Vorlesung
Informatik III – Theoretische Informatik

Formale Sprachen, Berechenbarkeit, Komplexitätstheorie

Matthias Heizmann

Basierend auf einem Mitschrieb von Ralph Lesch*

der von Prof. Dr. Peter Thiemann im WS 2015/16 gehaltenen Vorlesung

WS 2017/18

Zuletzt aktualisiert: 2017-11-25

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

Inhaltsverzeichnis

1 Vorspann: Sprachen

Vorlesung: 18.10.2017

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

Zeichen sind hier beliebige abstrakte Symbole.

Bsp.: für Alphabete, die in dieser Vorlesung, im 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.[1] Im Folgenden sei Σ immer ein beliebiges Alphabet.[2]

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

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

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

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

[1] Dies ist eine Konventionen analog zu den folgenden, die Sie möglicherweise in der Schule befolgten: Verwende n, m für natürliche Zahlen. Verwende α, β für Winkel in Dreiecken. Verwende A für Matrizen.

[2] Dieser Satz dient dazu, dass die Autoren dieses Skripts nicht jede Definition mit "Sei Σ ein Alphabet..." beginnen müssen.

[3] Wir schreiben also z.B. `einhorn` statt `e,i,n,h,o,r,n`.

[4] Eine analoge Konvention, die sie aus der Schule kennen: Verwende immer π als Symbol für die Kreiszahl.

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

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

Bsp.:

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

Eigenschaften von „$\cdot$":

- assoziativ
- ε ist neutrales Element
- *nicht* kommutativ

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

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

Bemerkung: Die Zeichenfolge $\texttt{rambo}\varepsilon$ 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. 1.4: 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. 1.5: Eine *Sprache* über Σ ist eine Menge $L \subseteq \Sigma^*$. ◇

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

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

und die *Potenzierung* von U induktiv definiert durch

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

Bsp.:

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

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

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

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

2 Reguläre Sprachen und endliche Automaten

Vorlesung: 20.10.17

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

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

Ein weiteres Beispiel ist die folgende Zeile.

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

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 Entscheidunsproblem 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*)

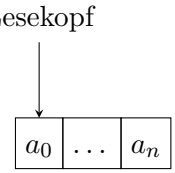


Abb. 1: Endliches Band

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 q aus *endlicher* Zustandsmenge Q

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

Akzeptierende Zustände $F \subseteq Q$

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

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

Bsp. 2.1: Aufgabe:
„Erkenne alle Stapel von Macarons, in denen höchstens ein grünes Macaron vorkommt.“


[5]

Ein passendes Alphabet wäre $\Sigma = \{\texttt{grün}, \texttt{nicht-grün}\}$. 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.

[5] 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

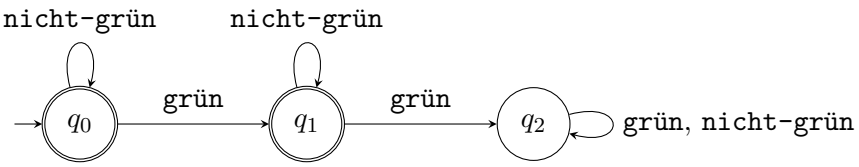
Def. 2.1 (DEA)**:** Ein *deterministischer endlicher Automat (DEA)*, (DFA $\hat{=}$ deterministic finite automaton) ist ein 5-Tupel

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

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.

Def. 2.2: 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. $\tilde{\delta}(q, aw) = \tilde{\delta}(\delta(q, a), w)$ (Rest im Folgezustand verarbeiten) ◇

Def. 2.3: 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.,

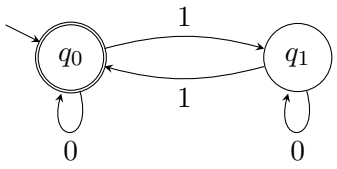
$$L(\mathcal{A}) = \{w \in \Sigma^* \mid \tilde{\delta}(q^{\mathsf{init}}, w) \in F\}.$$

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

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

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

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

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

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

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

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

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

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

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

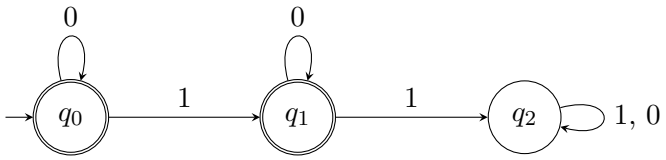
Schließlich zeigen wir $L(A_\cap) = L(A_1) \cap L(A_2)$ mit Hilfe der folgenden Umformungen für ein beliebiges $w \in \Sigma^*$.

