|vx| \geq 1$ , da  $\zeta$  entweder durch  $B$  oder durch  $C$  läuft. Angenommen,  $\zeta$  verläuft durch  $B$ . Somit muss  $C \vdash^* x$  gelten. In CNF ist  $C \vdash^* \varepsilon$  nicht möglich und somit ist  $|x| \geq 1$ . Der Fall, dass  $\zeta$  durch  $C$  verläuft, ist analog.
- $|vwx| \leq 2^{|N|} = n$ , da das obere  $A$  höchstens  $|N|$  Schritte von der Blattebene entfernt und  $\mathcal{T}$  ein Binärbaum ist.
- Für die Aussage  $\forall i \in \mathbb{N} : uv^iwx^iy \in L$  unterscheiden wir drei Fälle:
  - $i = 0$ : Aus dem „unteren“  $A$  haben wir  $w$  abgeleitet, aus dem oberen  $A$  haben wir  $vwx$  abgeleitet. Wir könnten also auch aus dem oberen  $A$  direkt  $w$  ableiten und erhalten  $S \vdash^* uwy$ . (Siehe Abb. 5b).
  - $i = 1$ : Gilt trivialerweise, da  $uv^1wx^1y = z$ .
  - $i \geq 1$ : Aus dem „oberen“  $A$  haben wir  $vAx$  abgeleitet, aus dem unteren  $A$  haben wir  $w$  abgeleitet. Wir könnten also auch aus dem unteren  $A$  nochmal  $vAx$  ableiten. Wenn wir dies oft genug wiederholen, können wir jedes  $uv^iwx^iy$  mit  $i \geq 2$  erzeugen (siehe auch Abb. 5c).

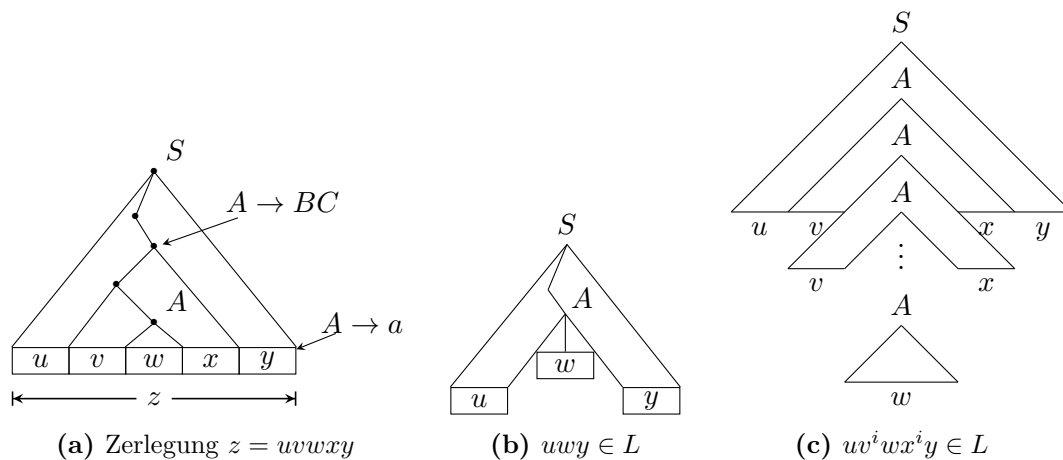


Abb. 5: Erklärung zu Satz 5.1

□

Vorlesung:  
15.12.2017

**Lemma 5.3:** Die Sprache  $L = \{a^n b^n c^n \mid n \geq 1\}$  ist nicht kontextfrei.

BEWEIS: Wir nehmen an,  $L$  wäre kontextfrei. Sei  $n$  die Konstante aus dem PL. Wir betrachten  $z = a^n b^n c^n$ . Es gilt also  $|z| = 3n \geq n$ . Dann gibt es eine Zerlegung  $z = uvwxy$  mit  $|vx| \geq 1$  und  $|vwx| \leq n$ . Wir zeigen nun, dass  $\forall i \in \mathbb{N} : uv^iwx^iy \in L$  nicht gilt (und folgern, dass unsere Annahme falsch war).

Durch  $|vwx| \leq n$  ergeben sich für  $vwx$  folgende Möglichkeiten:

- $vwx = a^j$ : Für  $i = 0$  ist  $v^0wx^0 = w$  mit  $|w| < j$ . Die Anzahl der  $a$  stimmt also nicht mehr mit der Anzahl der  $b$  überein.
- $vwx = a^k b^j$ :
  - Falls  $v = a^{k'}$ ,  $x = b^{j'}$ : Für  $i = 0$  stimmt die Anzahl der  $a$  oder die Anzahl der  $b$  nicht mehr mit der Anzahl der  $c$  überein.
  - Falls  $v$  ein Gemisch aus  $a$  und  $b$  enthält: Für  $i \geq 2$  würde eine Folge  $a^n$  durch eine  $b$ -Folge unterbrochen.
  - Falls  $x$  ein Gemisch aus  $a$  und  $b$  enthält: Für  $i \geq 2$  würde eine Folge  $a^n$  durch eine  $b$ -Folge unterbrochen.
- $vwx = b^j$ : analog zu Fall  $a^j$
- $vwx = b^k c^j$ : analog zu Fall  $a^k b^j$
- $vwx = c^j$ : analog zu Fall  $a^j$

