1 Vorspann: Sprachen

Vorlesung: 21.04.26

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

Zeichen sind hier beliebige abstrakte Symbole.

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

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

Wir verwenden typischerweise den griechischen Buchstaben Σ als Namen für ein Alphabet und die lateinischen Buchstaben a, b, c als Namen für Zeichen.Im Folgenden sei Σ immer ein beliebiges Alphabet.

Def. 1.2: Ein *Wort* ist eine endliche Folge von Elementen aus Σ. Die leere Folge nennen wir das *leere Wort* geschrieben als ε. Wir bezeichnen die Menge aller Wörter mit Σ^* und die Menge aller nicht leeren Wörter mit $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$.

Hier die induktive Definition.

1. $\varepsilon \in \Sigma^*$.
2. $a \in \Sigma$ und $w \in \Sigma^*$ impliziert $a.w \in \Sigma^*$.

◇

Hier einige Konventionen:

- Wir schreiben ein Wort immer ohne Trennsymbole wie z.B. Komma. Also `einhorn` statt `e,i,n,h,o,r,n`.
- Wir verwenden ε für das leere Wort.
- Wir verwenden typischerweise u, v, w als Namen für Wörter.

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

- `rambo` (Länge 5)
- `eis`, `ies` (beide Länge 3 aber ungleich)
- ε (Länge 0)

Aus der induktiven Definition ergibt sich das folgende Beweisprinzip für Aussagen $\varphi(w)$ über Worte w:

$$\frac{\varphi(\varepsilon) \quad \forall a, w.\varphi(w) \Rightarrow \varphi(a.w)}{\forall w \in \Sigma^*.\varphi(w)}$$

Dieses Prinzip können wir auch verwenden um Funktionen auf Worten zu definieren. Als erstes Beispiel definieren wir die Länge eines Wortes, d.h. die Anzahl der Symbole des Wortes.

Def. 1.3 (Länge eines Worts)**:** Die *Länge* eines Worts, $|\_| : \Sigma^* \to \mathbb{N}$, wird für alle $w \in \Sigma^*$ induktiv definiert durch

1. $|\varepsilon| = 0$;
2. für alle $a \in \Sigma$ und $w \in \Sigma^*$: $|a.w| = 1 + |w|$.

$\diamond$

Wörter lassen sich „verketten"/„aneinanderhängen". Die entsprechende Operation heißt *Konkatenation*, geschrieben „$\cdot$" (wie Multiplikation).

Def. 1.4 (Konkatenation von Wörtern)**:** Die *Konkatenation*, $\cdot : \Sigma^* \times \Sigma^* \to \Sigma^*$, wird für alle $v \in \Sigma^*$ induktiv definiert durch

1. $\varepsilon \cdot v = v$;
2. für alle $a \in \Sigma$ und $u \in \Sigma^*$: $(a.u) \cdot v = a.(u \cdot v)$.

$\diamond$

Lemma 1.1: Eigenschaften der Konkatenation: $(\Sigma^*, \varepsilon, \cdot)$ ist ein Monoid:

1. $\varepsilon \cdot v = v$ (leeres Wort ist linksneutrales Element);
2. $v \cdot \varepsilon = v$ (leeres Wort ist rechtsneutrales Element);
3. $u \cdot (v \cdot w) = (u \cdot v) \cdot w$ (Assoziativität).

Beweis: 1. $\varepsilon \cdot v = v$ (Definition)

2. $v \cdot \varepsilon = v$ Induktion über v:
   - IA: $(v = \varepsilon)\ \varepsilon \cdot \varepsilon = \varepsilon$ (Definition)
   - IS: $(v \rightsquigarrow a.v)\ (a.v) \cdot \varepsilon = a.(v \cdot \varepsilon) \overset{IV}{=} a.v$
3. $u \cdot (v \cdot w) = (u \cdot v) \cdot w$ Induktion über u:
   - IA: $(u = \varepsilon)\ \varepsilon \cdot (v \cdot w) = (v \cdot w) = (\varepsilon \cdot v) \cdot w$ (Definition)
   - IS: $(u \rightsquigarrow a.u)\ (a.u) \cdot (v \cdot w) \overset{Def}{=} a.(u \cdot (v \cdot w)) \overset{IV}{=} a.((u \cdot v) \cdot w) \overset{Def}{=} ((a.u) \cdot v) \cdot w$

□

Bsp.:

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

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

$w_1w_2w_3$ 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 $\texttt{rambo}\varepsilon$ ist *kein* Wort. Diese Zeichenfolge ist lediglich eine Notation für eine Konkatenationsoperation, die ein Wort der Länge 5 (nämlich `rambo`) beschreibt.

Wörter lassen sich außerdem *potenzieren*:

Def. 1.5: Die *Potenzierung* von Wörtern, $\cdot^{\cdot} : \Sigma^* \times \mathbb{N} \to \Sigma^*$, ist induktiv definiert durch

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

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

Def. 1.6: Eine *Sprache* über Σ ist eine Menge $L \subseteq \Sigma^*$. Gleichwertig: $L \in \mathcal{P}(\Sigma^*)$. ◇

Bsp.:

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

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. 1.7 (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.:

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

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

Def. 1.8 (Kleene-Abschluss, Kleene-Stern, Kleene-Plus)**:** Gegeben eine Sprache $U \subseteq \Sigma^*$, definieren wir den *Kleene-Abschluss* U^* wie folgt.

$$U^* = \bigcup_{n \in \mathbb{N}} U^n$$

Wie nennen den Operator $\cdot^* : \mathcal{P}(\Sigma^*) \to \mathcal{P}(\Sigma^*)$, der für eine gegebene Sprache den Kleene-Abschluss liefert, den *Kleene-Stern*. Analog definieren wir den *Kleene-Plus* Operator $\cdot^+ : \mathcal{P}(\Sigma^*) \to \mathcal{P}(\Sigma^*)$ als

$$U^+ = \bigcup_{n \geq 1} U^n.$$

◇

Es gilt also $U^* = U^+ \cup \{\varepsilon\}$.

Zum Abschluss eine alternative Definition, die in manchen Textbüchern oder manchmal in der Literatur verwendet wird. In diesem Skript werden wir sie nicht weiter verwenden.

Def. 1.9 (Alternative Definition für Σ^*)**:** Sei $n \in \mathbb{N}$.

- $\Sigma^n = \{1, \ldots, n\} \to \Sigma$ Worte der Länge n
- insbesondere $\Sigma^0 = \{\} \to \Sigma$ enthält nur ein Element, ε
- $\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i$
- Konkatenation: $u \in \Sigma^n$, $v \in \Sigma^m \Rightarrow u \cdot v \in \Sigma^{n+m}$ definiert durch

$$(u \cdot v)(i) = \begin{cases} u(i) & 1 \leq i \leq n \\ v(i-n) & n < i \leq n+m \end{cases}$$

◇

2 Reguläre Sprachen und endliche Automaten

Vorlesung: 22.04.26

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 Sprache darstellten.

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

Ein weiteres Beispiel ist die folgende Sprache.

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

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. „$1100101 \in L_{\mathsf{prim}}$?“ oder „$1100101 \in L_{\mathsf{even}}$?“

Die obige Darstellung der unendlichen Mengen L_{prim} und L_{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_{prim} und L_{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_{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*)

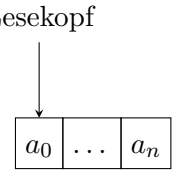


Abb. 1: Endliches Band

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 Zu jedem Zeitpunkt befindet sich der Automat in einem Zustand q aus einer *endlichen* Zustandsmenge Q.

Startzustand $q^{\mathsf{init}} \in Q$

Akzeptierende Zustände $F \subseteq Q$

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

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.“

[1]

Ein passendes Alphabet wäre $\Sigma = \{\texttt{grün}, \texttt{nicht-grün}\}$. 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 δ ist durch die folgende Tabelle gegeben.

[1] 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 `grün`, ansonsten verweile
q_1	q_2	q_1	wechsle nach q_2 falls `grün`, ansonsten verweile
q_2	q_2	q_2	verweile, da es nichts mehr zu retten gibt

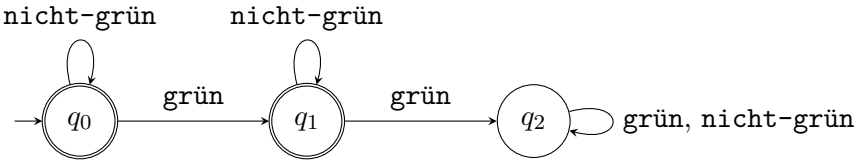
Def. 2.1 (DEA)**:** Ein *deterministischer endlicher Automat (DEA)*[2] ist ein 5-Tupel

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

Dabei ist

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

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 $A_{\texttt{Macaron}}$ aus Beispiel 2.1:



Die folgenden beiden Definitionen erlauben uns mit Hilfe eines DEA eine Sprache zu charakterisieren. Hierbei sei $\mathcal{A} = (\Sigma, Q, \delta, q^{\mathsf{init}}, F)$ ein DEA.

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

1. $\tilde{\delta}(q, \varepsilon) = q$ (Wortende erreicht)

[2] engl. DFA ≙ deterministic finite automaton

2. $\tilde{\delta}(q, aw) = \tilde{\delta}(\delta(q, a), w)$ (Rest im Folgezustand verarbeiten) ◇

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

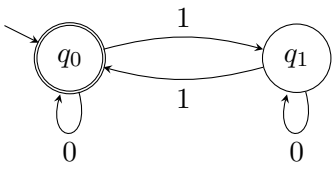
$$L(\mathcal{A}) = \{w \in \Sigma^* \mid \tilde{\delta}(q^{\mathsf{init}}, w) \in F\}.$$

Eine durch einen DEA akzeptierte Sprache heißt *regulär*. ◇

Bsp. 2.2: Ein möglicher DEA für die Sprache

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

aus der Einleitung dieses Kapitels hat die folgende graphische Repräsentation.



Frage: Angenommen wir haben zwei DEAs A_1 und A_2, können wir dann immer einen DEA konstruieren der den Durchschnitt $L(A_1) \cap L(A_2)$ akzeptiert?

Idee: Lasse beide DEAs parallel laufen, akzeptiere nur wenn beide DEAs akzeptieren.

Frage: Lassen sich zwei *parallel laufende* DEAs in einem einzigen DEA implementieren?

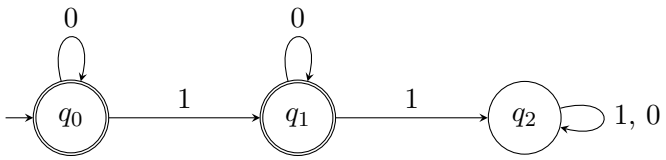
Die nächste Konstruktion und der drauf folgende Satz liefern eine positive Antwort auf diese Frage.

Def. 2.4 (Produktautomat für Durchschnitt)**:** Seien $A_1 = (\Sigma, Q_1, \delta_1, q_1^{\mathsf{init}}, F_1)$ und $A_2 = (\Sigma, Q_2, \delta_2, q_2^{\mathsf{init}}, F_2)$ DEAs. Wir definieren den *Produktautomaten für Durchschnitt* als den DEA $A_\cap = (\Sigma, Q_\cap, \delta_\cap, q_\cap^{\mathsf{init}}, F_\cap)$ mit den folgenden Komponenten.