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

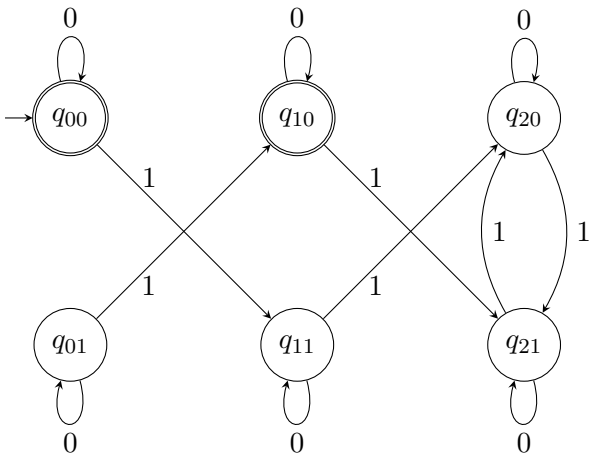
□

Vorlesung: 25.10.17

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



Bsp. 2.3: Der Produktautomat für Schnitt 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.



2.2 Minimierung endlicher Automaten

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

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

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

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

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

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

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

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

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

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

2.2.1 Exkurs: Äquivalenzrelationen

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

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

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

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

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

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

[7] `http://home.mathematik.uni-freiburg.de/junker/ss17/matheII.html`

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

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

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

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

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

Fakt 1 Die Äquivalenzklassen von R sind paarweise disjunkt.

Fakt 2 Die Vereinigung aller Äquivalenzklassen ist die Menge X.

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

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

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

Bsp. 2.6: Für den DEA aus Bsp. 2.3 hat die Relation $\equiv$ drei Äquivalenzklassen.[8]

$$[q_{00}] = \{q_{00}, q_{10}\},$$
$$[q_{01}] = \{q_{01}\},$$
$$[q_{11}] = \{q_{11}, q_{20}, q_{21}\}$$

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.

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

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

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

Beweis:

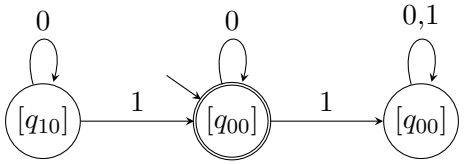
$$\begin{array}{lll} p \equiv q & \text{gdw} & \forall w \in \Sigma^* : \tilde{\delta}(p,w) \in F \Leftrightarrow \tilde{\delta}(q,w) \in F \\ & \text{gdw} & (p \in F \Leftrightarrow q \in F) \wedge \forall a \in \Sigma : \forall w \in \Sigma^* : \tilde{\delta}(p,aw) \in F \Leftrightarrow \tilde{\delta}(q,aw)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

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:

$$Q_\equiv = \{[q] \mid q \in Q\} \qquad \delta_\equiv([q],a) = [\delta(q,a)]$$
$$q^{\mathsf{init}}_\equiv = [q^{\mathsf{init}}] \qquad F_\equiv = \{[q] \mid q \in F\}$$ ◇

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



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

Beweis:

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

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

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

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

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

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

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

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 $\mathcal{A}_{\equiv}$ der kleinste DEA ist, der $L(\mathcal{A})$ akzeptiert.

Vorlesung: 27.10.17

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

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

Bsp. 2.8: 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. 2.9: Für eine Sprache $L \subseteq \Sigma^*$ ist die *Nerode-Relation* wie folgt definiert.

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

Beobachtung 1 Die Nerode-Relation ist eine Äquivalenzrelation.

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

Beobachtung 2 Die Nerode-Relation ist rechtskongruent.

Beweis: Sei $(u, v) \in R_L$. Zeige $\forall w \in \Sigma^* : (uw, vw) \in R_L$. Wir führen diesen Beweis via Induktion über die Länge von w.
I.A. $(n = 0)$ Für $w = \varepsilon$ ist $(u\varepsilon, v\varepsilon) = (u, v) \in R_L$.
I.S. $(n \rightsquigarrow n + 1)$ Betrachte mit $w = w'a$ ein beliebiges Wort der Länge n. Nach Induktionsvoraussetzung ist dann auch $(uw', vw') \in R_L$.

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

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

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

Bsp. 2.10: Für ein beliebiges Alphabet 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. 2.11: 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:

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

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

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

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

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

Beweis: Wir beweisen die paarweise Äquivalenz in drei Schritten:

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

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

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

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

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

Es genügt zu zeigen, dass die Nerode-Relation R_L eine Obermenge von R ist.[9]

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

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
- $\delta([w], a) = [wa]$ wohldefiniert, da R_L rechtskongruent
- $q^{\mathsf{init}} = [\varepsilon]$
- $F = \{[w] \mid w \in L\}$

[9] 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.

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

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

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

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

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

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

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

Vorlesung: 3.11.16

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 $\mathcal{A}'$ 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.

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

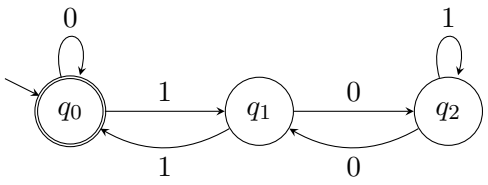
Für einen beliebigen DEA A' ist $|Q|$ also nie größer als $|Q'|$. □

2.3 Pumping Lemma (PL) für reguläre Sprachen

Welche interessanten Eigenschaften haben reguläre Sprachen?