In jeder der Möglichkeiten lässt sich also durch „Pumpen“ ein Wort  $w \notin L$  finden. Daher kann  $L$  nicht kontextfrei sein. □

**Lemma 5.4:** Die Menge der kontextfreien Sprachen ( $\mathcal{M}_2$ ) ist eine echte Teilmenge der Menge der kontextsensitiven Sprachen ( $\mathcal{M}_1$ ).

BEWEIS: Aus Beobachtung 3.1 wissen wir bereits, dass  $\mathcal{M}_2 \subseteq \mathcal{M}_1$  gilt. Aus Bsp. 3.3 wissen wir, dass  $L = \{a^n b^n c^n \mid n \geq 1\}$  von einer kontextsensitiven Grammatik erzeugt wird. Mit Hilfe von Lemma 5.3 folgern wir  $\mathcal{M}_2 \subsetneq \mathcal{M}_1$ .  $\square$

## 5.2 Abschlusseigenschaften für kontextfreie Sprachen

**Satz 5.5:** Die Menge der kontextfreien Sprachen (CFL) ist abgeschlossen unter Vereinigung ( $\cup$ ), Konkatenation ( $\cdot$ ) und dem Sternoperator ( $*$ ).

BEWEIS: Für zwei CFG  $\mathcal{G}_i = (\Sigma, N_i, P_i, S_i), i = 1, 2$ , konstruiere  $\mathcal{G} = (\Sigma, N, P, S)$  mit

$$\begin{aligned} \text{„}\cup\text{“} : N &= N_1 \dot{\cup} N_2 \dot{\cup} \{S\} \\ P &= \{S \rightarrow S_1, S \rightarrow S_2\} \cup P_1 \cup P_2 \\ \text{„}\cdot\text{“} : N &= N_1 \dot{\cup} N_2 \dot{\cup} \{S\} \\ P &= \{S \rightarrow S_1 S_2\} \cup P_1 \cup P_2 \\ \text{„}\ast\text{“} : N &= N_1 \dot{\cup} \{S\} \\ P &= \{S \rightarrow \varepsilon, S \rightarrow S_1 S\} \dot{\cup} P_1 \end{aligned} \quad \square$$

**Satz 5.6:** Die CFL sind *nicht* unter Schnitt ( $\cap$ ) abgeschlossen.

BEWEIS: für  $n \geq 1$

$$\underbrace{\{a^n b^n c^m \mid n, m \geq 1\}}_{\in \text{CFL}} \cap \underbrace{\{a^m b^n c^n \mid m, n \geq 1\}}_{\in \text{CFL}} = \underbrace{\{a^n b^n c^n\}}_{\notin \text{CFL}}$$

Also CFL nicht abgeschlossen unter  $\cap$ . (siehe Lemma 5.3)  $\square$

**Satz 5.7:** Die CFL sind *nicht* unter Komplement ( $\overline{\phantom{x}}$ ) abgeschlossen.

BEWEIS: Angenommen, CFL wäre abgeschlossen unter  $\overline{\phantom{x}}$ .

Falls  $L_1, L_2 \in \text{CFL}$ , dann ist  $\overline{L_1}, \overline{L_2} \in \text{CFL}$  nach Annahme.

$\Rightarrow \overline{L_1} \cup \overline{L_2} \in \text{CFL}$  wegen Abschluss unter „ $\cup$ “.

$\Rightarrow \overline{L_1} \cup \overline{L_2} = L_1 \cap L_2 \in \text{CFL}$  – Widerspruch zu Nicht-Abschluss unter „ $\cap$ “.  $\square$

## 5.3 Deterministisch kontextfreie Sprachen

**Def. 5.1:** Wir definieren einen *Kellerautomaten mit akzeptierenden Zuständen* als 7-Tupel

$$\mathcal{E} = (\Sigma, Q, \Gamma, q^{\text{init}}, Z^{\text{init}}, \delta, F)$$



mit den Komponenten  $\Sigma, Q, \Gamma, q^{\text{init}}, Z^{\text{init}}, \delta$  wie für einen gewöhnlichen Kellerautomaten und  $F \subseteq Q$  als die Menge der akzeptierenden Zustände.