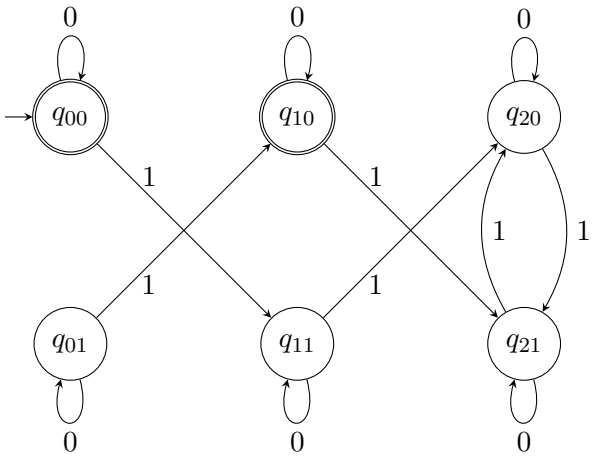
$$\begin{aligned} Q_\cap &= Q_1 \times Q_2 \\ \delta_\cap((q_1, q_2), a) &= (\delta_1(q_1, a), \delta_2(q_2, a)) \,, \quad \text{für alle } a \in \Sigma \\ q_\cap^{\mathsf{init}} &= (q_1^{\mathsf{init}}, q_2^{\mathsf{init}}) \\ F_\cap &= F_1 \times F_2 \end{aligned}$$ ◇

Vorlesung: 28.04.26

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



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



Satz 2.1: Für beliebige DEAs A_1 und A_2 akzeptiert der entsprechende Produktautomat für Durchschnitt die Sprache $L(A_1) \cap L(A_1)$.

Beweis: [3] Seien $A_1 = (\Sigma, Q_1, \delta_1, q_1^{\mathsf{init}}, F_1)$ und $A_2 = (\Sigma, Q_2, \delta_2, q_2^{\mathsf{init}}, F_2)$ DEAs und $A_\cap = (\Sigma, Q_\cap, \delta_\cap, q_\cap^{\mathsf{init}}, F_\cap)$ der zugehörige Produktautomat für den Durchschnitt. Wir zeigen zunächst per Induktion über 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 $\varphi(w)$ gilt.

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

Der Induktionsanfang $\varphi(\varepsilon)$ folgt dabei direkt aus Def. 2.2.

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

[3] Dieser erste Beweis ist außergewöhnlich detailliert. In den folgenden Beweisen werden wir einfache Umformungen zusammenfassen.

Den Induktionsschritt $\varphi(w) \rightsquigarrow \varphi(a.w)$ zeigen wir mit Hilfe der folgenden Umformungen, wobei $a \in \Sigma$ ein beliebiges Zeichen und $w \in \Sigma^*$ ein beliebiges Wort.

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

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

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

□

2.2 Minimierung endlicher Automaten

Beobachtung: Der Zustand q_{01} im Beispiel 2.3 scheint nutzlos. Wir charakterisieren die Nützlichkeit eines Zustands in einem DEA $\mathcal{A} = (Q, \Sigma, \delta, q^{\mathsf{init}}, F)$ formal wie folgt.

Def. 2.5: Ein Zustand $q \in Q$ heißt *erreichbar*, falls ein $w \in \Sigma^*$ existiert, sodass $\tilde{\delta}(q^{\mathsf{init}}, w) = q$. ◇

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

- Fasse $\mathcal{A}$ als Graph auf.
- Wende Tiefensuche ab q^{init} an und markiere dabei alle besuchten Zustände.
- Genau die markierten Zustände sind erreichbar.

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. 2.6: Zwei Zustände $p, q \in Q$ eines DEA sind *äquivalent*, geschrieben $p \equiv q$, falls

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

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:

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

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

Beweis: Zu zeigen ist, dass $\equiv$ reflexiv, transitiv und symmetrisch ist.

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

□

Bsp. 2.5: Für den DEA aus Bsp. 2.3 hat die Relation $\equiv$ drei Äquivalenzklassen.[4]

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

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.

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

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

[4] 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.

Beweis:

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

□

Wir formalisieren das „Verschmelzen“ von Zuständen wie folgt.

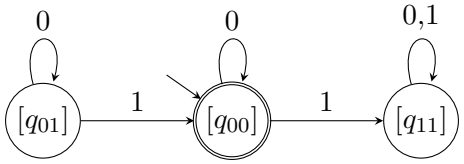
Def. 2.7: Der Äquivalenzklassenautomat $\mathcal{A}_\equiv = (Q_\equiv, \Sigma, \delta_\equiv, q^{\mathsf{init}}{}_\equiv, F_\equiv)$ zu einem DEA $\mathcal{A} = (Q, \Sigma, \delta, q^{\mathsf{init}}, F)$ ist bestimmt durch:

$$
\begin{aligned}
Q_\equiv &= \{[q] \mid q \in Q\} \\
\delta_\equiv(q_\equiv, a) &= [\delta(q, a)] \qquad \text{wobei } q \in q_\equiv \text{ beliebig gewählt (also } q_\equiv = [q]) \\
q^{\mathsf{init}}_\equiv &= [q^{\mathsf{init}}] \\
F_\equiv &= \{[q] \mid q \in F\}
\end{aligned}
$$

◇

Vorlesung: 29.04.26

Bsp. 2.6: 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 per Induktion über w, dass für alle $w \in \Sigma^*$ und alle $q \in Q$ die folgende Äquivalenz gilt.

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

I.A. ($\varphi(\varepsilon)$): $\tilde{\delta}(q,\varepsilon) = q \in F \Leftrightarrow \tilde{\delta}_{\equiv}([q],\varepsilon) = [q] \in F_{\equiv}$
I.S. ($\forall a, w : \varphi(w) \Rightarrow \varphi(aw)$):

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

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

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

□

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 $\mathcal{A}_{\equiv}$ ein DEA ist, der $L(\mathcal{A})$ akzeptiert, sodass kein DEA, der ebenfalls $L(\mathcal{A})$ akzeptiert, weniger Zustände haben kann.

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

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

Motivation: Ist eigentlich jede Sprache regulär? Falls nein, wie können wir nicht-reguläre Sprachen finden? Wie können wir beweisen dass eine Sprache nicht regulär ist? In weiteren Verlauf dieses Unterkapitels wollen wir charakteristische Merkmale von reguläre Sprachen finden. Die Idee dazu basiert auf den folgenden Überlegungen.

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

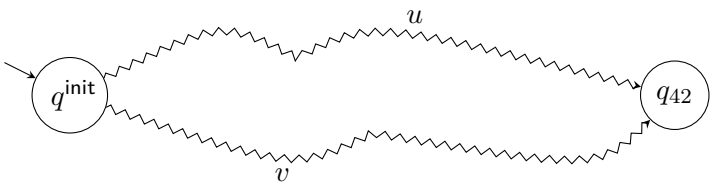


Abb. 2: Schematische Darstellung eines DEA, mit $\tilde{\delta}(q^{\mathsf{init}},u) = \tilde{\delta}(q^{\mathsf{init}},v) = q_{42}$

- Angenommen, es gibt ein Wort $w \in \Sigma^*$, dass die Äquivalenz $uw \in L \Leftrightarrow vw \in L$ verletzt. Dann muss der Entwurf noch einen Fehler enthalten und wir müssen dafür sorgen, dass q^{init} 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, q^{init} 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 solch 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.

Def. 2.9: Für einen DEA $\mathcal{A}$ definiere

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

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^{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.

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

Def. 2.10: Für eine Sprache $L \subseteq \Sigma^*$ ist die *Nerode-Relation* wie folgt definiert.

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

Lemma 2.5: Die Nerode-Relation ist eine Äquivalenzrelation.

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

Lemma 2.6: Die Nerode-Relation ist rechtskongruent.

Dafür starten wir mit folgender Beobachtung.

Lemma 2.7: $\forall u, v \in \Sigma^*. \forall a \in \Sigma.\ (u, v) \in R_L \Rightarrow (ua, va) \in R_L$

Beweis:

$$\begin{aligned} & (u, v) \in R_L \\ \Leftrightarrow\ & \forall w \in \Sigma^* : uw \in L \Leftrightarrow vw \in L \\ & \text{wähle alle } w \text{ der Form } w = aw' \\ \Rightarrow\ & \forall a \in \Sigma : \forall w' \in \Sigma^* : uaw' \in L \Leftrightarrow vaw' \in L \\ \Leftrightarrow\ & (ua, va) \in R_L \end{aligned}$$

D.h. $(u, v) \in R_L \Rightarrow \forall a \in \Sigma : (ua, va) \in R_L$. □

Beweis (Lemma 2.6): Wir wählen die folgende Induktionsbehauptung

$$\forall w \in \Sigma^* : \forall (u, v) \in R_L : (uw, vw) \in R_L$$

und führen eine Induktion über w durch.

I.A. $(w = \varepsilon)$: $(u\varepsilon, v\varepsilon) = (u, v) \in R_L$
I.S. $(w \rightsquigarrow a.w)$: Zeige nun $\forall (u, v) \in R_L : (uaw, vaw) \in R_L$.
Sei $(u, v) \in R_L$, dann ist nach Lemma 2.7 auch $(ua, va) \in R_L$. Daraus liefert die Induktionsbehauptung sofort $(uaw, vaw) \in R_L$.

□

Bsp. 2.7: 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. 2.8: Für ein beliebiges Alphabet Σ gilt:

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

Bsp. 2.9: Sei $\Sigma = \{0, 1\}$. Die Sprache $L_{\text{centered}} = \{0^n 1 0^n \mid n \in \mathbb{N}\}$ hat bezüglich der Nerode-Relation die folgende Menge von Äquivalenzklassen:

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

Bemerkung 1: Die Äquivalenzklasse [11] enthält alle Wörter, die kein Präfix eines Worts aus L_{centered} sind.

Bemerkung 2: Nicht für alle Präfixe von Wörtern in L_{centered} sind die zugehörigen Äquivalenzklassen paarweise verschieden. Es gilt z.B. $[01] = [0010]$.

Bemerkung 3: Für je zwei verschiedene $k \in \mathbb{N}$ sind die Äquivalenzklassen $[0^k 1]$ verschieden. Es gibt also unendlich viele Äquivalenzklassen.

Bsp. 2.10: Die Sprache $L = \{w \in \{a, b\}^* \mid \#_a(w) > \#_b(w)\}$ hat bezüglich der Nerode-Relation die folgende Menge von Äquivalenzklassen:

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

Satz 2.8 (Myhill und Nerode)**:** Für Sprachen $L \subseteq \Sigma^*$ sind folgende Aussagen äquivalent.

1. L wird von einem DEA akzeptiert.
2. L ist Vereinigung 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) ⇒ (2) Sei $\mathcal{A}$ ein DEA. Es gilt

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

Nun sind die Mengen $\{w \mid \tilde{\delta}(q^{\mathsf{init}}, w) = q\}$ für erreichbare $q \in Q$ genau die Äquivalenzklassen der Relation $R_{\mathcal{A}}$ aus Def. 2.9, einer rechtskongruenten Äquivalenzrelation. Der Index von $R_{\mathcal{A}}$ ist die Anzahl der *erreichbaren* Zustände und somit endlich: $\text{Index}(R_{\mathcal{A}}) \leq |Q| < \infty$.