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



Beobachtungen:

- Es gilt offensichtlich, dass $11 \in L$.
- Es gilt auch, dass $1\underline{00}1 \in L$.
- 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 2.6 (Pumping Lemma)**:** Sei L eine reguläre Sprache. Dann gilt:

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

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.

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

Es folgt für beliebiges $i \in \mathbb{N}$: $\tilde{\delta}(q^{\mathsf{init}}, uv^i w) = \tilde{\delta}(q, v^i w)$
$= \tilde{\delta}(q, w)$ denn $\forall i : \tilde{\delta}(q, v^i) = q$
$\in F$ □

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

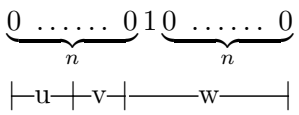
Wir geben hierfür einen Widerspruchsbeweis mit Hilfe des Pumping Lemmas PL.

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

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

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

Zur Illustration:



2.4 Nichtdeterministischer endlicher Automat (NEA)

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

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

Naiver Lösungsversuch:

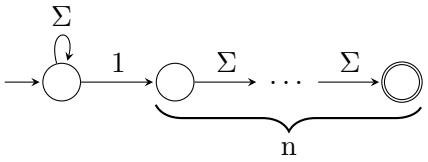


Abb. 2: Nichtdeterministischer Automat für L_n

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. 2.10 (NEA)**:** Ein *nichtdeterministischer endlicher Automat (NEA)*, (NFA $\hat{=}$ nondeterministic finite automaton) 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 also nur in der Transitionsfunktion. Beim DEA ist der Bildbereich der Transitionsfunktion die Menge der Zustände Q. Beim NEA ist der Bildbereich die Potenzmenge $\mathcal{P}(Q)$ der Zustandsmenge Q. Analog zu DEAs werden wir auch NEAs mit Hilfe eines Zustandsdiagramms beschrieben. Zum Beispiel beschreibt Abb. 2 für jedes $n \in \mathbb{N}$ einen NEA für die Sprache L_n.

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

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

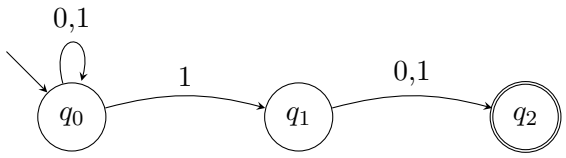
Vorlesung: 8.11.17

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

Bsp. 2.14: Der NEA für die Sprache

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

hat die folgende graphische Repräsentation.



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

Bemerkung: Die Definitionen von NEA und DEA in der Literatur sind nicht einheitlich. Es gibt äquivalente NEA-Definitionen, die statt der Transitionsfunktion $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ 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.

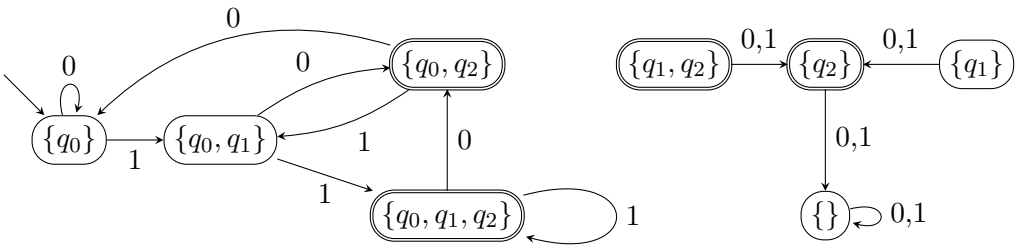
Satz 2.7 (Rabin und Scott)**:** Zu jedem NEA $\mathcal{N}$ mit n Zuständen gibt es einen DEA $\mathcal{A}_\mathcal{P}$ mit 2^n Zuständen, sodass $L(\mathcal{A}_\mathcal{P}) = L(\mathcal{N})$ gilt.

Zur Vorbereitung des Beweises machen wir zunächst die folgende Definition.

Def. 2.13 (Potenzmengenautomat)**:** Für einen gegebenen NEA $\mathcal{N} = (\Sigma, Q, \delta, q^{\mathsf{init}}, F)$ 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. 2.15: Der Potenzmengenautomat für den NEA aus Bsp. 2.14 hat das folgende Zustandsdiagramm.



Die vier Zustände auf der rechten Seite sind nicht erreichbar.

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

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

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

I.A. ($n = 0$, also $w = \varepsilon$): Gilt trivialerweise, da $p \neq \{\}$.
I.S. ($n \rightsquigarrow n+1$): Sei $w = aw'$ ein beliebiges Wort der Länge $n+1$.