Weiterhin definieren wir das Akzeptanzverhalten folgendermaßen:

Eine Konfiguration  $(q, w, \gamma)$  eines Kellerautomaten mit akzeptierenden Zuständen ist *akzeptierend*, falls  $q \in F$  und  $w = \varepsilon$ . Die von einem Kellerautomaten mit akzeptierenden Zuständen  $\mathcal{E}$  *akzeptierte Sprache* ist

$$L(\mathcal{E}) = \{w \in \Sigma^* \mid \exists q \in F, \gamma \in \Gamma^* : (q^{\text{init}}, w, Z^{\text{init}}) \triangleright^* (q, \varepsilon, \gamma)\}.$$

Hierbei ist  $\triangleright$  die Schrittrelation genau wie für gewöhnliche Kellerautomaten.  $\diamond$

**Lemma 5.8:** Zu jedem (gewöhnlichen) Kellerautomaten gibt es einen Kellerautomaten mit akzeptierenden Zuständen, der die gleiche Sprache akzeptiert. Gleiches gilt in die andere Richtung.

BEWEIS: Siehe Übungsblatt 7 Aufgabe 5.  $\square$

**Def. 5.2:** Ein deterministischer Kellerautomat (DPDA) ist ein 7-Tupel

$$\mathcal{D} = (\underbrace{\Sigma, Q, \Gamma, q^{\text{init}}, Z^{\text{init}}}_{\text{wie gehabt}}, \delta, F)$$

- $F \subseteq Q$  akzeptierende Zustände
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma^*)$  wobei für alle  $q \in Q, a \in \Sigma, Z \in \Gamma$  gelten muss, dass  $|\delta(q, a, Z)| + |\delta(q, \varepsilon, Z)| \leq 1$
- Die Schrittrelation „ $\triangleright$ “ ist definiert wie bei NPDAs.
- $L(\mathcal{D}) = \{w \in \Sigma^* \mid (q^{\text{init}}, w, Z^{\text{init}}) \triangleright^* (q', \varepsilon, \gamma) \wedge q' \in F\}$   $\diamond$

**Lemma 5.9:** Zu jedem DPDA  $\mathcal{D}$  gibt es einen äquivalenten DPDA  $\mathcal{D}_e$ , der jede Eingabe bis zum Ende liest.

BEWEIS: Sei  $\mathcal{D} = (\Sigma, Q, \Gamma, q^{\text{init}}, Z^{\text{init}}, \delta, F)$ . Es gibt zwei Möglichkeiten, warum  $\mathcal{D}$  nicht die gesamte Eingabe verarbeitet: Die Transitionsrelation ist nicht total oder der Automat bleibt bei leerem Keller stecken.

Abhilfe: Führe einen Senkzustand ein, auf dem  $\mathcal{D}_S$  weiterrechnet, wenn  $\mathcal{D}$  nicht mehr weiterrechnen kann. Definiere dazu  $\mathcal{D}_S = (\Sigma, Q_S, \Gamma_S, q_S^{\text{init}}, Z_S^{\text{init}}, \delta_S, F_S)$  mit

$Q_S = Q \cup \{q_S^{\text{init}}, q_s\}$  mit neuen Zuständen:  $q_S^{\text{init}}$  neuer Startzustand und  $q_s$  Senkzustand  
 $F_S = F$

$\Gamma_S = \Gamma \cup \{Z_S^{\text{init}}\}$  neues Kellerbodensymbol

und Transitionsfunktion  $\delta_S$  gegeben durch

$$\delta_S(q_S^{\text{init}}, \varepsilon, Z_S^{\text{init}}) = \{(q^{\text{init}}, Z^{\text{init}} Z_S^{\text{init}})\}$$

- Transitionsfunktion totalisieren: Für alle  $q \in Q$ ,  $Z \in \Gamma$

$$\delta_S(q, \varepsilon, Z) = \begin{cases} \delta(q, \varepsilon, Z) & \text{falls } \delta(q, \varepsilon, Z) \neq \emptyset \vee \exists a \in \Sigma, \delta(q, a, Z) \neq \emptyset \\ \{(q_s, Z)\} & \text{sonst} \end{cases}$$

$$\delta_S(q, a, Z) = \begin{cases} \delta(q, a, Z) & \text{falls } \delta(q, \varepsilon, Z) \neq \emptyset \vee \delta(q, a, Z) \neq \emptyset \\ \{(q_s, Z)\} & \text{sonst} \end{cases}$$

Intuition: Wenn  $\delta(q, a, Z) = \emptyset$  und es keinen  $\epsilon$ -Übergang gibt, kann das Wort von  $\mathcal{D}$  nicht weiter abgearbeitet werden.  $\mathcal{D}_S$  geht nun in Senkzustand  $q_s$  und arbeitet dort weiter. Analog für  $\delta(q, \epsilon, Z)$ .

- Kellerunterlauf vermeiden:  $\mathcal{D}$  hat  $Z^{\text{init}}$  abgeräumt, so dass  $Z_S^{\text{init}}$  sichtbar geworden ist. Füge eine  $\epsilon$ -Transition in Senkzustand  $q_s$  hinzu, indem man für alle  $q \in Q$  definiert:

$$\delta_S(q, \varepsilon, Z_S^{\text{init}}) = \{(q_s, Z_S^{\text{init}})\}$$

In  $q_s$  kann  $\mathcal{D}_S$  nun das Restwort für alle  $a \in \Sigma$ ,  $Z \in \Gamma_S$  abarbeiten:

$$\delta_S(q_s, a, Z) = \{(q_s, Z)\}$$

Bemerkung:  $q_s \notin F_S$ , da ein Kellerunterlauf das Akzeptieren des eingelesenen Worts ausschließt.