(2) ⇒ (3) Sei R eine 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.[5]

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

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

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

- $Q = \{[w]_{R_L} \mid w \in \Sigma^*\}$ endlich, da $\text{Index}(R_L)$ endlich
- $\delta(q, a) = [wa]$ für ein $w \in q$—wohldefiniert (vgl. Lemma 2.7)
- $q^{\mathsf{init}} = [\varepsilon]$
- $F = \{[w] \mid w \in L\}$

Wir wollen nun $L(\mathcal{A}) = L$ zeigen. Dafür beweisen wir zunächst per Induktion über 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. $(w \rightsquigarrow a.w)$

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

[5] 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.

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

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

□

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

Beweis: Sei $\mathcal{A}' = (Q', \dots)$ ein *beliebiger* DEA mit $L(\mathcal{A}') = L$.

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

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

Aus „3 ⇒ 1“ erhalten wir $\mathcal{A} = (Q, \dots)$ mit $|Q| = \text{Index}(R_L)$.

Wegen $|Q| = \text{Index}(R_L) \leq \text{Index}(R_{\mathcal{A}'}) \leq |Q'|$ hat $\mathcal{A}$ nicht mehr Zustände als $\mathcal{A}'$. □

2.3 Nichtdeterministischer endlicher Automat (NEA)

Vorlesung: 12.05.2026

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

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

Naiver Lösungsversuch:

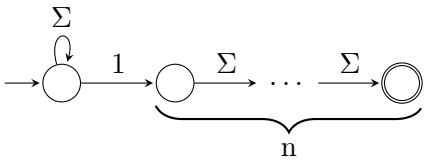


Abb. 3: Nichtdeterministischer Automat für L_n

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 einen neue Art von Automaten, bei dem ein Zustand pro Zeichen mehrere Nachfolger haben darf.

Def. 2.11 (NEA)**:** Ein *nichtdeterministischer endlicher Automat (NEA)*[6] ist ein 5-Tupel

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

Dabei ist

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

Bemerkung: Die Definition des NEA unterscheidet sich vom DEA 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. So 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. 2.12 (Lauf eines Automaten)**:** Die Funktion $\text{run} : Q \times \Sigma^* \to \mathcal{P}(Q^*)$ ist so definiert, dass $\text{run}(q, w)$ die Folgen von Zuständen liefert, die $\mathcal{N}$ vom Zustand q angesetzt auf die Eingabe w durchläuft.

$$\text{run}(q, \varepsilon) = \{q\} \tag{1}$$
$$\text{run}(q, a.w) = \{q.\overline{q} \mid \overline{q} \in \text{run}(q', w), q' \in \delta(q, a)\} \tag{2}$$

Ein Lauf $q_0 \dots q_n \in \text{run}(q_0, w)$ heißt *initial*, falls $q_0 = q^{\mathsf{init}}$. Er heißt *akzeptierend*, falls $q_n \in F$. ◇

Def. 2.13 (Akzeptanz)**:** 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.

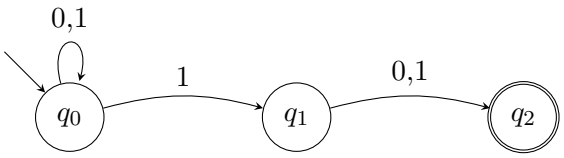
$$L(\mathcal{N}) = \{w \in \Sigma^* \mid \exists \text{ initialer, akzeptierender Lauf von } \mathcal{N} \text{ über } w\}. \quad \diamond$$

Bsp. 2.11: 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.

[6] engl. NFA $\hat{=}$ nondeterministic finite automaton



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 11 hat obiger NEA bereits drei verschiedene initiale Läufe: $q_0q_0q_0$, $q_0q_0q_1$ und $q_0q_1q_2$.

Bemerkung: Die Definitionen von NEA und DEA in der Literatur sind nicht einheitlich. Es gibt äquivalente NEA-Definitionen, die statt der Transitionsfunktion $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ 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 $\mathcal{A} = (\Sigma, Q, \delta, q^{\mathsf{init}}, F)$ gibt es einen NEA, der die gleiche Sprache akzeptiert. Beispiel: der NEA $\mathcal{N} = (\Sigma, Q, \delta_{\mathsf{NEA}}, q^{\mathsf{init}}, F)$ mit $\delta_{\mathsf{NEA}}(q, a) = \{\delta(q, a)\}$, der sich von $\mathcal{A}$ nur in der Transitionsfunktion unterscheidet.

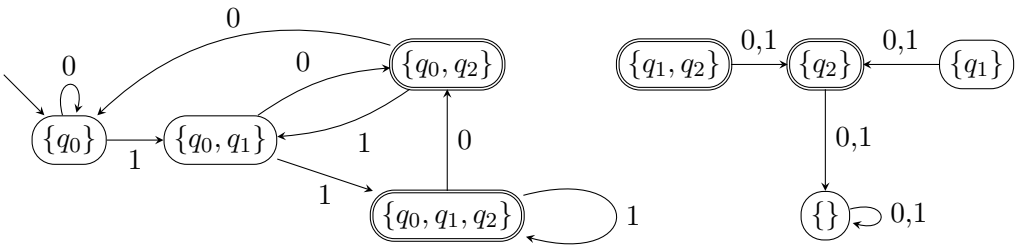
Satz 2.6 (Rabin und Scott)**:** Zu jedem NEA $\mathcal{N}$ mit n Zuständen gibt es einen DEA $\mathcal{A}_{\mathcal{P}}$ mit 2^n Zuständen, sodass $L(\mathcal{A}_{\mathcal{P}}) = L(\mathcal{N})$ gilt.

Zur Vorbereitung des Beweises machen wir zunächst die folgende Definition.

Def. 2.14 (Potenzmengenautomat)**:** Für einen gegebenen NEA $\mathcal{N} = (\Sigma, Q, \delta, q^{\mathsf{init}}, F)$ ist der Potenzmengenautomat $\mathcal{A}_{\mathcal{P}}$ wie folgt definiert.

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

Bsp. 2.12: Der Potenzmengenautomat für den NEA aus Bsp. 2.11 hat das folgende Zustandsdiagramm.



Die vier Zustände auf der rechten Seite sind nicht erreichbar.

Beweis (von Satz 2.6): Zeige $L(\mathcal{A}_\mathcal{P}) = L(\mathcal{N})$. Dafür beweisen wir zunächst per Induktion über w die folgende Eigenschaft:

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

$$q \in \tilde{\delta}_\mathcal{P}(p, w) \Leftrightarrow \exists q_0 \in p. q_0, q_1, \ldots, q_n \in \mathrm{run}(q_0, w), \text{ sodass } q_n = q$$

I.A. ($w = \varepsilon$): $q \in \tilde{\delta}_\mathcal{P}(p, \varepsilon) = p$ wähle $q_0 = q$.
I.S. ($w = aw'$) ein beliebiges Wort.

$$\begin{aligned}
q \in \tilde{\delta}_\mathcal{P}(p, w) &\Leftrightarrow q \in \tilde{\delta}_\mathcal{P}(\delta_\mathcal{P}(p, a), w') \\
&\overset{\text{I.V.}}{\Leftrightarrow} \exists q_1 \in \delta_\mathcal{P}(p, a).\ q_1, q_2, \ldots, q_n \in \mathrm{run}(q_1, w'), \text{ sodass } q_n = q \\
&\Leftrightarrow \exists q_0 \in p.\ q_1 \in \delta_\mathcal{P}(p, a).\ q_1, q_2, \ldots, q_n \in \mathrm{run}(q_1, w'), \text{ sodass } q_n = q \\
&\Leftrightarrow \exists q_0 \in p.\ q_0, q_1, q_2, \ldots, q_n \in \mathrm{run}(q_1, aw'), \text{ sodass } q_{n+1} = q
\end{aligned}$$

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

$$\begin{aligned}
w \in L(\mathcal{A}_\mathcal{P}) &\Leftrightarrow \tilde{\delta}_\mathcal{P}(q_\mathcal{P}^{\mathsf{init}}, w) \in F_\mathcal{P} \\
&\Leftrightarrow \exists p_f \in F_\mathcal{P} : \tilde{\delta}_\mathcal{P}(q_\mathcal{P}^{\mathsf{init}}, w) = p_f \\
&\Leftrightarrow \exists q_f \in F : q_f \in \tilde{\delta}_\mathcal{P}(q_\mathcal{P}^{\mathsf{init}}, w) \\
&\Leftrightarrow \exists q_0 \in q_\mathcal{P}^{\mathsf{init}} = \{q_\mathcal{A}^{\mathsf{init}}\}.\ q_0, q_1, \ldots, q_n \in \mathrm{run}(q_0, w), \text{ sodass } q_n = q_f \in F \\
&\Leftrightarrow \exists \text{ initialer, akzeptierender Lauf von } \mathcal{N} \text{ über } w \\
&\Leftrightarrow w \in L(\mathcal{N})
\end{aligned}$$

□

Bemerkung: Es gelten also die folgenden Äquivalenzen.

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

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

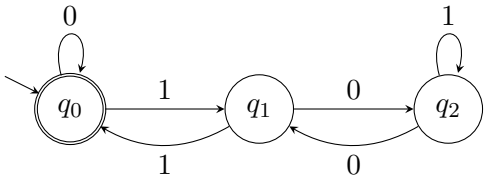
1. Es gibt mit L_n (n-letztes Zeichen) eine Menge von Sprachen, die sich durch einen NEA mit $n+1$ Zuständen darstellen lassen, aber bei denen ein minimaler DEA mindestens 2^n Zustände hat. (Siehe Übungsblatt 3, Aufgabe 2).
2. Zu jedem NEA mit n Zuständen gibt es einen DEA mit 2^n Zuständen, der die gleiche Sprache akzeptiert. (Satz 2.6).
3. Zu jedem DEA mit n Zuständen gibt es einen NEA mit n Zuständen, der die gleiche Sprache akzeptiert.

2.4 Pumping Lemma (PL) für reguläre Sprachen

Vorlesung: 13.05.2026

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

Bsp. 2.13: 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 $1\underline{00}1 \in L$.
- 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 2.7 (Pumping Lemma)**:** Sei L eine reguläre Sprache. Dann gilt:

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

BEWEIS: Sei $\mathcal{A} = (\Sigma, Q, \delta, q^{\mathsf{init}}, F)$ 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 $\overbrace{|z|+1}^{\geq n+1}$ 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.

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

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

Bsp. 2.14: Die Sprache $L_{\text{centered}} = \{0^n 1 0^n \mid n \in \mathbb{N}\}$ ist nicht regulär.

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

Sei n die Konstante aus dem PL. Wähle $z = 0^n 1 0^n$. (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 $k \geq 1$
- $w = 0^{n-m} 1 0^n$

Betrachte $uv^2w = 0^{m-k} 0^{2k} 0^{n-m} 1 0^n = 0^{n+k} 1 0^n \notin L$. Widerspruch! Somit ist L nicht regulär.