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

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

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

□

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 2.7).
3. Zu jedem DEA mit n Zuständen gibt es einen NEA mit n Zuständen, der die gleiche Sprache akzeptiert.

2.4.1 ε-Übergänge

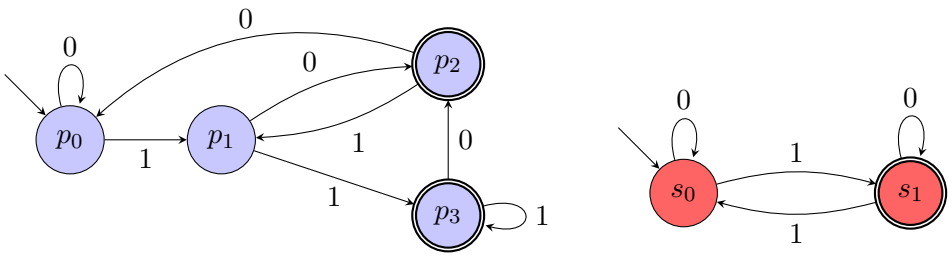
In diesem Unterkapitel führen wir mit dem ε-NEA ein weiteres Automatenmodell ein. Wir wollen ε-NEAs zunächst durch die folgende Fragestellung und anschließende Diskussion motivieren.

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

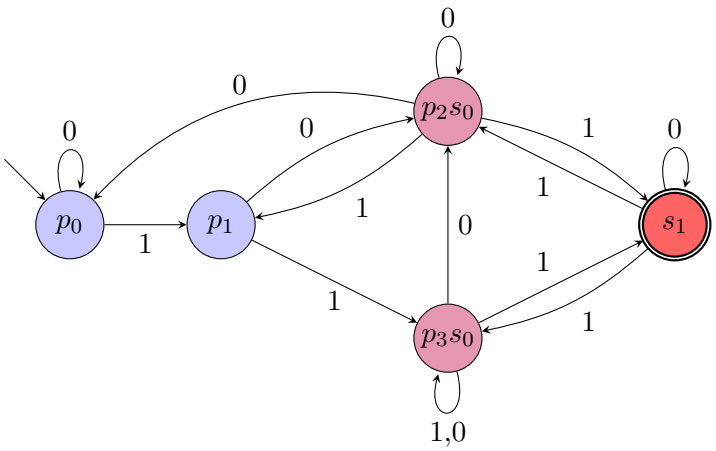
Idee: Gegeben DEA $\mathcal{A}_1$ mit $L(\mathcal{A}_1) = L_1$ und DEA $\mathcal{A}_2$ mit $L(\mathcal{A}_2) = L_2$, konstruiere NEA für $L_1 \cdot L_2$ durch „Hintereinanderschalten“ von $\mathcal{A}_1$ und $\mathcal{A}_2$; immer wenn wir in einem akzeptierenden Zustand von $\mathcal{A}_1$ sind, erlauben 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. 2.16: Links: DEA $\mathcal{A}_1$, der Automat aus Bsp. 2.14 eingeschränkt auf die erreichbaren Zustände. Rechts: DEA $\mathcal{A}_2$ mit der Sprache $\{w \in \{0,1\}^* \mid \text{Anzahl 1 in } w \text{ ungerade}\}$.



Unten: NEA $\mathcal{N}_{\mathsf{naiv}}$ 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-Übergang und definieren einen Automaten, der solche Zustandsübergänge haben kann, wie folgt.

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

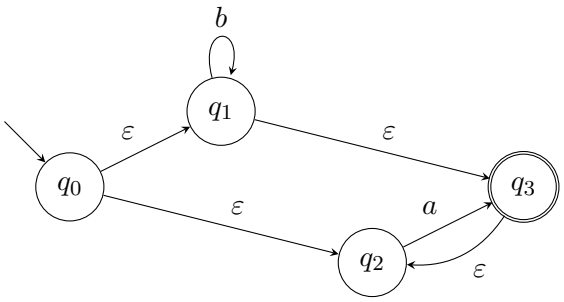
$$\mathcal{B} = (\Sigma, Q, \delta, q^{\mathsf{init}}, F)$$

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

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

Vorlesung: 10.11.17

Bsp. 2.17: 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-Übergänge 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. 2.15: Der *ε-Abschluss* $\mathsf{ecl}_{\mathcal{B}} : Q \to \mathcal{P}(Q)$ ist die kleinste Abbildung, die für alle $q, q', q'' \in Q$ die folgenden Eigenschaften erfüllt:

$$q \in \mathsf{ecl}_{\mathcal{B}}(q)$$
$$q' \in \mathsf{ecl}_{\mathcal{B}}(q) \text{ und } q'' \in \delta(q', \varepsilon) \quad \Rightarrow \quad q''' \in \mathsf{ecl}_{\mathcal{B}}(q) \qquad \diamond$$

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

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

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

Für den ε-NEA aus Bsp. 2.17 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_i, w, q_j) in $\mathsf{reach}_{\mathcal{B}}$ enthalten ist.