Es verbleibt zu zeigen:

- $\forall w \in \Sigma^* \exists q \in Q_S, \gamma \in \Gamma_S^* : (q_S^{\text{init}}, w, Z_S^{\text{init}}) \triangleright^* (q, \varepsilon, \gamma)$ .  
(Induktion über die Länge von  $w$ )
- $L(\mathcal{D}) = L(\mathcal{D}_S)$ . □

**Satz 5.10:** Die deterministischen CFL sind unter Komplement abgeschlossen.

BEWEIS: Sei  $L$  deterministisch kontextfrei. Nach Lemma 5.9 gibt es einen DPDA  $\mathcal{D}$  mit  $L(\mathcal{D}) = L$ , der die komplette Eingabe liest.

Konstruiere einen DPDA  $\mathcal{D}_K$ , sodass  $L(\mathcal{D}_K) = \bar{L}$ . Definiere dazu  $Q_K = Q \times \{0, 1, 2\}$ . Bedeutung des zusätzlichen „Flags“ in  $\{0, 1, 2\}$ :

- 0: seit dem Lesen des letzten Symbols  $a \in \Sigma$  wurde kein akzeptierender Zustand durchlaufen
- 1: seit dem Lesen des letzten Symbols  $a \in \Sigma$  wurde mindestens ein akzeptierender Zustand durchlaufen
- 2: markiert einen akzeptierenden Zustand in  $\mathcal{D}_K$

$$F_K = F \times \{2\}$$

Definiere Hilfsfunktion  $acc : Q \rightarrow \{0, 1\}$  durch

$$acc(q) = \begin{cases} 0 & q \notin F \\ 1 & q \in F \end{cases}$$

$$q_K^{\text{init}} = (q^{\text{init}}, acc(q^{\text{init}}))$$

Wir definieren  $\delta'$  für alle  $q \in Q$ ,  $Z \in \Gamma$  folgendermaßen.

Falls  $\delta(q, \varepsilon, Z) = \{(q', \gamma)\}$ , dann

$$\begin{aligned} \delta'((q, 0), \varepsilon, Z) &= ((q', acc(q')), \gamma) \\ \delta'((q, 1), \varepsilon, Z) &= ((q', 1), \gamma) \end{aligned}$$

Falls  $\delta(q, a, Z) = \{(q', \gamma)\}$ , dann

$$\begin{aligned} \delta'((q, 0), \varepsilon, Z) &= \{((q, 2), Z)\} \\ \delta'((q, 2), a, Z) &= ((q', acc(q')), \gamma) \\ \delta'((q, 1), a, Z) &= ((q', acc(q')), \gamma) \end{aligned}$$

□

**Satz 5.11:** Die deterministisch kontextfreien Sprachen sind eine echte Teilmenge der kontextfreien Sprachen.

BEWEIS: Offensichtlich gibt es eine Teilmengenbeziehung, denn die von nichtdeterministischen Kellerautomaten mit akzeptierendem Zustand akzeptierten Sprachen sind gerade die kontextfreien Sprachen und jeder deterministische Kellerautomat ist ein spezieller nichtdeterministischer Kellerautomat mit akzeptierenden Zuständen. Die deterministisch kontextfreien Sprachen sind unter Komplement abgeschlossen, die kontextfreien Sprachen sind es nicht. Die beiden Mengen von Sprachen können also nicht gleich sein. □

## 6 Turingmaschinen

### 6.1 Turingmaschine (informell)

TODO (teilweise siehe Folien)

### 6.2 Formalisierung der TM

**Def. 6.1:** Eine TM ist ein 7-Tupel

$$\mathcal{M} = (\Sigma, Q, \Gamma, \delta, q^{\text{init}}, \sqcup, F)$$

Dabei ist

- $\Sigma$  ist Alphabet,
- $Q$  ist endliche Menge deren Elemente wir Zustände nennen,
- $\Gamma \supsetneq \Sigma$  ist Menge die wir *Bandalphabet* nennen,
- $\delta : Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{R, L, N\})$  eine Funktion, die wir *Transitionsfunktion* nennen,
- $q^{\text{init}} \in Q$  ein Zustand, den wir *Startzustand* nennen,
- $\sqcup \in \Gamma \setminus \Sigma$  ein Zeichen dass wir *Blank* nennen und
- $F \subseteq Q$  eine Teilmenge der Zustände, deren Elemente wir *akzeptierende Zustände* nennen.

Die TM  $\mathcal{M}$  heisst *deterministisch* (DTM), falls  $\forall q \in Q, \forall a \in \Gamma, |\delta(q, a)| \leq 1$ .  
Ansonsten ist  $\mathcal{M}$  *nichtdeterministisch* (NTM). ◇

Im Folgenden sei  $\mathcal{M} = (\Sigma, Q, \Gamma, \delta, q^{\text{init}}, \sqcup, F)$  eine TM.