Zur Illustration:

$$
\underbrace{0 \ \ldots\ldots \ 0}_{n} 1 \underbrace{0 \ \ldots\ldots \ 0}_{n}
$$

$$
\vdash\!\text{u}\!\dashv\vdash\!\text{v}\!\dashv\!\text{———w———}\!\vdash
$$

2.5 ε-Transitionen

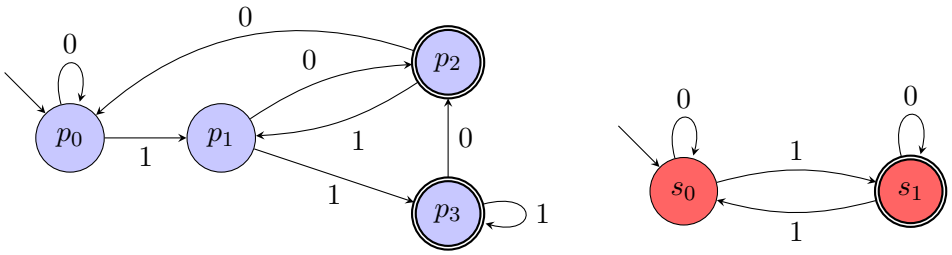
In diesem Unterkapitel führen wir mit dem ε-NEA ein weiteres Automatenmodell ein. Wir wollen ε-NEAs zunächst durch die folgende Fragestellung und anschließende Diskussion motivieren.

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

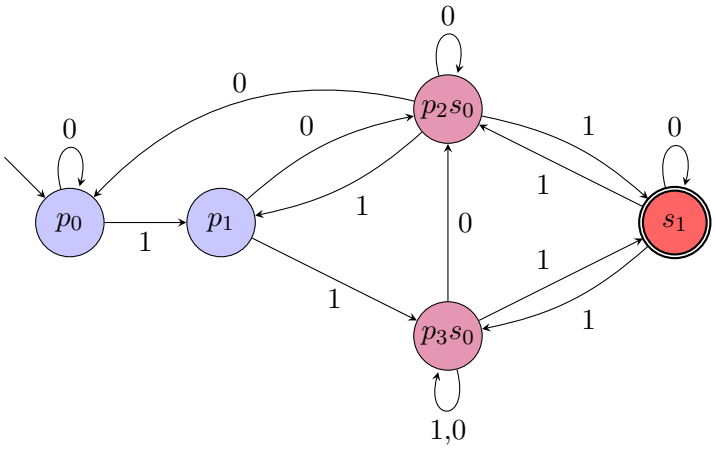
Idee: Gegeben DEA $\mathcal{A}_1$ mit $L(\mathcal{A}_1) = L_1$ und DEA $\mathcal{A}_2$ mit $L(\mathcal{A}_2) = L_2$, konstruiere NEA für $L_1 \cdot L_2$ durch „Hintereinanderschalten“ von $\mathcal{A}_1$ und $\mathcal{A}_2$; immer wenn wir in einem akzeptierenden Zustand von $\mathcal{A}_1$ sind, erlauben 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. 2.15: Links: DEA $\mathcal{A}_1$, der Automat aus Bsp. 2.11 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 $\mathcal{N}_{\mathsf{naiv}}$ aus der naiven und inkorrekten Konstruktion für die Konkatenation.



Dieser NEA akzeptiert nun auch das Wort $w = 11011$. Allerdings ist w nicht in der Konkatenation $L(\mathcal{A}_1) \cdot L(\mathcal{A}_2)$, 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 ε-Transition und definieren einen Automaten, der solche Zustandsübergänge haben kann, wie folgt.

Vorlesung: 19.05.2026

Def. 2.15 (ε-NEA)**:** Ein *nichtdeterministischer endlicher Automat mit ε-Transitionen* ist ein 5-Tupel

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

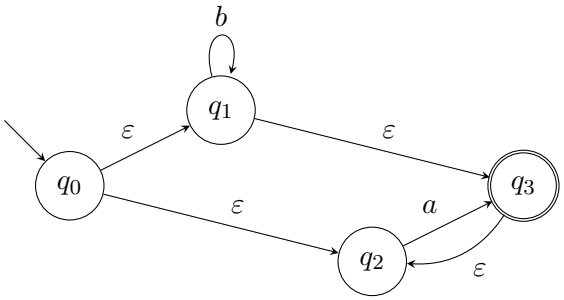
wobei Σ, Q, q^{init}, F wie bei NEAs (bzw. DEAs) definiert sind und die Transitionsfunktion den folgenden Typ hat.

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q) \qquad \diamond$$

Bemerkung: Wenn $f : A \to \mathcal{P}(B)$ ist, dann schreiben wir $\hat{f} : \mathcal{P}(A) \to \mathcal{P}(B)$ für die Fortsetzung von f auf Mengen. D.h. $\hat{f}(M) = \bigcup_{a \in M} f(a)$, falls $M \subseteq A$. Z.B. verwenden wir $\hat{\delta} : \mathcal{P}(Q) \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$.

Bemerkung: Funktionen $f, g : A \to \mathcal{P}(B)$ können bzgl. ihres Bildbereichs geordnet werden. Wir schreiben $f \subseteq g$, falls $\forall a \in A. f(a) \subseteq g(a)$. Diese Relation auf $A \to \mathcal{P}(B)$ ist eine partielle Ordnung mit kleinstem Element $f_0 : a \mapsto \emptyset$, die Funktion, die jedes $a \in A$ auf die leere Menge abbildet.

Bsp. 2.16: Zustandsdiagramm eines ε-NEA über dem Alphabet $\Sigma = \{a, b\}$.



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

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

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

Def. 2.16: Der *ε-Abschluss* $\mathsf{ecl}_{\mathcal{B}} : Q \to \mathcal{P}(Q)$ ist die kleinste Abbildung (bzgl. $\subseteq$), die für alle $q, q', q'' \in Q$ die folgenden Eigenschaften erfüllt:

$$q \in \mathsf{ecl}_{\mathcal{B}}(q) \qquad \frac{q' \in \mathsf{ecl}_{\mathcal{B}}(q) \qquad q'' \in \delta(q', \varepsilon)}{q'' \in \mathsf{ecl}_{\mathcal{B}}(q)}$$

◇

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

Bsp.: Für den ε-NEA aus Bsp. 2.16 sieht $\mathsf{ecl}_{\mathcal{B}}$ wie folgt aus:

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

Als Nächstes definieren wir eine mengenwertige Funktion, die uns für einen Zustand q und ein Wort w sagt, welche Zustände von q durch w erreichtbar sind. Der Name der Funktion „reach“ soll dabei an des englische Wort „reachability“ erinnern.

Def. 2.17: Die *Erreichbarkeitsfunktion* $\mathsf{reach}_{\mathcal{B}} : Q \times \Sigma^* \to \mathcal{P}(Q)$ ist definiert durch:

$$\begin{aligned}
\mathsf{reach}_{\mathcal{B}}(q, \varepsilon) &= \mathsf{ecl}_{\mathcal{B}}(q) \\
\mathsf{reach}_{\mathcal{B}}(q, a.w) &= \bigcup \{\mathsf{reach}_{\mathcal{B}}(q'', w) \mid q' \in \mathsf{ecl}_{\mathcal{B}}(q), q'' \in \delta(q', a)\} \\
&= \bigcup \{\mathsf{reach}_{\mathcal{B}}(q'', w) \mid q'' \in \hat{\delta}(\mathsf{ecl}_{\mathcal{B}}(q), a)\} \\
&= \widehat{\mathsf{reach}_{\mathcal{B}}}(\hat{\delta}(\mathsf{ecl}_{\mathcal{B}}(q), a), w)
\end{aligned}$$

◇

Für den ε-NEA aus Bsp. 2.16 können wir $\mathsf{reach}_{\mathcal{B}}$ 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 $q_j \in \mathsf{reach}_{\mathcal{B}}(q_i, w)$.

$\mathsf{reach}_{\mathcal{B}}$	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. 2.18: Ein Wort $w \in \Sigma^*$ wird von $\mathcal{B}$ *akzeptiert*, wenn $\mathsf{reach}_{\mathcal{B}}(q^{\mathsf{init}}, w) \cap F \neq \emptyset$. Die von $\mathcal{B}$ akzeptierte Sprache $L(\mathcal{B})$ ist die Menge der von $\mathcal{B}$ akzeptierten Wörter, d.h. $L(\mathcal{B}) = \{w \in \Sigma^* \mid \mathsf{reach}_{\mathcal{B}}(q^{\mathsf{init}}, w) \cap F \neq \emptyset\}$. ◇

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

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

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

Satz 2.8: Zu jedem ε-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.

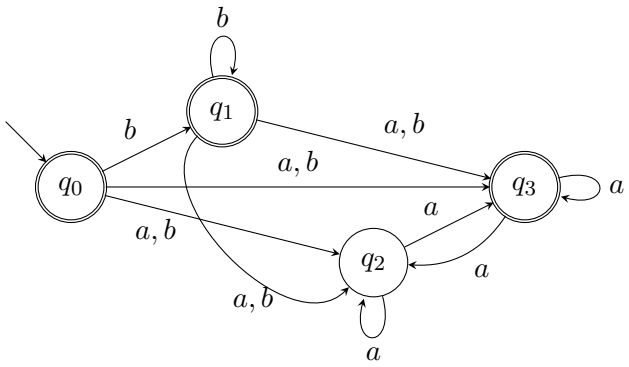
Def. 2.19 (ε-freier Automat)**:** Für einen gegebenen ε-NEA $\mathcal{B} = (\Sigma, Q, \delta, q^{\mathsf{init}}, F)$ definieren wir den NEA $\mathcal{N} = (\Sigma, Q, \delta_{\mathcal{N}}, q^{\mathsf{init}}, F_{\mathcal{N}})$ mit

$$\delta_{\mathcal{N}}(q, a) = \bigcup \{\mathsf{ecl}_{\mathcal{B}}(q'') \mid q'' \in \hat{\delta}(\mathsf{ecl}_{\mathcal{B}}(q), a)\}$$

$$F_{\mathcal{N}} = \{q \in Q \mid \mathsf{ecl}_{\mathcal{B}}(q) \cap F \neq \emptyset\}$$

und nennen diesen NEA den *ε-freien Automaten* von $\mathcal{B}$. ◇

Bsp. 2.17: Der ε-freie Automat für den ε-NEA aus Bsp. 2.16 hat das folgende Zustandsdiagramm.