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

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

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

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.

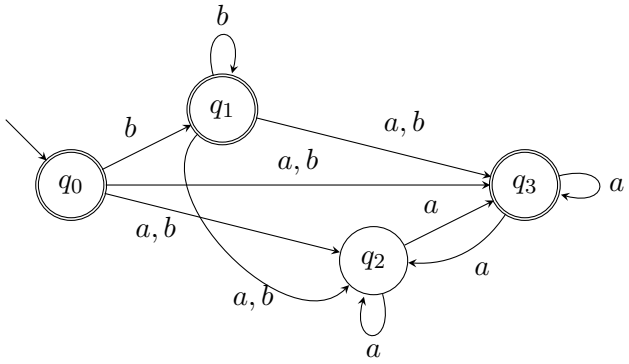
Def. 2.18 (ε-freier Automat)**:** Für einen gegebenen ε-NEA $\mathcal{B} = (\Sigma, Q, \delta, q^{\mathsf{init}}, F)$ definieren wir den NEA $\mathcal{N} = (\Sigma, Q, \delta_{\mathcal{N}}, q^{\mathsf{init}}, F_{\mathcal{N}})$ mit

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

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

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

Bsp. 2.18: Der ε-freie Automat für den ε-NEA aus Bsp. 2.17 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 via Induktion über die Länge von w in den Übungen zeigen werden.

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

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

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

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

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

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

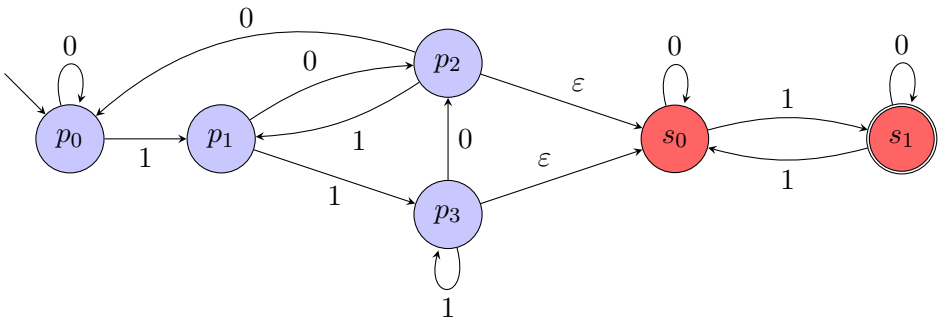
Wir geben hierfür zunächst eine Konstruktion an.

Def. 2.19: Gegeben zwei ε-NEA $\mathcal{B}_i = (\Sigma, Q_i, \delta_i, q_i^{\mathsf{init}}, F_i)$, $i = 1, 2$, definieren wir den *ε-NEA für Konkatenation* $\mathcal{B} = (\Sigma, Q, \delta, q^{\mathsf{init}}, F)$ wie folgt.

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

◇

Bsp. 2.19: Der ε-NEA für Konkatenation für die beiden Automaten aus Bsp. 2.16 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 via Induktion über die Länge von w_1, dass $\forall w_1, w_2 \in \Sigma^*, \forall q_1 \in Q_1$, $\forall q_1' \in F, \forall q_2' \in Q_2$ die folgende Eigenschaft gilt.

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

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

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

2.5 Abschlusseigenschaften

Vorlesung: 15.11.17

Def. 2.20: 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. $\square$

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

BEWEIS: Sei L eine reguläre Sprache. Dann gibt es einen 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-Übergang zum alten Startzustand hinzufügen und
- von jedem akzeptierenden Zustand einen ε-Übergang zum alten Startzustand hinzufügen.

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

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

2.6 Reguläre Ausdrücke

2.6.1 Motivation

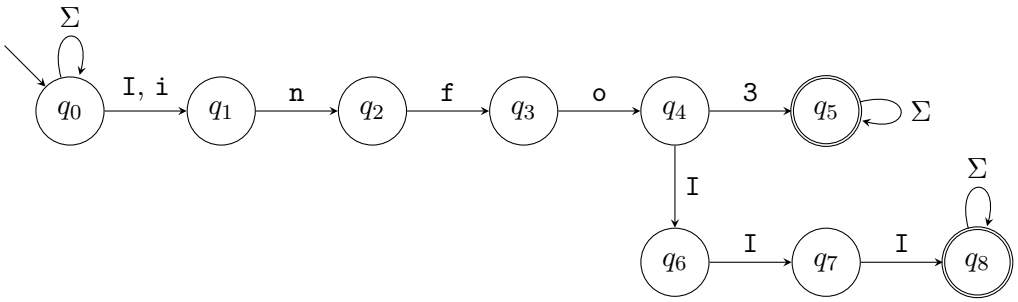
Problemstellung: Sie haben auf Ihrem Rechner irgendwo eine Textdatei mit Notizen zu dieser Vorlesung, können sich aber gerade nicht an den Pfad erinnern. Sie wissen aber noch, dass die Zeichenkette `info3` oder `infoIII` enthalten ist. Möglicherweise war das `i` am Anfang aber auch groß geschrieben. Da Sie mehrere Dateien mit dieser Zeichenkette haben, soll der Rechner jeweils Dateinamen und die entsprechenden Zeilen ausgeben.