Die Konfiguration einer Turingmaschine wird durch drei Dinge beschrieben:

1. Der aktuelle Zustand
2. Der Bandinhalt
3. Die Position des Kopfes



Analog zu den vorigen Kapiteln schreiben wir  $\vdash^* \subseteq \text{Konf}(\mathcal{M}) \times \text{Konf}(\mathcal{M})$  für die reflexive transitive Hülle von  $\vdash$ .

- Das Symbol  $\vdash$  steht also für eine Relation auf  $\text{Konf}(\mathcal{M})$  die einzelne Berechnungsschritte beschreibt und
- der Ausdruck  $\vdash^*$  steht für eine Relation auf  $\text{Konf}(\mathcal{M})$  die eine endliche Sequenz von Berechnungsschritten beschreibt.

**Def. 6.4** (Die von TM  $\mathcal{M}$  akzeptierte Sprache):

$$L(\mathcal{M}) = \{w \in \Sigma^* \mid q^{\text{init}}w \vdash^* uqv \text{ Haltekonfiguration} \\ q \in F\}$$

◇

**Def. 6.5:** Eine deterministische TM (DTM)  $\mathcal{M}$  *terminiert* auf Eingabe  $w$  falls es eine Haltekonfiguration  $uq'v$  gibt sodass  $q^{\text{init}}w \vdash^* uq'v$  gilt. ◇

Falls eine DTM  $\mathcal{M}$  ein Wort  $w$  nicht akzeptiert kann dies zwei Ursachen haben:

1.  $\mathcal{M}$  terminiert nicht auf  $w$ .
2. Der Zustand der Haltekonfiguration von  $\mathcal{M}$  auf  $w$  ist nicht akzeptierend.

**Def. 6.6** (Die von einer DTM  $\mathcal{M}$  berechnete Funktion): Sei  $\mathcal{M}$  eine DTM dann ist die von  $\mathcal{M}$  *berechnete partielle* <sup>26</sup> *Funktion*  $f_{\mathcal{M}} : \Sigma^* \rightarrow \Sigma^*$  wie folgt definiert.

$$f_{\mathcal{M}}(w) = \begin{cases} v & \text{falls } \exists(u, q, v') \in \text{Konf}(\mathcal{M}) \text{ sodass } q^{\text{init}}w \vdash^* uqv' \\ & \text{und } uqv' \text{ ist Haltekonfiguration} \\ & \text{und } v = \text{out}(v') \\ \text{undef.} & \text{sonst} \end{cases}$$

Dabei ist  $\text{out}$  die Funktion, die ein Wort auf das Eingabealphabet  $\Sigma$  projiziert. Wie definieren  $\text{out} : \Gamma^* \rightarrow \Sigma^*$  formal wie folgt.

$$\begin{aligned} \text{out}(\varepsilon) &= \varepsilon \\ \text{out}(au) &= a \cdot \text{out}(u) \quad a \in \Sigma \\ \text{out}(bu) &= \text{out}(u) \quad b \in \Gamma \setminus \Sigma \end{aligned}$$

*Beobachtung:* Die Funktion  $f_{\mathcal{M}}(w)$  ist genau für die Wörter definiert, für die  $\mathcal{M}$  angesetzt auf  $w$  terminiert. ◇

<sup>26</sup>Im Gegensatz zu einer Funktion muss eine partielle Funktion nicht für alle Elemente des Urbildes definiert sein. Ist  $f$  keine Funktion, sondern nur eine partielle Funktion von  $X$  nach  $Y$  so schreiben wir  $f : X \rightarrow Y$  statt  $f : X \rightarrow Y$ .

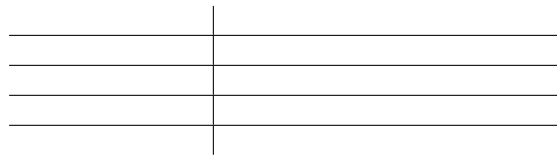
### 6.3 Techniken zur TM Programmierung

- Endlicher Speicher

Zum Abspeichern eines Elements aus endl. Menge  $A$  verwende

$$Q' = Q \times A$$

- Mehrspurmachines



**Abb. 7:** Mehrspurmachine

Eine  $k$ -Spur TM kann gleichzeitig  $k \geq 1$  Symbole  $\leftarrow \Gamma$  unter dem Kopf lesen.