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

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

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

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

$$\begin{aligned} w \in L(\mathcal{B}) &\overset{\text{Def. 2.3}}{\Leftrightarrow} \mathsf{reach}_{\mathcal{B}}(q^{\mathsf{init}}, w) \cap F \neq \emptyset \\ &\Leftrightarrow \exists q_f \in F_{\mathcal{N}} : \underbrace{q_0, q_1, \ldots, q_n}_{\text{Lauf}} \in Q, \text{ sodass } n = |w|, q_0 = q^{\mathsf{init}} \text{ und } q_n = q_f \\ &\overset{\text{Def. 2.18}}{\Leftrightarrow} w \in L(\mathcal{N}) \end{aligned}$$

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

Mit Hilfe der ε-NEAs greifen wir nun die zu Beginn von Abschnitt 2.5 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. 2.20: Gegeben zwei NEA-ε $\mathcal{B}_i = (\Sigma, Q_i, \delta_i, q_i^{\mathsf{init}}, F_i)$, $i = 1, 2$, definieren wir den

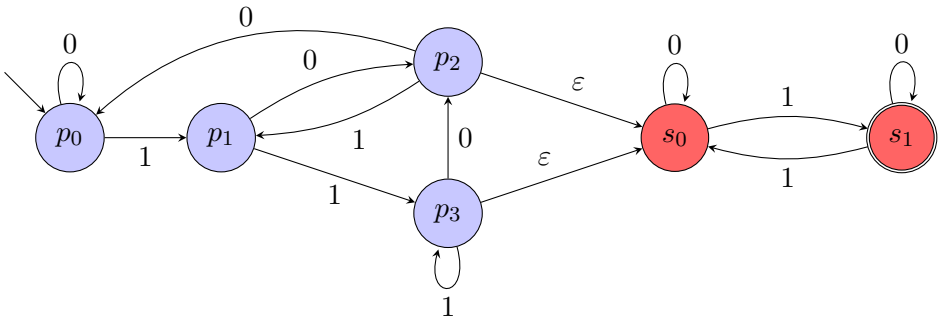
ε-NEA für Konkatenation $\mathcal{B} = (\Sigma, Q, \delta, q^{\mathsf{init}}, F)$ wie folgt.

$$Q = Q_1 \dot\cup Q_2$$

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \\ \delta_2(q,a) & q \in Q_2 \end{cases} \qquad \delta(q,\varepsilon) = \begin{cases} \delta_1(q,\varepsilon) \cup \{q_2^{\mathsf{init}}\} & q \in F_1 \\ \delta_1(q,\varepsilon) & q \in Q_1 \setminus F_1 \\ \delta_2(q,\varepsilon) & q \in Q_2 \end{cases}$$

$$q^{\mathsf{init}} = q_1^{\mathsf{init}}$$
$$F = F_2 \qquad \diamond$$

Bsp. 2.18: Der ε-NEA für Konkatenation für die beiden Automaten aus Bsp. 2.15 hat das folgende Zustandsdiagramm.



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

Beweis: Zeige mittels Induktion über 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.

$$q_1' \in \mathsf{reach}_{\mathcal{B}_1}(q_1, w_1) \text{ und } q_2' \in \mathsf{reach}_{\mathcal{B}_2}(q^{\mathsf{init}}, w_2) \quad \Leftrightarrow \quad q_2' \in \mathsf{reach}_{\mathcal{B}}(q_1, w_1 w_2)$$

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

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

2.6 Abschlusseigenschaften

Vorlesung: 20.05.2026

Def. 2.21: Eine Menge X heißt *abgeschlossen* unter Operation $f : X^n \to X$, falls $\forall x_1, \ldots, x_n \in X : f(x_1, \ldots, x_n) \in X$. $\diamond$

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 $\mathcal{A} = (\Sigma, Q, \delta, q^{\mathsf{init}}, F)$ der L akzeptiert. Wir konstruieren den DEA $\overline{\mathcal{A}} = (\Sigma, Q, \delta, q^{\mathsf{init}}, Q \backslash F)$ 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. □

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 ε-NEA $\mathcal{B} = (\Sigma, Q, \delta, q^{\mathsf{init}}, F)$, der L akzeptiert. Wir konstruieren einen NEA $\mathcal{B}_* = (\Sigma, Q \dot\cup \{q_*^{\mathsf{init}}\}, \delta_*, q_*^{\mathsf{init}}, F \dot\cup \{q_*^{\mathsf{init}}\})$ für L^*, indem wir

- einen neuen akzeptierenden Startzustand einführen,
- vom neuen Startzustand einen ε-Transition zum alten Startzustand hinzufügen und
- von jedem akzeptierenden Zustand einen ε-Transition zum alten Startzustand hinzufügen.

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

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

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
$\overline{}$	(Komplement)	Lemma 2.10
$\cdot$	(Konkatenation)	Lemma 2.9
*	(Kleene-Stern)	Lemma 2.11

2.7 Reguläre Ausdrücke

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

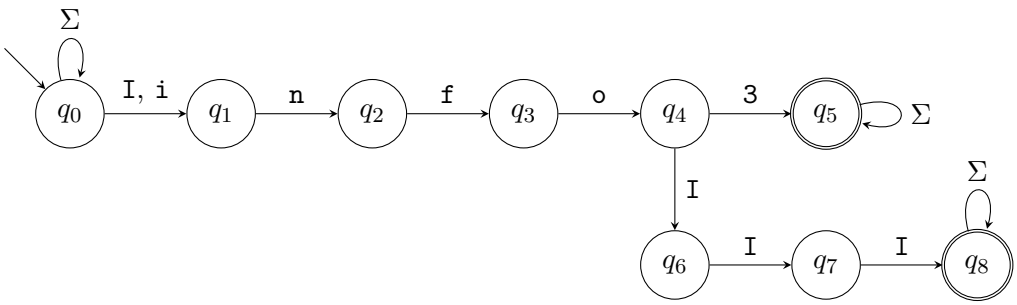
noch, dass die Zeichenkette `info3` oder `infoIII` enthalten ist. Möglicherweise war das `i` am Anfang aber auch groß geschrieben. Da Sie mehrere Dateien mit dieser Zeichenkette haben, soll der Rechner jeweils Dateinamen und die entsprechenden Zeilen ausgeben.

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

- $\underline{\emptyset} \in RE(\Sigma)$
- $\underline{\varepsilon} \in RE(\Sigma)$
- $a \in RE(\Sigma)$ 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. 2.19:

1. $(\underline{\emptyset\varepsilon})^* + a$ ist ein regulärer Ausdruck über $\Sigma = \{a\}$.
2. $\underline{\emptyset\varepsilon\emptyset\varepsilon}$ ist ein regulärer Ausdruck über jedem Alphabet.
3. $aaaabbbbbbb$ ist ein regulärer Ausdruck über $\Sigma = \{a, b\}$.
4. a^4b^7 ist **kein** regulärer Ausdruck
5. $(0 + 1 \cdot (01^*0)^* \cdot 1)^*$ ist ein regulärer Ausdruck über $\Sigma = \{0, 1\}$.
6. `(A+...+Z+a+...+z+0+...+9)*(I+i)nfo(3+III)(A+...+Z+a+...+z+0+...+9)*` ist ein regulärer Ausdruck[7] über dem Alphabet $\Sigma = \{\texttt{A}, \ldots, \texttt{Z}, \texttt{a}, \ldots, \texttt{z}, \texttt{0}, \ldots, \texttt{9}\}$.

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

Def. 2.23: Die durch einen regulären Ausdruck *beschriebene Sprache* $[\![\cdot]\!] : RE(\Sigma) \to \mathcal{P}(\Sigma^*)$ ist induktiv definiert durch:

$$\begin{aligned}
[\![\underline{\emptyset}]\!] &= \varnothing \\
[\![\underline{\varepsilon}]\!] &= \{\varepsilon\} \\
[\![a]\!] &= \{a\} \quad a \in \Sigma \\
[\![(r + s)]\!] &= [\![r]\!] \cup [\![s]\!] \\
[\![(r \cdot s)]\!] &= [\![r]\!] \cdot [\![s]\!] \\
[\![r^*]\!] &= [\![r]\!]^*
\end{aligned}$$ ◇

Bsp. 2.20:

$$[\![a + (\underline{\emptyset\varepsilon})^*]\!] \quad = \quad [\![a]\!] \cup ([\![\underline{\emptyset}]\!] \cdot [\![\underline{\varepsilon}]\!])^* \quad = \quad \{a\} \cup (\emptyset)^* \quad = \quad \{a\} \cup \{\varepsilon\} \quad = \quad \{a, \varepsilon\}$$

[7] Wir verwenden hier `A+...+Z` als Abkürzung für die Disjunktion aus 26 Zeichen. Der eigentliche reguläre Ausdruck besteht aus der Disjunktion; die drei Punkte kommen darin nicht vor.

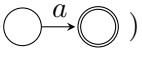
Bsp. 2.21: Die aus Bsp. 2.11 bekannte Sprache

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

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

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 $L = [\![r]\!]$. Zeige via strukturelle Induktion über den Aufbau regulärer Ausdrücke, dass $[\![r]\!]$ regulär ist.

I.A.:
- $r = \underline{\emptyset}$, $[\![r]\!] = \emptyset$ ist regulär (NEA: )
- $r = \underline{\varepsilon}$, $[\![r]\!] = \{\varepsilon\}$ ist regulär (NEA: )
- $r = a$, $[\![r]\!] = \{a\}$ ist regulär (NEA: )

I.V.: Für $i \in \{1,2\}$ gilt: $[\![r_i]\!]$ ist regulär.

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

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 „⇒“ von Satz 2.12). Um unsere Idee zu beschreiben, benötigen wir zunächst die folgende Definition.

Def. 2.24: Sei $\mathcal{A} = (\Sigma, Q, \delta, q^{\mathsf{init}}, F)$ 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. ◇

Im Folgenden betrachten wir einen beliebigen DEA und nummerieren dessen Zustände $Q = \{q_0, q_1, \ldots, q_n\}$. 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 $\emptyset$.

$$= \{\varepsilon \mid q_i \in F\} \cup \bigcup_{j=0}^{n} \underbrace{\{a \in \Sigma \mid \delta(q_i, a) = q_j\}}_{A_{ij} \not\ni \varepsilon} L_{q_j}$$

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

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

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

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

Bsp. 2.22: Wir wollen diese Gleichungen zunächst an einem Beispiel betrachten und verwenden dafür den aus Bsp. 2.13 bekannten DEA, der die Sprache der Binärcodierung von durch drei teilbaren Zahlen akzeptiert: $L = \{w \in \{0,1\}^* \mid \text{bin}(w) \equiv_3 0\}$.

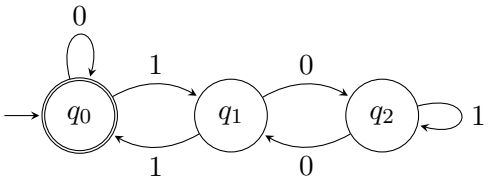


Abb. 4: DEA „modulo 3“

Wir erhalten das folgende Gleichungssystem mit drei Unbekannten.

$$\begin{aligned} r_0 &= \underline{\varepsilon} + 0 \cdot r_0 + 1 \cdot r_1 \\ r_1 &= 1 \cdot r_0 + 0 \cdot r_2 \\ r_2 &= 0 \cdot r_1 + 1 \cdot r_2 \end{aligned}$$

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 (Teil 1): Durch Einsetzen in die Gleichung rechnen wir nach, dass A^*B tatsächlich eine Lösung ist.

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

Daraus folgt $A^*B \subseteq X$. □

Beweis (Beschreibung des Agda-Beweises von `ardens-left`): Sei $\varepsilon \notin A$ und sei X eine Lösung der Gleichung $X = A \cdot X \cup B$. Der Agda-Beweis zeigt die Inklusion

$$X \subseteq A^*B.$$

Zusammen mit der bereits im manuellen Beweis verwendeten Inklusion $A^*B \subseteq X$ folgt daraus die Eindeutigkeit der Lösung.

Der eigentliche Induktionsbeweis steckt in der Hilfsaussage `ardens-left-gen`. Gezeigt wird die stärkere Aussage:

$$\forall n \in \mathbb{N} : \forall w \in \Sigma^* : |w| < n \text{ und } w \in X \quad \Rightarrow \quad w \in A^*B.$$

Wir führen eine Induktion über n durch.