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

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

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

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



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

Def. 2.21: Die Menge $RE(\Sigma)$ der *regulären Ausdrücke über* 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. 2.20:

1. `(A+...+Z+a+...+z+0+...+9)*(I+i)nfo(3+III)(A+...+Z+a+...+z+0+...+9)*` ist ein regulärer Ausdruck[10] über dem Alphabet $\Sigma = \{\mathtt{A}, \ldots, \mathtt{Z}, \mathtt{a}, \ldots, \mathtt{z}, \mathtt{0}, \ldots, \mathtt{9}\}$.
2. $(\underline{\emptyset}\underline{\varepsilon})^* + a$ ist ein regulärer Ausdruck über $\Sigma = \{a\}$.

[10] 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.

3. $\underline{\emptyset\varepsilon\emptyset\varepsilon}$ ist ein regulärer Ausdruck über jedem Alphabet.
4. $aaaabbbbbbb$ ist ein regulärer Ausdruck über $\Sigma = \{a, b\}$.
5. a^4b^7 ist **kein** regulärer Ausdruck
6. $(0 + 1 \cdot (01^*0)^* \cdot 1)^*$ ist ein regulärer Ausdruck über $\Sigma = \{0, 1\}$.

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

Def. 2.22: Die durch einen regulären Ausdruck *beschriebene Sprache* $[\![\cdot]\!] : RE(\Sigma) \to \mathcal{P}(\Sigma^*)$ 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. 2.21:

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

Bsp. 2.22: Die aus Bsp. 2.14 bekannte Sprache

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

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

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
- $r = \underline{\varepsilon}$, $[\![r]\!] = \{\varepsilon\}$ ist regulär
- $r = a$, $[\![r]\!] = \{a\}$ 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. 2.23: Sei $\mathcal{A} = (Q, \Sigma, \delta, q^{\mathsf{init}}, F)$ 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:

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

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

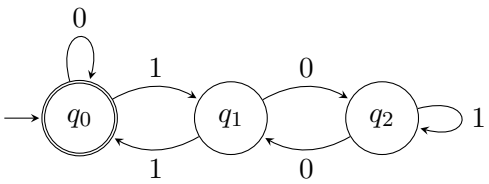


Abb. 3: DEA „modulo 3“

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.

Vorlesung: 17.11.2017

Lemma 2.13 (Ardens Lemma)**:**
Für die Gleichung $X = A \cdot X \cup B$ über den Sprachen $A, B, X \subseteq \Sigma^*$ gilt:

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

Beweis:

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

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

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

   Angenommen es gäbe Wörter x mit dieser Eigenschaft. Dann gibt es auch Wörter minimaler Länge mit dieser Eigenschaft. Sei w solch ein Wort minimaler Länge. Da alle Lösungen von der Form $\underbrace{A^nX}_{\ni w} \cup \underbrace{A^{n-1}B \cup \cdots \cup AB \cup B}_{\not\ni w}$ sind, hat w die Form $w = u_1 \dots u_n w'$ mit $u_1, \dots, u_n \in A$ und $w' \in X$. Fallunterscheidung:

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

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

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

Korollar 2.14: Seien r_X, r_A, r_B reguläre Ausdrücke mit $\varepsilon \notin [\![r_A]\!]$, 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 (TODO refer to) iterativ lösen. Wir beginnen mit Gleichung r_n:

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

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:

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

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

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

(Zusammenlegen der Summen und Ausklammern von r_j)

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

Nach diesen Umformungen ergeben sich 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 $[\![r_0]\!] = L_{q_0} = L_{q^{\mathsf{init}}} = L(\mathcal{A})$. □

Bsp. 2.24: Wir betrachten noch einmal Bsp. 2.23 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

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

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

Bsp. 3.1: $\mathcal{G} = (\Sigma, N, P, S)$ mit[11]

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

Def. 3.2 (Ableitungsrelation, Ableitung, Sprache einer Grammatik)**:** Sei $\mathcal{G} = (\Sigma, N, P, S)$ eine Grammatik. Die *Ableitungsrelation*[12] zu $\mathcal{G}$ ist

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

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

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

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

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

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

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

[11] 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.

[12] Analog zur Relation P verwenden wir auch für die Ableitungsrelation wieder Infix-Notation, also $\alpha \vdash \beta$ statt $(\alpha, \beta) \in \vdash$.