Kann durch Standard TM simuliert werden:

$$\Gamma' = \Sigma \dot{\cup} \Gamma^k \text{ mit } \sqcup' = \sqcup^k$$

... vereinfacht die Programmierung

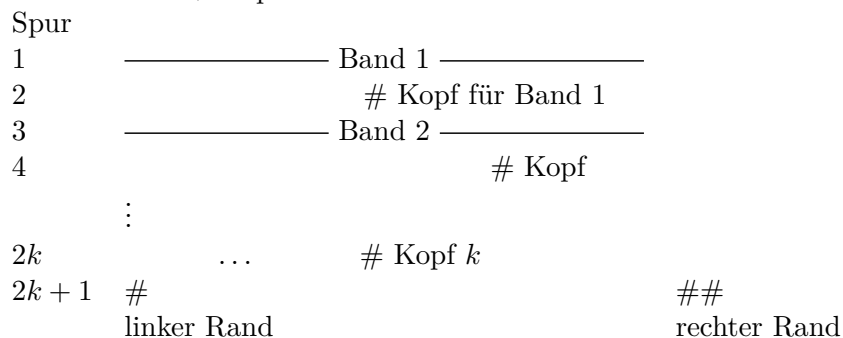
**Bsp.:** Schulalg. für binäre Addition, Multiplikation

- *Mehrbandmaschinen*

Eine  $k$ -Band TM besitzt  $k \geq 1$  Bänder und  $k$  Köpfe, die bei jedem Schritt lesen, schreiben und sich unabhängig voneinander bewegen.

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

- keine herkömmliche TM (für  $k > 1$ )
- kann durch  $2k + 1$  Spur TM simuliert werden:



**Satz 6.1:** Eine  $k$ -Band TM kann durch eine 1-Band TM simuliert werden.  $M = (Q \dots)$

BEWEIS: Zeige: ein Schritt der  $k$ -Band TM wird durch endlich viele Schritte auf einer 1-Band TM simuliert.

1. Schritt: Kodierung der Konfiguration der  $k$ -Band TM

Definiere  $M'$  als TM mit  $2k + 1$  Spuren und  $\Gamma' = \Gamma \cup \{\#\}$

- Die Spuren  $1, 3, \dots, 2k - 1$  enthalten das entspr. Band von  $M$ : Band  $i \leftrightarrow$  Spur  $2i - 1$
- Die Spuren  $2, 4, \dots, 2k$  sind leer bis auf eine Marke  $\#$ , die auf Spur  $2i$  die Position des Kopfes auf Band  $i$  markiert
- Spur  $2k + 1$  enthält  
 $\#$  Marke für linken Rand  
 $\#\#$  Marke für rechten Rand  
 Zwischen den beiden Marken befindet sich der bearbeitete Bereich des Bands.  
 D.h. die TM arbeitet zwischen der linken und rechten Marke und schiebt die Marken bei Bedarf weiter.

2. Schritt: Herstellen der Start-Konfiguration.

Annahme: Eingabe für  $M$  auf Band 1

Jetzt Eingabe (für  $M'$ )  $w = a_1 \dots a_n$

- a) Kopiere  $w$  auf Spur 1
- b) Kopf setzen auf Spur  $2, \dots, 2k$  an die Position des ersten Symbols von  $w$
- c) auf Spur  $2k + 1$ :  $\# \sqcup \#\#$

$2k + 1$	$\# \sqcup \#\#$
$2k$	$\#$
$2k - 1$	$\sqcup$
$\vdots$	
$4$	$\#$
$3$	$\sqcup$
$2$	$\#$
Spur 1	$a_1 a_2 \dots a_n$

Springe nach  $\text{Sim}(q^{\text{init}})$ , der Zustand in  $M'$ , an dem die Simulation des Zustands  $q$  aus  $M$  beginnt.

3. Simulation eines Rechnerschritts im Zustand  $\text{Sim}(q)$ :

Kopf auf linker Begrenzung, d.h. linker  $\#$  auf Spur  $2k + 1$

- Durchlauf bis rechter Rand, sammle dabei Symbole unter den Köpfen, speichern in endl. Zustand  $\vec{\gamma} \in \Gamma^k$



- Berechne  $\delta(q, \vec{\gamma}) = (q', \vec{\gamma}', \vec{d})$   
neuer Zustand, für jeden Kopf ein neues Symbol  $\vec{\gamma}'$  und Richtung  $\vec{d}$ .
- Rücklauf nach links, dabei Schreiben um  $\vec{\gamma}'$  und Versetzen der Köpfe gemäß  $\vec{d}$ .

Falls eine Kopfbewegung den Rand auf Spur  $2k + 1$  überschreitet, dann verschiebe Randmarke entsprechend.

Beim Rücklauf: Test auf Haltekonfiguration der  $k$ -Band TM.

Falls ja, dann Sprung in Haltekonf. von  $M'$

Weiter im Zustand  $\text{Sim}(q')$ .

□

**Korollar:** Beim Erkunden eines Worts der Länge  $n$  benötige die  $k$ -Band Maschine  $M$   $T(n)$  Schritte und  $S(n)$  Zellen auf den Bändern.

- $M'$  benötigt  $O(S(n))$  Zellen
- $M'$  benötigt  $O(S(n \cdot T(n)))$  Schritte  $= O(T(n)^2)$

Weitere TM-Booster

- Unbeschränkt großer Speicher  
→ für jede „Variable“ ein neues Band
- Datenstrukturen  
↳ entsprechend kodieren.