I.A. ($n = 0$): Es gibt kein Wort w mit $|w| < 0$, also ist nichts zu zeigen.

I.S. ($n \rightsquigarrow n + 1$): Sei $w \in \Sigma^*$ mit $|w| < n + 1$ und $w \in X$. Wegen $X = A \cdot X \cup B$ gibt es zwei Möglichkeiten.

- Falls $w \in B$, dann ist $w = \varepsilon \cdot w$ mit $\varepsilon \in A^0$. Also gilt $w \in A^*B$.
- Falls $w \in A \cdot X$, dann gibt es Wörter u, v mit $w = uv$, $u \in A$ und $v \in X$. Der Fall $u = \varepsilon$ ist unmöglich, denn daraus würde $\varepsilon \in A$ folgen, im Widerspruch zur Annahme. Also ist $u \neq \varepsilon$. Dann ist v ein echtes Restwort von w; aus $|w| < n + 1$ folgt $|v| < n$. Nach Induktionsvoraussetzung gilt daher $v \in A^*B$. Also gibt es ein j sowie Wörter y, z mit $v = yz$, $y \in A^j$ und $z \in B$. Wegen $u \in A$ folgt $uy \in A^{j+1}$ und damit $w = uv = uyz \in A^{j+1}B \subseteq A^*B$.

Damit ist die Hilfsaussage bewiesen. Für `ardens-left` selbst wählt der Agda-Beweis zu einem beliebigen $w \in X$ den Wert $n = |w| + 1$. Dann gilt $|w| < n$, und die Hilfsaussage liefert sofort $w \in A^*B$. Also gilt $X \subseteq A^*B$. □

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

Korollar 2.14: Seien r_X, r_A, r_B reguläre Ausdrücke mit $\varepsilon \notin [\![r_A]\!]$, sodass die folgende Gleichung gilt:

$$[\![r_X]\!] = [\![r_A \cdot r_X + r_B]\!]$$

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

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

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

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

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

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

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

(Distributivgesetz: $R_{in}R_{nn}^*$)

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

(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 ε-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 $[\![r_0]\!] = L_{q_0} = L_{q^{\text{init}}} = L(\mathcal{A})$. □

Bsp. 2.23: Wir betrachten noch einmal Bsp. 2.22 und lösen das Gleichungssystem auf die im Beweis beschriebene Weise.

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

Ardens Lemma auf r_2 anwenden:

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

Einsetzen in r_1:

$$r_1 = \underbrace{1 \cdot r_0}_{B} + \underbrace{0 \cdot 1^* \cdot 0}_{A} \cdot r_1$$

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

Vorlesung: 02.06.2026

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

- Σ ist ein Alphabet, dessen Elemente wir *Terminalsymbole* nennen.
- N ist eine endliche Menge, $N \cap \Sigma = \emptyset$, 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 *Startsymbol*. ◇

Bsp. 3.1: $\mathcal{G} = (\Sigma, N, P, S)$ mit[8]

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

Def. 3.2 (Ableitungsrelation, Ableitung, Sprache einer Grammatik)**:** Die *Ableitungsrelation*[9] zu $\mathcal{G} = (\Sigma, N, P, S)$ ist $\cdot \vdash_{\mathcal{G}} \cdot \subseteq (N \cup \Sigma)^* \times (N \cup \Sigma)^*$ definiert durch

$$\frac{\alpha \to \beta \in P}{\gamma \cdot \alpha \cdot \delta \vdash_{\mathcal{G}} \gamma \cdot \beta \cdot \delta}$$

Die *Ableitung von β aus α*, geschrieben $\alpha \vdash_{\mathcal{G}}^* \beta$ ist induktiv definiert als die reflexive, transitive Hülle von „$\vdash_{\mathcal{G}}$“:

$$\frac{}{\alpha \vdash_{\mathcal{G}}^* \alpha} \qquad \frac{\alpha \vdash_{\mathcal{G}} \beta \quad \beta \vdash_{\mathcal{G}}^* \gamma}{\alpha \vdash_{\mathcal{G}}^* \gamma}$$

Falls $\alpha \vdash_{\mathcal{G}}^* \beta$, so ist β eine *Satzform von $\mathcal{G}$*. Oft wird die Ableitung durch eine Liste von Satzformen $\alpha_0, \ldots, \alpha_n$ angegeben mit $\alpha_i \vdash_{\mathcal{G}} \alpha_{i+1}$ für $0 \leq i < n$.

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

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

Wir nennen zwei Grammatiken *äquivalent*, falls sie die gleiche Sprache erzeugen. ◇

[8] P ist eine „normale“ binäre Relation, doch wir verwenden statt „$(x, y) \in P$“ meist „$x \to y$“, also einen Pfeil und Infix-Notation, um die Lesbarkeit zu erhöhen.

[9] 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: $1001 \in L(\mathcal{G})$.

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

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

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

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 mittels Induktion über die Ableitung $S \vdash_{\mathcal{G}}^* w$ zeigen. Der Beweis wird als Übung dem Leser überlassen.

Wir zeigen $L \subseteq L(\mathcal{G})$, d.h. $\forall w \in \Sigma^*.w \in L \Rightarrow w \in L(\mathcal{G})$. Hierzu definieren wir eine Hilfsfunktion $d : \Sigma^* \to \mathbb{Z}$:

$$d(\varepsilon) = 0 \qquad d(1w) = d(w) + 1 \qquad d(0w) = d(w) - 1$$

Mittels Induktion über u zeige, dass $u \in L \Leftrightarrow d(u) = 0$ und $d(u \cdot v) = d(u) + d(v)$.

Wir betrachten nun die folgende Aussage

$$P(n) = \forall w.\, |w| = n \Rightarrow w \in L \Rightarrow w \in L(\mathcal{G}).$$

und beweisen sie mit Hilfe des Beweisprinzips der vollständigen (bzw. starken) Induktion:

$$\frac{\forall n.\, (\forall j.\, j < n \Rightarrow P(j)) \Rightarrow P(n)}{\forall n.\, P(n)}$$

Die Ausformulierung des Beweises erfolgt in der Übung.

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

$$\Sigma = \{a, b, c\}$$
$$N = \{S, B, C\}$$
$$P = \{S \to aSBC,\ S \to aBC,\ CB \to BC,\ aB \to ab, bB \to bb, bC \to bc, cC \to cc\}$$

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 $\alpha \to \beta \in P$ ist $|\alpha| \leq |\beta|$. Ausnahme: falls S nicht in einer rechten Regelseite auftritt, dann ist $S \to \varepsilon$ erlaubt.
- Eine Grammatik heißt *Typ-2* oder *kontextfrei*, falls alle Regeln die Form $A \to \alpha$ 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:

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

Eine Sprache heißt Typ-i-Sprache, falls es eine Typ-i-Grammatik für sie gibt. ◇

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 ε-freie[10] 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 CHi die Menge der Typ-i Sprachen; dann gilt: $CH3 \subsetneq CH2 \subsetneq CH1 \subsetneq CH0$.

3.1 Kontextfreie Sprachen

Vorlesung: skipped

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

$$P = \{E \to a \mid E + E \mid E * E\}.^{11}$$

[10] Auf keiner rechten Seite steht ε; Ausnahme: Startsymbol S; vgl. Elimination von ε-Produktionen.

[11] Notation: Wenn wir mehrere Regeln mit gleicher linker Seite wie z.B. „$A \to \alpha$“ und „$A \to \beta$“ haben, dann dürfen wir diese auch mit Hilfe eines senkrechten Striches als „$A \to \alpha \mid \beta$“ schreiben.

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

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

Bsp. 3.6: Syntax von Programmiersprachen. Wir zeigen hier nur exemplarisch Produktionen für einige typische Bestandteile einer Programmiersprache.[12] Wörter in spitzen Klammern sind hier Nichtterminalsymbole. Terminalsymbole sind grau unterlegt.

<Stmt> → <Var> `=` <Exp>
| <Stmt> `;` <Stmt>
| `if` `(` <Exp> `)` <Stmt> `else` <Stmt>
| `while` `(` <Exp> `)` <Stmt>

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

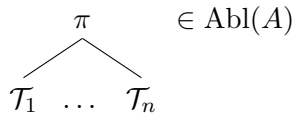
- $E \vdash_{\mathcal{G}} E + E \vdash_{\mathcal{G}} a + E \vdash_{\mathcal{G}} a + a$
- $E \vdash_{\mathcal{G}} E + E \vdash_{\mathcal{G}} E + a \vdash_{\mathcal{G}} a + a$

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.

Vorlesung: 03.06.2026

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

Falls $\pi = A \to w_0 A_1 w_1 \dots A_n w_n \in P$ mit $A_i \in N$, $w_i \in \Sigma^*$, $0 \le i \le n$ und $\mathcal{T}_i \in \mathrm{Abl}(A_i)$, dann ist



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

[12] Sie finden am Ende der Java Language Specification `https://docs.oracle.com/javase/specs/` auf ca. 25 Seiten die kontextfreie Grammatik für diese Programmiersprache.

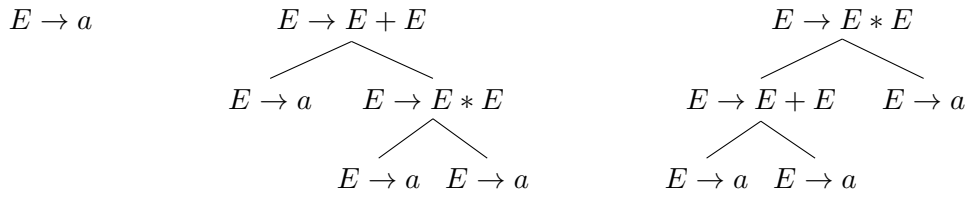
Das *abgeleitete Wort zu einem* $\mathcal{T} \in \mathrm{Abl}(A)$, $Y : \mathrm{Abl}(A) \to \Sigma^*$[13], ist definiert durch

$$Y\left(\begin{array}{c}\pi\\ \diagup\ \diagdown\\ \mathcal{T}_1 \quad \ldots \quad \mathcal{T}_n\end{array}\right) = w_0 Y(\mathcal{T}_1) w_1 \ldots Y(\mathcal{T}_n) w_n$$

wobei $\pi = A \to w_0 A_1 w_1 \ldots A_n w_n \in P$. ◇

Bemerkung: Nach dieser Definition gibt es für jede Regel π, bei der nur Terminalsymbole auf der rechten Seite vorkommen, einen Ableitungsbaum, der die ein-elementige Knotenmenge $\{\pi\}$ und eine leere Menge von Kanten hat.

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.

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

(Ohne Beweis.)

3.2 Die Chomsky-Normalform für kontextfreie Sprachen

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

- Es gibt einen einfachen Algorithms für das Wortproblem ($w \in L(\mathcal{G})$?).
- Jede kontextfreie Grammatik lässt sich effektiv in CNF transformieren.

Def. 3.5 (CFG in CNF)**:** Eine kontextfreie Grammatik (CFG) $\mathcal{G} = (\Sigma, N, P, S)$ ist in Chomsky-Normalform (CNF), falls jede Regel die Form $A \to a$, $A \to BC$, oder $S \to \varepsilon$ hat, wobei $A, B, C \in N$, $a \in \Sigma$. Falls $S \to \varepsilon \in P$, dann darf S auf keiner rechten Seite einer Regel vorkommen. ◇

[13] Wir verwenden $Y(\mathcal{T})$, um an das englische Wort „yield“ zu erinnern.