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

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

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

Vorlesung: 22.11.17

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

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 via Induktion über die Länge der Ableitung von $S \overset{*}{\vdash}_{\mathcal{G}} w$ zeigen. Der Beweis wird als Übung dem Leser überlassen.

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

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

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

Wir zeigen nun via Induktion über n die folgende Eigenschaft:

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

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

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

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

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

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

Sei $w' = a_1 \dots a_n$. Betrachte die Folge $d_0, \dots, d_n$ mit $d_0 = 0$ und $d_i = d(a_1 \dots a_i)$ für $1 \leq i \leq n$. Wähle $0 \leq i < n$ maximal, sodass für alle $0 \leq j \leq i$ gilt: $d_j < 1$.[13]

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

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

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

[13] Wir wissen, dass $i < n$, da $d_n = d(w') = 1$.

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

Es folgt mit den Produktionsregeln $S \vdash_{\mathcal{G}} 0S1S \overset{*}{\vdash}_{\mathcal{G}} 0w_1 1S \overset{*}{\vdash}_{\mathcal{G}} 0w_1 1w_2$.

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:

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

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

Beobachtung: 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[14] Typ-2-Grammatik transformiert werden kann. Diese erfüllt dann auch alle Bedingungen einer Typ-1-Grammatik.
- Jede Typ-1-Grammatik ist auch eine Typ-0-Grammatik.

Im Lauf der Vorlesung werden wir die folgende Aussage zeigen:
Sei $\mathcal{M}_i$ die Menge der Typ-i Sprachen; dann gilt: $\mathcal{M}_3 \subsetneq \mathcal{M}_2 \subsetneq \mathcal{M}_1 \subsetneq \mathcal{M}_0$.

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

3.1 Kontextfreie Sprachen

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

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

$$P = \{E \to a \mid E + E \mid E * E\}.$$[15]

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.[16] 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:

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

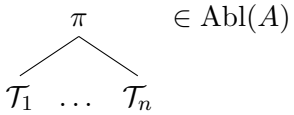
Beide Ableitungen unterscheiden sich nur durch die Reihenfolge, in der Variablen ersetzt werden. Die folgende Definition erlaubt es uns, beide Ableitungen durch das gleiche Objekt, den sogenannten Ableitungsbaum, darzustellen.

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

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

[15] 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.

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



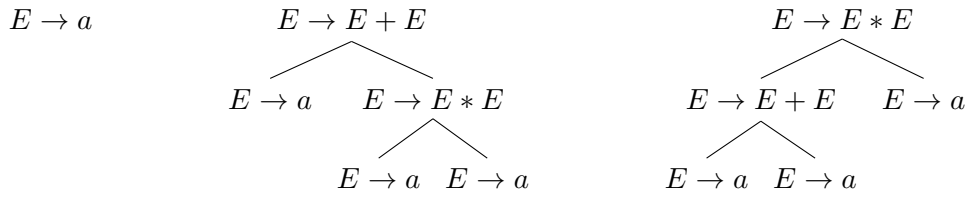
Da es manchmal aufwendig ist, Bäume zu zeichnen, verwenden wir alternativ die folgende Notation: $\pi(\mathcal{T}_1, \ldots, \mathcal{T}_n)$

Das *abgeleitete Wort zu einem* $\mathcal{T} \in \mathrm{Abl}(A)$, $Y(\mathcal{T})$[17], ist definiert durch

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

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

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.)

Def. 3.5:

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

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

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

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

3.2 Die Chomsky-Normalform für kontextfreie Sprachen

Vorlesung: 24.11.2017

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

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

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

- $N' = N \dot\cup \{Y_a \mid a \in \Sigma\}$[18]
- $P' = \{Y_a \to a \mid a \in \Sigma\} \cup \{A \to \beta[a \to Y_a] \mid A \to \beta \in P\}$.

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

- Laufe für jedes Terminal einmal über alle Regeln.
- Für jedes Nichtterminal bekommen wir eine Regel der Größe 2 zusätzlich

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

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

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

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

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

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

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

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:[19]

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

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

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

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

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

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

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

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

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

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

IS $i \rightsquigarrow i + 1$

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

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

  $$A_j \overset{*}{\vdash} \varepsilon$$

  sodass auch eine Ableitung von A nach ε existiert:

  $$A \vdash A_1 \dots A_n \overset{*}{\vdash} \underbrace{\varepsilon \dots \varepsilon}_{n \text{ mal}} = \varepsilon$$

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

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

IA $m = 1$. Die Ableitung ist $A \vdash \varepsilon$, sodass $A \in M_1$.