◇

## 6.4 Das Gesetz von Church-Turing (Churchsche These)

**Satz 6.2:** Jede intuitiv berechenbare Funktion ist mit TM (in formalem Sinn) berechenbar.

„Intuitiv berechenbar“  $\equiv$  man kann Algorithmus hinschreiben

- endliche Beschreibung
- jeder Schritt effektiv durchführbar
- klare Vorschrift

Status wie Naturgesetz – nicht beweisbar

→ allgemein anerkannt

→ weitere Versuche Berechenbarkeit zu formulieren, äquivalent zu TMen erwiesen.

## Liste der Definitionen

1.1	Def. (Alphabet $\Sigma$ ) . . . . .	4
1.2	Def. (Wort $w$ über $\Sigma$ ) . . . . .	4
1.3	Def. (Konkatenation von Wörtern) . . . . .	5
1.4	Def. . . . .	5
1.5	Def. (Sprache über $\Sigma$ ) . . . . .	5
1.6	Def. (Konkatenation und Potenzierung von Sprachen) . . . . .	6
1.7	Def. (Kleene-Abschluss, Kleene-Stern) . . . . .	6
2.1	Def. (DEA) . . . . .	9
2.2	Def. (Induktive Erweiterung von $\delta$ auf Wörter) . . . . .	9
2.3	Def. (Die durch einen DEA akzeptierte Sprache) . . . . .	10
2.4	Def. . . . .	12
2.5	Def. (Äquivalenz von DEA-Zuständen) . . . . .	12
2.6	Def. . . . .	14
2.7	Def. (Äquivalenzklassenautomat) . . . . .	15
2.8	Def. (Rechtskongruente Äquivalenzrelation) . . . . .	16
2.9	Def. . . . .	17
2.10	Def. (NEA) . . . . .	23
2.11	Def. (Lauf eines Automaten) . . . . .	23
2.12	Def. (NEA zu DEA) . . . . .	24
2.13	Def. (Potenzmengenautomat) . . . . .	24
2.14	Def. ( $\varepsilon$ -NEA) . . . . .	27
2.15	Def. . . . .	28
2.16	Def. . . . .	29
2.17	Def. . . . .	29
2.18	Def. ( $\varepsilon$ -freier Automat) . . . . .	29
2.19	Def. . . . .	30
2.20	Def. (Abgeschlossenheit von $\mathcal{L}$ ) . . . . .	31
2.21	Def. ( $\text{RE}(\Sigma)$ ) . . . . .	33
2.22	Def. (Semantik eines regulären Ausdrucks) . . . . .	34
2.23	Def. . . . .	35
3.1	Def. . . . .	40
3.2	Def. (Ableitungsrelation, Ableitung, Sprache einer Grammatik) . . . . .	40
3.3	Def. (Chomsky-Hierarchie) . . . . .	42
3.4	Def. (Ableitungsbaum) . . . . .	43
3.5	Def. (Eindeutigkeit von CFG und CFL) . . . . .	44
3.6	Def. . . . .	45
3.7	Def. . . . .	46
3.8	Def. . . . .	48

3.9	Def. (CFG in CNF) . . . . .	49
4.1	Def. (NPDA) . . . . .	55
4.2	Def. (Menge der Konfigurationen eines NPDA) . . . . .	55
5.1	Def. . . . .	64
5.2	Def. (DPDA) . . . . .	65
6.1	Def. (TM) . . . . .	68
6.2	Def. (Konfiguration einer TM) . . . . .	69
6.3	Def. (Rechenschrittrelation) . . . . .	69
6.4	Def. (Die von TM $\mathcal{M}$ akzeptierte Sprache) . . . . .	70
6.5	Def. (Terminierung) . . . . .	70
6.6	Def. (Die von einer DTM $\mathcal{M}$ berechnete Funktion) . . . . .	70

## Liste der Sätze

2.1	Satz . . . . .	10
2.2	Lemma ( $\equiv$ ist Äquivalenzrelation) . . . . .	14
2.3	Lemma . . . . .	15
2.4	Satz (Äquivalenzklassenautomat ist wohldefiniert) . . . . .	15
2.5	Satz (Myhill und Nerode) . . . . .	19
2.5	Korollar . . . . .	20
2.6	Lemma (Pumping Lemma) . . . . .	21
2.7	Satz (Rabin und Scott) . . . . .	24
2.8	Satz . . . . .	29
2.9	Lemma . . . . .	31
2.10	Lemma . . . . .	32
2.11	Lemma . . . . .	32
2.12	Satz (Kleene) . . . . .	35
2.13	Lemma (Ardens Lemma) . . . . .	36
2.14	Korollar . . . . .	37
3.1	Lemma . . . . .	44
3.2	Lemma (SEP) . . . . .	45
3.3	Lemma (BIN) . . . . .	46
3.4	Satz . . . . .	46
3.5	Korollar . . . . .	48
3.6	Lemma (DEL) . . . . .	48
3.7	Lemma (UNIT) . . . . .	48
3.8	Satz . . . . .	49
3.9	Lemma . . . . .	49
3.10	Satz (Wortproblem für CFL entscheidbar) . . . . .	50
3.11	Satz (Entscheidbarkeit des Leerheitsproblems für kontextfreie Sprachen) . . . . .	51