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.

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

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 Formen hat.

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

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

- $N' = N \dot\cup \{Y_a \mid a \in \Sigma\}$[14]
- $P' = \{Y_a \to a \mid a \in \Sigma\} \cup \{A \to \beta[a \to Y_a] \mid A \to \beta \in P\}$.

  Die Schreibweise $\beta[a \to Y_a]$ bedeutet hier, dass alle $a \in \Sigma$, die in β 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 einmal über alle Regeln und ersetze jeweils $a \mapsto Y_a$.
- Für jedes neue Nichtterminalsymbol fügen wir eine Regel der Größe 2 hinzu.

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

BEWEIS: Ersetze jede Regel der Form

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

[14] „$\dot\cup$“ steht für „disjunkte Vereinigung“

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

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

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

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

$$\text{Nullable}(\mathcal{G}) = \{A \in N \mid A \overset{*}{\vdash}_{\mathcal{G}} \varepsilon\}.$$ ◇

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 ε mit einem Ableitungsbaum der Höhe $< i$ ableiten lässt:[15]

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

Es gilt für alle $i \in \mathbb{N}$, dass $M_i \subseteq M_{i+1} \subseteq N$. 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 daher $\bigcup_{i\in\mathbb{N}} M_i$ mit dem Algorithmus in Abb. 5 in $O(|\mathcal{G}|^3)$ berechnen.

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

„$\supseteq$“ Wir zeigen mittels Induktion, dass

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

[15] Im Folgenden ist mit $*$ bei M_i^* und M^* der Kleene-Stern gemeint.

```
M = {}
repeat
    changed = false
    M_old = M
    foreach A → α ∈ P:
        if (A ∉ M ∧ α ∈ M*_old):
            M = M ∪ {A}
            changed = true
until not changed
return M
```

Abb. 5: Berechung von 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 existiert $A \to A_1 \dots A_n \in P$ mit $A_j \in M_i$ für alle $1 \leq j \leq n$. Nach IV existieren Ableitungsbäume $\mathcal{T}_j$ für $A_j \vdash_{\mathcal{G}}^* \varepsilon$, die zu einem Ableitungsbaum von A nach ε kombiniert werden können:

$$\mathcal{T} = \left\{ \begin{array}{c} A \to A_1 \dots A_n \\ \diagup \quad \dots \quad \diagdown \\ \mathcal{T}_1 \quad \dots \quad \mathcal{T}_n \end{array} \right.$$

Wegen $Y(\mathcal{T}_j) = \varepsilon$, $1 \leq j \leq n$, gilt auch $Y(\mathcal{T}) = \varepsilon$. Also gilt $A \in \text{Nullable}(\mathcal{G})$.

„$\subseteq$“

Wir zeigen mittels Induktion über einen Ableitungsbaum von $A \vdash_{\mathcal{G}}^* \varepsilon$, dass $\exists i.\, A \in M_i$.

IA $A \to \varepsilon$ ist eine Produktion. Also ist $A \in M_1$.

IS Die Wurzel des Ableitungsbaums ist mit $A \to A_1 \dots A_n$ beschriftet und die Teilbäume entsprechen den Ableitungen $A_j \vdash_{\mathcal{G}}^* \varepsilon$, $1 \leq j \leq n$.

Die IV liefert für jeden Teilbaum ein i_j, sodass $A_j \in M_{i_j}$. Also gilt für $i = \max\{i_1, \dots, i_n\}$, dass $A_j \in M_i$. Daher ist $A \in M_{i+1}$. □

Vorlesung: 09.06.2026

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

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

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 ε-Regel $S' \to \varepsilon$ ist (falls $\varepsilon \in L(\mathcal{G})$) und bei der S' auf keiner rechten Seite einer Regel vorkommt.

BEWEIS:

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

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 Regel der Form $A \to B \in P$ mit $A, B \in N$ heißt *Kettenregel*. ◇

Lemma 3.7 (UNIT)**:** Es gibt einen Algorithmus der für eine CFG $\mathcal{G}$ in $O(|\mathcal{G}|^3)$ eine äquivalente CFG $\mathcal{G}'$ ohne Kettenregeln mit $|\mathcal{G}'| \in O(|\mathcal{G}|^2)$ berechnet.

BEWEIS: Beweisskizze zum Algorithmus in Abb. 6.

- Zur Äquivalenz:

  Wir zeigen, dass jeder Schritt, der die CFG ändert, eine äquivalente CFG liefert. Dazu zeigen wir, dass jedes Wort, das in der alten CFG abgeleitet werden konnte, auch in der neuen CFG abgeleitet werden kann und umgekehrt. Für Schritt 6 gilt die Äquivalenz sogar für jede Iteration der mittleren for-Schleife.

- Zur Kettenregelfreiheit:

  In Schritt 3 wird mindestens ein Kettenregelzykel entfernt. Da eine Grammatik nur endlich viele Regeln hat, wird nach spätestens $|\mathcal{G}|$ Anwendungen von Schritt 3 der Schritt 5, in dem alle Zyklen entfernt sind, erreicht.

  In der mittleren for-Schleife von Schritt 6 gilt aufgrund der topologischen Sortierung immer $i < k$. Die äußere Schleife hat die Invariante, dass am Ende jedes Durchlaufs für $n \geq i \geq j$ keine Kettenregel $A_i \to \ldots$ in P' existiert. Beim Verlassen der Schleife ist $j = 1$ und es existieren überhaupt keine Kettenregeln mehr.

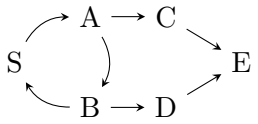
- Zu Laufzeit und Größe des Resultates:

  Vor Schritt 6 wird die Grammatik nie vergrößert (wichtig sowohl für Laufzeit als auch für Größe!) und die Kosten übersteigen $O(|\mathcal{G}|^3)$ nicht. Die Anzahl der Schleifendurchläufe ist jeweils durch $|\mathcal{G}|$ beschränkt. Seien $B \to \alpha_1, \ldots, B \to \alpha_n$ alle neuen Regeln, die für eine Variable $B \in N$ in Schritt 6 hinzugefügt werden. Dann gilt, dass die Summe der Größe der α_i die Größe der Grammatik $|\mathcal{G}|$ nicht übersteigt. Somit ist die Größe des Resultates durch $|\mathcal{G}|^2$ beschränkt. □

Bsp. 3.8: Sei $\Sigma = \{0, 1\}$. Betrachte die CFG $\mathcal{G} = (\Sigma, \{S, A, B, C, D, E\}, P, S)$ deren Regeln P wie folgt definiert sind.

$$
\begin{aligned}
S &\to A \\
A &\to B \mid 0A1B \mid C \\
B &\to S \mid 1B0A \mid D \\
C &\to E \\
D &\to E \\
E &\to \varepsilon
\end{aligned}
$$

Nach der erstmaligen Anwendung von Schritt 2 erhalten wir den folgenden gerichteten Graphen.



[17] gerichteter azyklischer Graph (DAG) (engl. *directed acyclic graph*).

[17] Topologisches Sortieren ist ein aus der Informatik 2 Vorlesung bekannter Begriff und bezeichnet das Erweitern einer gegebenen partiellen Ordnung zu einer totalen Ordnung.

Eingabe: CFG $\mathcal{G} = (\Sigma, N, P, S)$

Ausgabe: CFG $\mathcal{G}' = (\Sigma, N', P', S)$, die äquivalent zu $\mathcal{G}$ ist, aber keine Kettenregeln enthält.

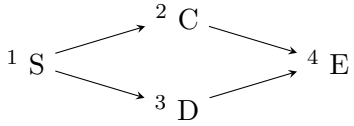
1. Setze $N' = N$ und $P' = P$.
2. Betrachte den gerichteten Graphen G, dessen Knotenmenge N und Kantenmenge $\{(A, B) \mid A \to B \in P'\}$ ist. (Es gibt also genau eine Kante pro Kettenregel.)
3. Suche, z.B. mittels Tiefensuche, einen Zyklus in G. Wenn kein Zyklus gefunden wurde, fahre mit Schritt 5 fort.
4. Wurde ein Zyklus $A_1 \to A_2 \to \ldots \to A_k \to A_1$ gefunden, dann sei o.B.d.A. $A_1 = S$ falls S im Zyklus enthalten. Ersetze nun in P' alle Vorkommen von A_j mit $j > 1$ durch A_1 (auf linker *und* rechter Regelseite). Entferne dann alle Regeln der Form $A \to A$. Fahre fort mit Schritt 2.
5. Sortiere die Knoten des nun azyklischen Graphen[16] G topologisch[17] als $A_1, \ldots, A_n$, sodass A_n auf keiner linken Seite einer Kettenregel vorkommt.
6. **for** $j = n - 1, \ldots, 1$
   for each $A_j \to A_k \in P'$
   entferne $A_j \to A_k$ aus P'
   for each $A_k \to \alpha \in P'$
   füge $A_j \to \alpha$ zu P'

Abb. 6: Algorithmus UNIT

In Schritt 2 wählen wir den Zykel $S \to A \to B \to S$ und erhalten für P' die folgenden den Regeln.

$$\begin{aligned} S &\to C \mid D \mid 0S1S \mid 1S0S \\ C &\to E \\ D &\to E \\ E &\to \varepsilon \end{aligned}$$

Nach dem zweiten Anwenden von Schritt 2 erhalten wir den folgenden gerichteten Graphen und wählen in Schritt 5 die durch die Zahlen angedeutete topologische Sortierung.



Am Ende von Schritt 6 sieht die Regelmenge P' wie folgt aus[18].

$$\begin{aligned} S &\to \varepsilon \mid 0S1S \mid 1S0S \\ C &\to \varepsilon \\ D &\to \varepsilon \\ E &\to \varepsilon \end{aligned}$$

Satz 3.8: Es gibt einen Algorithmus der für eine CFG $\mathcal{G}$ in $O(|\mathcal{G}|^3)$ Rechenschritten eine äquivalente CFG $\mathcal{G}'$ in CNF mit $|\mathcal{G}'| \in O(|\mathcal{G}|^2)$ berechnet.

Beweis: (Beweisskizze) Ein Algorithmus der diese Eigenschaften erfüllt ist das Hintereinanderschalten von Sep, Bin, Del und Unit.[19]

- Zur Korrektheit:

  Wir zeigen dass jedes dieser vier Verfahren die vom vorherigen Verfahren etablierte Eigenschaft nicht zerstört und dass die vier etablierten Eigenschaften für die CNF hinreichend sind.

[18] Das Beispiel zeigt dass das Resultat des Algorithmus völlig unnötige Regeln enthalten kann.
[19] Es genügt sogar nur Del und dann Unit anzuwenden, da Del schon Sep und Bin ausführt.

- Zu Laufzeit und Größe des Resultates:

  Die ersten drei Verfahren erhöhen die Größe der Grammatik nur linear und die Laufzeiten sind durch $O(|\mathcal{G}|^3)$ beschränkt. Für UNIT sind die in diesem Satz genannten Kosten eine obere Schranke. □

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 ε-freie Typ-2-Grammatik. Diese ist nach Definition auch eine Typ-1-Grammatik. □

3.3 Wortproblem und Leerheitsproblem für kontextfreie Sprachen

Satz 3.10: Es gibt einen Algorithmus, der für eine CFG $\mathcal{G}$ in CNF und ein Wort w in $O(|w|^3 \cdot |\mathcal{G}|)$ Schritten und mit $O(|w|^2 \cdot |\mathcal{G}|)$ Speicherplatz entscheidet ob $w \in L(\mathcal{G})$ gilt.