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

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

Korollar 3.5: Man kann zu einer CFG $\mathcal{G}$ berechnen, 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. 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.
2. Wende erst Sep, dann Bin an. Nun hat jede rechte Regelseite die Form $a \in \Sigma$ oder ε oder A oder BC.
3. 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.
4. Falls $S \in \text{Nullable}(\mathcal{G})$, füge $S' \to \varepsilon$ hinzu
5. 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 *Kettenregel* von $\mathcal{G}$ ist eine Regel in P der Form $A \to B$, wobei $A, B \in N$. ◇

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

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

1. Betrachte den gerichteten Graphen G mit Knoten N und Kanten $\{(A, B) \mid A \to B \in P'\}$.
2. Suche, z.B. mittels Tiefensuche, einen Zyklus in G. Wenn kein Zyklus gefunden wurde, fahre mit Schritt 4 fort.
3. Wurde der Zyklus $A_1 \to A_2 \to \ldots \to A_k \to A_1$ gefunden, dann ersetze in P' alle Vorkommen von A_j mit $j > 1$ durch A_1 (auf linker *und* rechter Regelseite). Entferne dann alle Regeln der Form $A \to A$. Fahre fort mit Schritt 1.
4. Der Graph G ist ein gerichteter azyklischer Graph (DAG) (engl. *directed acyclic graph*). Sortiere die Knoten von G topologisch als $A_1, \ldots, A_n$, sodass A_n auf keiner linken Seite einer Kettenregel vorkommt.
5. **for** $j = n - 1, \ldots, 1$

   for each $A_j \to A_k \in P'$

   (wegen topologischer Sortierung gilt $k > j$)

   entferne $A_j \to A_k$ aus P'

   füge $A_j \to BC$ zu P' hinzu, falls $A_k \to BC \in P', \quad B, C \in N$.

   füge $A_j \to a$ zu P' hinzu, falls $A_k \to a \in P', \quad a \in \Sigma$.

   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. □

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

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

Vorlesung: 29.11.2017

Def. 3.9: Eine 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. ◇

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

Beweis: Wir erhalten eine äquivalente CFG in CNF, indem wir der Reihe nach Sep, Bin, Del und Unit anwenden.[20] □

[20] Es genügt sogar nur Del und dann Unit anzuwenden, da Del schon Sep und Bin ausführt.

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

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

Beweis: Zu jeder Typ-2 Grammatik existiert eine äquivalente ε-freie Typ-2 Gammatik □

Beobachtung: Ist $\mathcal{G} = (\Sigma, N, P, S)$ in CNF, dann lassen sich für die Ableitungsbäume $\mathcal{T} \in \text{Abl}(S)$ folgende Eigenschaften feststellen:

1. $\mathcal{T}$ ist ein Binärbaum.
2. Falls $Y(\mathcal{T}) = w$, so ist die Anzahl der Blätter des Baumes $|w|$.

3.3 Entscheidungsprobleme für kontextfreie Sprachen

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

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

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

Es gilt für $i = j$

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

Es gilt für $i < j$

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

Es gilt:

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

Wir können die M_{ij} wie folgt ermitteln:

CYK$(\mathcal{G}, a_1, \dots, a_n)$
M $n \times n$-Matrix mit $M_{ij} = \varnothing$ für alle i, j

```
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
```

□

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

$$S \to XY$$
$$X \to ab \mid aXb$$
$$Y \to c \mid cY$$

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

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

$w =$	a	a	a	b	b	b	c	c
$M =$	A	•	•	•	•	X 6	S 7	S 8
		A	•	•	X 4	Z 5	•	•
			A	X 2	Z 3	•	•	•
				B	•	•	•	•
					B	•	•	•
						B	•	•
							C, Y	Y 1
								C, Y

Satz 3.11: L ist regulär $\Leftrightarrow$ L ist Type-3 Sprache

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

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

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

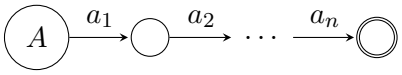
zeige $L(\mathcal{G}) = L(\mathcal{A})$.

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

Def. $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ NFA

$$\begin{aligned} Q &= N \\ q_0 &= S \\ A \to aB \in P &\qquad \delta(A,a) \ni B \\ A \to a, \ldots a_n &\qquad n = 0 \Rightarrow A \in F \end{aligned}$$

$n > 0$: n weitere Zustände erforderlich, letzter Endzustand



Zeige noch $L(\mathcal{G}) = L(\mathcal{A})$ □

Lemma 3.12: $\mathcal{L}_3 \subsetneq \mathcal{L}_2$

Beweis: Betrachte $L = \{a^n b^n \mid n \in \mathbb{N}\}$
Bekannt, dass L nicht regulär. Aber es gibt eine Typ-2 Grammatik für L:

$$\mathcal{G} = (\{S\}, \{a, b\}, \{S \to \varepsilon, S \to aSb\}, S)$$

Korrektheitsbeweis $L = L(\mathcal{G})$ ist dem Leser zur Übung selbst überlassen. □

Liste der Definitionen

Liste der Sätze

Abbildungsverzeichnis

Abkürzungsverzeichnis

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

Anmerkungsverzeichnis