3.12	Satz (Typ-3-Sprache ist regulär) . . . . .	52
3.13	Lemma . . . . .	53
4.1	Lemma . . . . .	56
4.2	Lemma (Mehr Keller – mehr Möglichkeiten.) . . . . .	56
4.3	Lemma . . . . .	58
4.4	Lemma . . . . .	58
4.5	Satz . . . . .	61
5.1	Satz (Pumping Lemma für CFL) . . . . .	62
5.2	Lemma . . . . .	62
5.3	Lemma . . . . .	63
5.4	Lemma . . . . .	64
5.5	Satz . . . . .	64
5.6	Satz . . . . .	64
5.7	Satz . . . . .	64
5.8	Lemma . . . . .	65
5.9	Lemma (DPDA, der gesamte Eingabe verarbeitet) . . . . .	65
5.10	Satz (Abgeschlossenheit der deterministischen CFL) . . . . .	66
5.11	Satz . . . . .	67
6.1	Satz (Simulation von $k$ -Band TM durch 1-Band TM) . . . . .	72
	Korollar . . . . .	73
6.2	Satz (Intuitiv berechenbare Funktionen sind mit TM berechenbar) . . . .	73

## Abbildungsverzeichnis

1	Endliches Band . . . . .	7
2	Schematische Darstellung eines DEA, mit $\tilde{\delta}(q^{\text{init}}, u) = \tilde{\delta}(q^{\text{init}}, v) = q_{42}$ . . .	16
3	Nichtdeterministischer Automat für $L_n$ . . . . .	23
4	DEA „modulo 3“ . . . . .	36
5	Erklärung zu Satz 5.1 . . . . .	63
6	Informelle graphische Darstellung einer Turingmaschinenkonfiguration . .	69
7	Mehrspurmaschine . . . . .	71

## Abkürzungsverzeichnis

<b>AL</b>	Aussagenlogik
<b>CFL</b>	Menge der kontextfreien Sprachen
<b>CFG</b>	kontextfreie Grammatik

---

<b>CNF</b>	Chomsky-Normalform
<b>CP</b>	Korrespondenzproblem
<b>CYK</b>	Cocke, Younger, Kasami
<b>DAG</b>	gerichteter azyklischer Graph
<b>DCFG</b>	deterministische CFG
<b>DCFL</b>	deterministische CFL
<b>DEA</b>	deterministischer endlicher Automat
<b>DFA</b>	engl.: deterministic finite automaton
<b>DPDA</b>	deterministischer Kellerautomat
<b>DTM</b>	deterministische TM
<b>EA</b>	endlicher Automat
<b>LBA</b>	Linear Bounded Automaton
<b>MPCP</b>	Das modifizierte PCP
<b>ND</b>	Nicht-Determinismus
<b>NEA</b>	nichtdeterministischer endlicher Automat
<b>NFA</b>	engl.: nondeterministic finite automaton
<b>NPDA</b>	nichtdeterministischer Kellerautomat
<b>NT</b>	Nichtterminal
<b>NTM</b>	Eine nichtdeterministische TM
<b>PCP</b>	Das Postsche Korrespondenzproblem
<b>PDA</b>	pushdown automaton (Kellerautomat)
<b>PL</b>	Pumping Lemma
<b><i>RE</i></b>	Menge der regulären Ausdrücke
<b><i>REG</i></b>	Menge der regulären Sprachen
<b>RM</b>	Registermaschine
<b>TM</b>	Turing-Maschine
<b>TT</b>	Turingtabelle

**Anmerkungsverzeichnis**

Vorlesung: 18.10.2017 . . . . .	4
Vorlesung: 20.10.17 . . . . .	7
Vorlesung: 25.10.17 . . . . .	11
Vorlesung: 27.10.17 . . . . .	16
Vorlesung: 3.11.16 . . . . .	20
Vorlesung: 8.11.17 . . . . .	24
Vorlesung: 10.11.17 . . . . .	28
Vorlesung: 15.11.17 . . . . .	31
Vorlesung: 17.11.2017 . . . . .	36
Vorlesung: 22.11.17 . . . . .	41
Vorlesung: 24.11.2017 . . . . .	45
Vorlesung: 29.11.2017 . . . . .	49
Vorlesung: 1.12.2017 . . . . .	52
Vorlesung: 6.12.2017 . . . . .	56
Vorlesung: 8.12.2017 . . . . .	58
Vorlesung: 13.12.2017 . . . . .	61
Vorlesung: 15.12.2017 . . . . .	63
Vorlesung: 20.12.17 . . . . .	68
Entwurf für Vorlesung: 10.01.2018 . . . . .	71