BEWEIS (CYK-Algorithmus): Der CYK-Algorithmus[20] in Abb. 7 erfüllt diese Eigenschaften. Wir wollten zunächst die Idee des Algorithmus erklären.

Sei $w = a_1 \dots a_n \in \Sigma^*$. Definiere für $1 \leq i \leq j \leq n$:

$$M_{ij} = \{A \in N \mid A \vdash_{\mathcal{G}}^* a_i \dots a_j\} \qquad \text{(EntryDeriv)}$$

Es gilt: $w \in L(\mathcal{G})$ genau dann, wenn $S \vdash_{\mathcal{G}}^* w$ genau dann, wenn $S \in M_{1n}$.

Es gilt für $i = j$

$M_{ii} = \{A \mid A \to a_i \in P\}$

Es gilt für $i < j$

$$\begin{aligned} M_{ij} &= \{A \mid A \to BC \in P \text{ und } BC \overset{*}{\vdash}_{\mathcal{G}} a_i \dots a_j\} \\ &= \{A \mid A \to BC \in P \text{ und } \exists k \in \{i, \dots j-1\} \text{ sodass } B \overset{*}{\vdash}_{\mathcal{G}} a_i \dots a_k \text{ und } C \overset{*}{\vdash}_{\mathcal{G}} a_{k+1} \dots a_j\} \\ &= \{A \mid A \to BC \in P \text{ und } \exists k \in \{i, \dots j-1\} \text{ sodass } B \in M_{ik} \text{ und } C \in M_{(k+1)j}\} \end{aligned}$$

Vorlesung: 10.06.2026

Beobachtung: Der Wert von M_{ij} hängt nur von $M_{i'j'}$ mit $i' > i, j' \leq j$ oder $i' \geq i, j' < j$ ab. Wir können die M_{ij} also mit dem Verfahren aus Abb. 7 berechnen.

[20] Benannt nach Cocke, Younger und Kasami, die den Algorithmus unabhängig voneinander entdeckt haben. Manchmal auch YCK.

Eingaben: CFG $\mathcal{G} = (\Sigma, N, P, S)$ in CNF und Wort $a_1 \dots a_n \in \Sigma^*$

Ausgabe: $a_1 \dots a_n \in L(\mathcal{G})$?

Sei M eine $n \times n$-Matrix mit (initial) $M_{ij} = \varnothing$ für alle i, j // $O(n^2)$

```
for i = 1,...,n do // O(|G|) · O(n)
  M_ii = {A | A → a_i}
for i = n − 1,...,1 do
  for j = i + 1,...,n do
    for k = i,...,j − 1 do // O(|G|) · O(n^3)
      M_ij = M_ij ∪ {A | A → BC, B ∈ M_ik, C ∈ M_(k+1)j}
return S ∈ M_1n
```

Abb. 7: CYK Algorithmus

Skizze der verbleibenden Beweisschritte.

Zur Korrektheit: Zeige mittels einer verschachtelten Induktion über Zeilen und Spalten, dass am Ende der zweiten for-Schleife für alle $M_{i'j'}$ mit $i' \in \{i, \dots, n-1\}$ und $j' \in \{i+1, \dots, j\}$ die oben definierte Gleichung (EntryDeriv) gilt.

Zur Laufzeitabschätzung: Die Iterationen jeder for-Schleife sind durch die Länge des Wortes $|w|$ beschänkt. In der innersten Schleife genügt es ein mal über die Regeln der Grammatik zu iterieren.

Zum Speicherplatz: Wir benötigen weniger als $|w|^2$ Matrixeinträge und jeder Eintrag enthält höchstens $|N|$ Variablen. □

Bsp. 3.9: Sei $\Sigma = \{a, b, c\}$. Betrachte das Wort $w = aaabbbcc$ und die CFG $\mathcal{G} = (\Sigma, \{S, X, Y\}, P, S)$[21] deren Regeln P wie folgt definiert sind.

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

Damit wir den CYK Algorithmus anwenden können transformieren wir $\mathcal{G}$ in eine äquivalente Grammatik in CNF. Z.B. in die CFG $\mathcal{G}' = (\Sigma, \{S, X, Y, Z, A, B, C\}, P', S)$, deren

[21] Es gilt $L(\mathcal{G}) = \{a^n b^n c^m \mid n, m \geq 1\}$ aber das bestimmen der erzeugten Sprache ist für das Entscheiden des Wortproblemes natürlich nicht erforderlich.

Regeln P' wie folgt definiert sind.

$$S \to XY$$
$$X \to AB \mid AZ$$
$$Z \to XB$$
$$Y \to CY \mid c$$
$$A \to a, B \to b, C \to c$$

Wir wenden den CYK Algorithmus an und erhalten die folgende Matrix.

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

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 $S \in M_{1n}$, gilt $w \in L(\mathcal{G}')$.

Korollar 3.11: Es gibt einen Algorithmus, der für eine CFG $\mathcal{G}$ und ein Wort w in $O(|w|^3 \cdot |\mathcal{G}|^2)$ Schritten und mit $O(|w|^2 \cdot |\mathcal{G}|^2)$ Speicherplatz entscheidet, ob $w \in L(\mathcal{G})$. ⋄

Beweis: Nach Satz 3.8 kann eine CFG in CNF konvertiert werden, sodass deren Größe höchstens quadratisch wächst. Wende anschließend den CYK Algorithmus auf das Resultat an. □

Satz 3.12: Es gibt einen Algorithmus, der das Leerheitsproblem[22] für kontextfreie Grammatiken in $O(|\mathcal{G}|^2)$ entscheidet.

Def. 3.9: Sei $\mathcal{G} = (\Sigma, N, P, S)$ eine CFG. Ein Nichtterminalsymbol $A \in N$ ist *co-erreichbar*, falls es ein Wort $w \in \Sigma$ existiert, sodass $A \overset{*}{\vdash}_{\mathcal{G}} w$ gilt. ⋄

Beweis: Die M die Menge der co-erreichbaren Nichtterminale von $\mathcal{G}$ ist gegeben durch

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

Dann gilt: $L = \varnothing \Leftrightarrow S \notin M$.

[22] Gefragt ist, ob $L(\mathcal{G}) = \{w \in \Sigma^* \mid S \overset{*}{\vdash} w\} \overset{?}{=} \varnothing$.

Wir können M wie folgt berechnen:

$$M_0 = \emptyset$$
$$M_{i+1} = \{A \mid A \to \alpha \in P, \alpha \in (\Sigma \cup M_i)^*\}$$

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

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

Bem.: Wenn wir in einer Grammatik $\mathcal{G}$ ein nicht co-erreichbares Nichtterminalsymbol A und alle Regeln, bei denen A auf der linken oder rechten Regelseite auftritt, entfernen, ändert sich $L(\mathcal{G})$ nicht. Wir können eine gegebene Grammatik also optimieren, indem wir alle nicht co-erreichbaren Nichtterminalsymbole entfernen.

3.4 Das Pumping Lemma für kontextfreie Sprachen

Satz 3.13 (Pumping Lemma für CFL)**:** Sei L eine kontextfreie Sprache. Dann gilt:

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

Lemma 3.14: In jedem Binärbaum mit $\geq 2^k$ Blättern gibt es mindestens ein Blatt sodass der einen Pfad von der Wurzel zu diesem Blatt Länge $\geq k$ hat. (ohne Beweis)

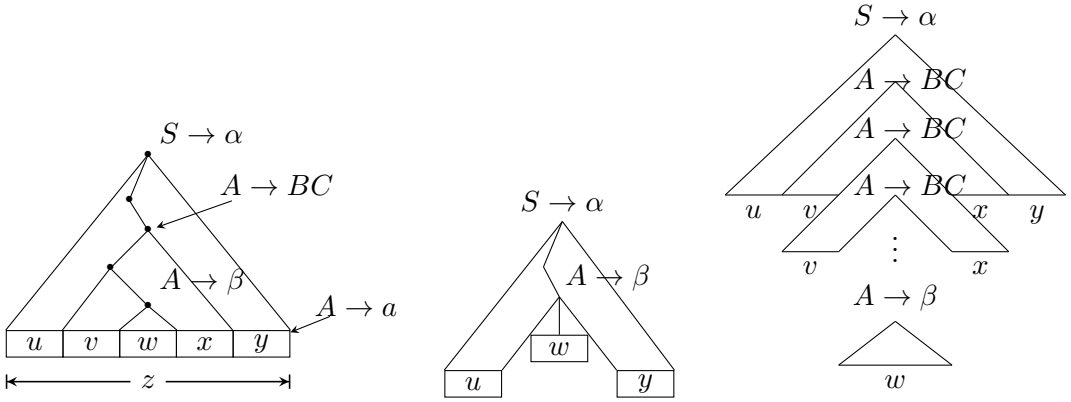
Beweis (von Satz 3.13): [23] Sei $\mathcal{G} = (\Sigma, N, P, S)$ in CNF mit $L(\mathcal{G}) = L$. Wähle $n = 2^{|N|}$.

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 3.14 ein Pfad ζ der Länge $\geq |N|$. Auf ζ liegen $\geq |N| + 1$ Knoten. Jeder Knoten ist mit einer Regel beschriftet deren linke Seite ein Nichtterminalsymbol ist. Es muss also mindestens ein Nichtterminalsymbol A mehrmals als linke Seite einer Regel vorkommen.

Folge ζ Rückwärts vom Blatt in Richtung Wurzel, bis sich eine linke Regelseite A das erste Mal wiederholt. Das geschieht nach $\leq |N|$ Schritten. Teile $z = uvwxy$ wie in Abb. 8a skizziert. Es gilt:

[23] Dieser Beweis wurde in der Vorlesung mit Hilfe von Abb. 8 erklärt aber nicht an die Tafel geschrieben.

- $|vx| \geq 1$, da ζ entweder durch den B-Nachfolger oder durch C-Nachfolger läuft. Angenommen, ζ verläuft durch den B-Nachfolger. Somit muss $C \overset{*}{\vdash} x$ gelten. In CNF ist $C \overset{*}{\vdash} \varepsilon$ nicht möglich und somit ist $|x| \geq 1$. Der Fall, dass ζ durch den C-Nachfolger verläuft, ist analog.
- $|vwx| \leq 2^{|N|} = n$, da der obere Knoten dessen linke Regelseite mit A beschrftet ist höchstens $|N|$ Schritte von der Blattebene entfernt und $\mathcal{T}$ ein Binärbaum ist.
- Für die Aussage $\forall i \in \mathbb{N} : uv^i wx^i y \in L$ 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. 8b).
  - $i = 1$: 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 $uv^i wx^i y$ mit $i \geq 2$ erzeugen (siehe auch Abb. 8c).



(a) Zerlegung $z = uvwxy$ **(b)** $uwy \in L$ **(c)** $uv^i wx^i y \in L$

Abb. 8: Illustration des Argumentes aus dem Beweis von Satz 3.13

□